Interoperability between .Net framework and Python in Component way

M. K. Pawar[1], Dr. Ravindra Patel[2] and Dr. N. S. Chaudhari[3]

**[1] Assistant Professor,
UIT, RGPV, Bhopal**

**[2] Associate Professor,
UIT, RGPV, Bhopal**

**[3] Professor, Deptt. Of CSE
IIT, Indore**

Abstract

The objective of this work is to make interoperability of the distributed object based on CORBA middleware technology and standards. The distributed objects for the client-server technology are implemented in C#.Net framework and the Python language. The interoperability result shows the possibilities of application in which objects can communicate in different environment and different languages. It is also analyzing that how to achieve client-server communication in heterogeneous environment using the OmniORBpy IDL compiler and IIOP.NET IDLtoCLS mapping. The results were obtained that demonstrate the interoperability between .Net Framework and Python language. This paper also summarizes a set of fairly simple examples using some reasonably complex software tools**.**

Keywords: *Component Interoperability, Component objects, cross communication among .NET and Python.*
.

1. Introduction and Background

There is an increasing demand of development of component based technology in software industries [1]. For the reason that in another engineering discipline in which, components have successfully developed and also have adapted to build the systems. (For example, Civil engineering, Mechanical engineering Electronics engineering etc.). As a result we can also think the same concept in software engineering. CORBA is continuously progressing in the research area of Component based software engineering. Since, CORBA middleware makes available the common platform [2] for various oops based language, some of the languages are very powerful in terms of compatibility of CORBA and some languages are less supportive the CORBA middleware. In the past few years, component-based software's have been well developed and motivated, for example Enterprise Java beans EJB of Sun Microsystems, CORBA Component Model of the OMG (Object Management Group) and COM (Component object model), DCOM and COM+ of Microsoft. Still there is a need to lot of development in CORBA standards and services for language compatibility in component based technology.

The overviews of well developed components are as follows:

Microsoft's COM, DCOM and COM+: Microsoft has implemented a COM component to develop the desktop applications [3]. DCOM is being implemented to operate remote applications, and COM+ is a higher version of COM. The limitations of the above components are running under the windows operating system. The awareness of Microsoft system based tools is required to implement the above domain specific components.

Java Beans Components: SUN Microsystems are required the familiarity of enterprise Java beans and Remote method invocation (RMI) to develop the Java Beans component [4]. Components of Java beans are platform independent, which overcome some of the limitations of Microsoft's component. RMI is used to invoke the component of one Java program into another Java program within the network boundary. The limitation of RMI is also that, it runs only for Java based applications.

CORBA of OMG Group: The domain specific limitations of Microsoft's and SUN Microsystems, the OMG has launched common object request broker architecture (OMG/CORBA). The application developer has to use CORBA component model [5], which cross the boundaries of domain specific applications. An ORB provides different services that enable the one component to communicate other component in a transparent manner. The CORBA supports the architecture of various programming language developed by different vendors. For the language and environmental interoperability [6], CORBA provides Interface Definition Language (IDL), which is used to implement the component in any programming language.

The ORB services are used for component communication as shown in figure: 1. The Stubs and Skeletons are generated for each component by using their IDL compiler, for example (C++ to IDL ACE+TAO, omniORB etc, Java to IDL, idlj, python to IDL, omniORBpy, and .NET to IDL, IDLtoCLS etc). Stubs and skeletons file play crucial role for client server communication.

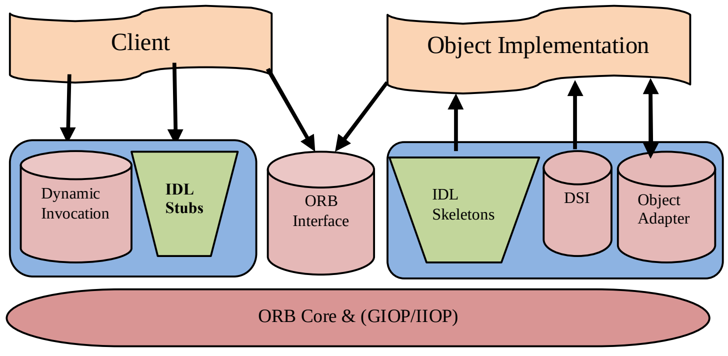


Fig1: ORB Architecture

To communicate with the ORB [7], the application uses a static IDL stub on the client end and static IDL skeleton at server end, which invokes the implementation of an IDL file that contains the interface definition.

2. Overview of IDL Compiler Tools

There are many IDL compiler tools were developed by different vendors and they successfully achieve the adaptive environment for most of the languages [8]. By using CORBA we can achieve various object-oriented languages interoperable in any environment and successfully build the component-based application [9] [3]. IDL compilers that support the CORBA [10] standard such as: **IIOP.NET**, interoperation between .NET, and CORBA or J2EE, Jacob wrote in Java IDL-to-Java Compiler, **R2CORBA,** a CORBA implementation of the Ruby Programming Language, VBOrb**,** CORBA Visual Basic clients and servers, MICO**,** IDL to C++ mapping, ACE ORB (TAO), IDL to C++ mapping, **omniORB**, ORB with C++ and Python bindings, ORBit**,** C and Perl bindings, *idlj - The IDL-to-Java Compiler* etc.
In our example we have used the omniORB IDL to python language mapping and **IIOP.NET** channel for C#.Net mapping, to make interoperable using CORBA middleware. Here we summarize the IDL compiler tools omniORB and IIOP.net channel:

2.1 OmniORB

The OmniORB [11] [12] is an Object Request Broker (ORB) that develop the specification of the Common Object Request Broker Architecture (CORBA). **OmniORB** is a robust high performance CORBA ORB for C++ and Python. It is an open source implementation and freely accessible under the terms of the GNU Lesser General Public License (for the libraries), and GNU General Public License (for the tools). OmniORB has always been designed to be portable. It runs on many versions of UNIX, Windows etc, It is designed to be easy to port to the new environment. The IDL to C++ mapping for all target platforms is similar. The main features of OmniORB are Multithreading and Portability. The major limitations of OmniORB are that it does not have its own interface repository and standard Portable Interceptor API.

2.2 IIOP Channel

The main requirement to communicate with the Common Object Request Broker Architecture (CORBA), a channel [10] for the IIOP protocol was implemented by Dominic Ullmann and Patrik Reali. The IIOP.NET does not provide interoperability with python ORB in different environments. The major importance of the IIOP.NET project has been to maintain the IIOP protocol between Java and .NET. However, IIOP.NET can also support the compatibility with C++ client and server through ACE+TAO.

3. Problem Definition

The main advantage of CORBA technology to achieve interoperability of component objects. We have implemented the two very simple client server model based on CORBA standards. IIOP.NET allows interoperation between .NET, CORBA and other distributed objects. This is done by incorporating CORBA/IIOP support into .NET, influencing the Remoting framework [10]. Since, python IOR does not support by .NET framework due to limitation of standards and lack of development in this area. We have implemented the python server that accepts the request of C#.Net Client. To test the efficiency of interoperable objects based on CORBA an example has implemented with the aim to calculate the multiplication and division operation. C#.Net and python seems important in terms of efficiency of communication between component objects, to test interoperability of these objects in different environments. In our approach, First server has implemented in IIOP C#.NET for multiplication and second server has implemented in Python and integrated mixed C#.NET and Python. The proposed model is demonstrated in figure: 2 as follows:

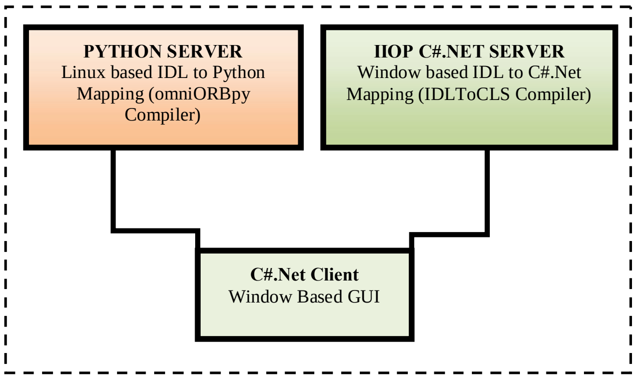

PYTHON SERVER
Linux based IDL to Python Mapping (omniORBpy Compiler)
IIOP C#.NET SERVER
Window based IDL to C#.Net Mapping (IDLToCLS Compiler)
C#.Net Client
Window Based GUI

Fig2: Communication between Python Server, IIOP C#.Net Server and C#.Net Client

In the Client-Server model, python server generates an IOR (Interoperable Object Reference), this IOR is copied into the client implementation file to invoke the server object, if the client and server running on the same machine then it can easily copy from server to client. If the client and server running on different machines, then, we need to remotely copy the IOR from server to client. It is a little difficult for the developer. So, we have implemented an approach that overcomes to the remotely copy the IOR, and extend the interoperability between C#.net and python.

4. Example

In our example, the work carried out on a network of two different machines. IIOP C#.NET Server based on Microsoft's windows 7 Operating System, and **IIOP.NET-1.9.3**, IDLToCLS compiler, IDL to C#.NET mapping. C#.NET server computes the result of the multiplication operation by using input parameters. Python Server based on Ubuntu10.10 and **omniORBpy-2.7** IDL compiler, IDL to Python mapping. The python server computes the result of a division operation by using the same parameters. GUI based client has implemented in C#.Net, which passes the input parameters to the distributed objects and receives the result of multiplication and division by using the same input parameters The process diagram of the communication is shown in fig: 3

Procedure:

1. Launch the C#.Net Server and Python Server on Different Machine by using different port no.
2. Launch the client application that receives the input parameter and choice for multiplication and division operation
3. Client side we have two choices to choose the operation. Choose one for multiplication and another for division.
4. C#.Net client simply sends the input parameters to the C#.Net Server and python server.
5. These input parameters are passed into a python object that computes the result.
6. The Result is sent back to the client.

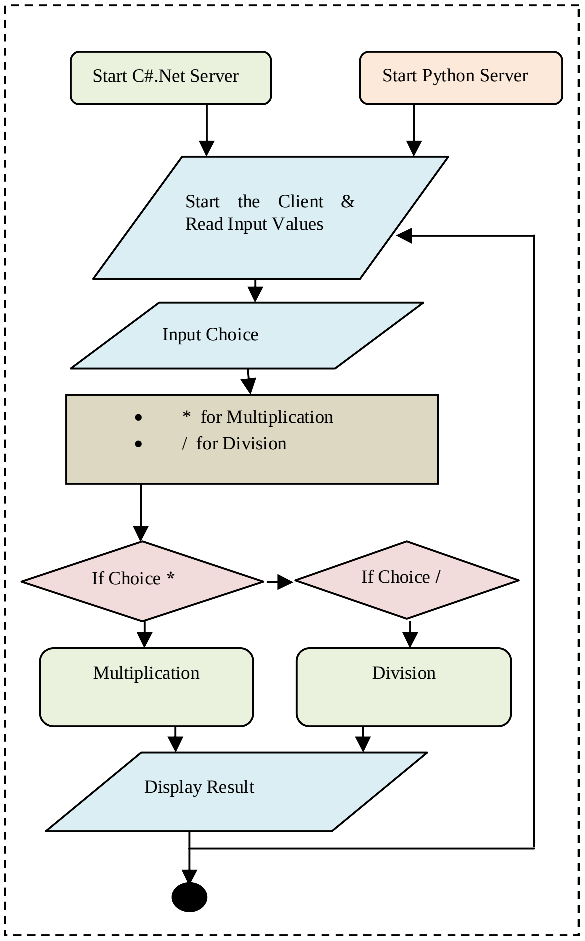


Fig: 3 Client- Server Communication Process

IDL Interface: straightforwardness of the IDL file is the proposed action [13][14]; this makes possible testing and directs to the transparency code. The IDL file content is very simple as shown in fig: 4. The IDL contains the definition of multiplication and division and an interface containing the method to calculate the operation. After initialization the C#.net server, python server and C#.net client, the client receives two input parameters, to compute the result of multiplication and division operation.

```
//Division.idl
interface Division
{
double div(in double a, in double b);
};
```

```
//Multiplication.idl
using System;
using System.Runtime.Remoting;

namespace CalciClient
{
    public interface Multiply

    {
        double mul(double a, double b);
    }
}
```

Fig: 4 The IDL Interface for C#.Net and Python

The distributed object recognizes the assignment of finding an implementation repository for the input parameters and forwards the calculated result to the client. Python server code as shown in figure: 5, receives the input parameter and establish the connection between C#.net client and python server. This input parameter is passed to the python object. These input parameters are used by a python object to evaluate the result of division operation and the result is given to the C#.net client.

5. Results

There are two cases, in which we have evaluated the results:

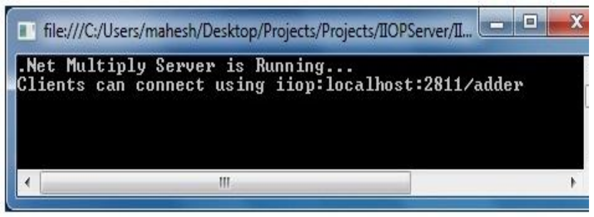
***Case1:* Communication between C#.Net server and C#.net client:**

Initially we start running the C#.net server and a Python server on a different machine by using different port no. as shown in figure: 6 and 8, and then we launch the C#.net client application. In this case, client-server communication using CORBA services is excellent due to the same environment of client and server.

```
#!/usr/bin/env python
import sys
import socket
from omniORB import CORBA, PortableServer

# Import the stubs and skeletons for the Example module

import _GlobalIDL, _GlobalIDL__POA

class division (_GlobalIDL__POA.Division):
   def div(self, a,b):
      return a/b

 # Initialise the ORB
orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
 # Find the root POA
poa = orb.resolve_initial_references("RootPOA")
 # Create an instance of Div
ei = division()
 # Create an object reference, and implicitly activate the object
eo = ei._this()

# calling python object using input parameter form C#.Net Client

x=ei.div(data[0],data[1])
print orb.object_to_string(eo)
conn.sendall(str(x))
print "Result of Division send to the client......"

poaManager = poa._get_the_POAManager()
poaManager.activate()
orb.run()
```

Fig: 5 Python Server code

The multiplication result, which is computed by C #.net server by using the input parameter, is shown in figure: 7


file:///C:/Users/mahesh/Desktop/Projects/Projects/IIOPServer/II...

```
.Net Multiply Server is Running...
Clients can connect using iiop:localhost:2811/adder
```

Fig: 6 Running process of C#.Net Server on port no.2811

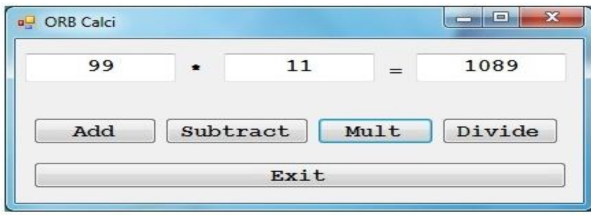
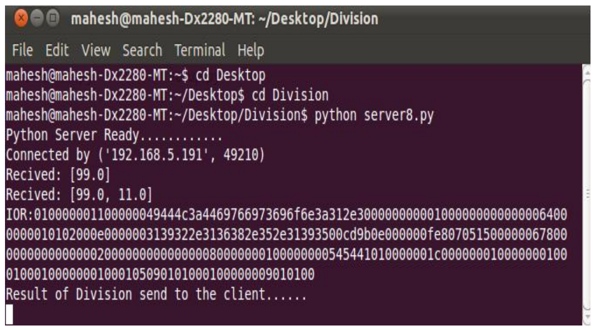


Fig: 7 Running process of C#.Net Client on port no.2811

***Case2:* Communication between Python server and C#.net client:**
In case1, client server communication is very strong using the CORBA standard and services. But, in case2, shown in figure: 8, for client server communication in different environment and different language (e.g. .Net framework and python), python ORB & IOR does not support directly. As a result, we ultimately employ the python servant object all the way through communication. In this case, the copy of IOR is not requisite to the client for servant object invocation. As an alternative, we use the python object services on the server side.


mahesh@mahesh-Dx2280-MT: ~/Desktop/Division
File Edit View Search Terminal Help

```
mahesh@mahesh-Dx2280-MT:~$ cd Desktop
mahesh@mahesh-Dx2280-MT:~/Desktop$ cd Division
mahesh@mahesh-Dx2280-MT:~/Desktop/Division$ python server8.py
Python Server Ready............
Connected by ('192.168.5.191', 49210)
Recived: [99.0]
Recived: [99.0, 11.0]
IOR:010000001100000049444c3a4469766973696f6e3a312e30000000000100000000000000006400
000001010200 0e0000003139322e3136382e352e31393500cd9b0e000000fe807051500000067800
000000000000200000000000000080000000100000000545441010000001c00000001000000100
01000100000001000105090101000100000000 9010100
Result of Division send to the client......
```

Fig:8 Running process of Python Server

The result of a division operation, which is computed by the Python server by using the same input parameter and CORBA, is shown in figure: 9

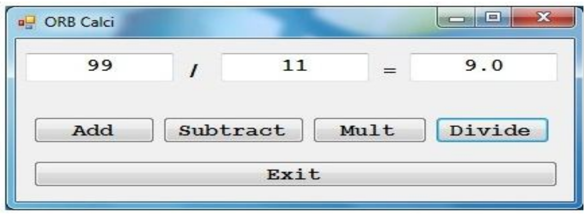


Fig: 9 Running process of C#.Net Client on port no.2809

6. Conclusion

In this paper we are presented opportunities of usage CORBA middleware standard and services for the component object for different programming languages .Net framework and python, with challenging importance on cross communication implementation. We have tested that how a C#. Net client written in C#, and running on Windows, can communicate directly with a .NET server, and python server by using the IIOP protocol and omniORBpy. Additionally, we have also seen how to write C#.Net client, using IIOP.NET, which can communicate with python server, running on Linux.

CORBA is a benchmark which supports the architecture of various programming languages, which makes it very reasonable means to implement the component based application. But, sometimes it may be challenging, As CORBA is used with various external tools such as ACE+TAO, omniORB, idlj, IIOP.net, omniORBpy etc.

Some component technology already exists, in which development of the application is not complicated because of their domain specific nature, but when we cross the domain, then there is need to such kind of standard and technology in which we can develop component based application. After the execution of implementation, it can be concluded that server and client in python and .NET framework is the most effective way for component object communication.

References

[1] F Bronsard, D Bryan, W Kozaczynski; Toward software plug-and-play SSR'97 Proceedings of the 1997 symposium on Software reusability Pages 19 – 29, ACM New York, NY, USA ©1997.

[2] Hall, L.; Hung, C.; Hwang, C.; Oyake, A.; Yin, J.; , "COTS-based OO-component approach for software interoperability and reuse (software systems engineering methodology)," Aerospace Conference, 2001, IEEE Proceedings. , Vol. 6, no., pp. 2871-2878 Vol. 6, 2001.

[3] Onderka Z., Cichy M.; The Comparison of the Communication Eciency for the CORBA and DCOM Standards in the Client Server Systems, Computer Networks, 2011. Will be published in Studia Informatica.

[4] Deitel & Deitel, 2001, Java How to write a program. USA, Prentice Hall.

[5] Z Onderka, The efficiency analysis of the object oriented realization of the client server systems based on the CORBA standard publication published online January 23, 2012. DOI 10.4467/20838476SI.11.010.0296.

[6] Hill, J.H.; "Towards Heterogeneous Composition of Distributed Real-Time and Embedded (DRE) Systems Using the CORBA Component Model," Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on , vol., no., pp. 73-80, Aug. 30 2011-Sept. 2 2011.

[7] A Yahiaoui, J Hensen, L Soethout; Developing CORBA-Based Distributed control and building performance environments by run-time coupling , International Conference on Computing in Civil and Building Engineering , ICCCBE , 10 , 2004.06.02-04 , Weimar.

[8] Object Management Group; Object Management Architecture Guide, OMG Document Number 92.11.1, Revision 2.0, 1992.

[9] Object Management Group; The Common Object Request Broker: Architecture and Speciation, OMG Document, Version 2.0., 1995.

[10] IIOP. NET-Documentation URL at http://iiop-net.sourceforge.net/documentation.html

[11] The omniORB version 4.1, User's Guide Duncan Grisby, Apasphere Ltd., Sai-Lai Lo, David Riddoch, AT&T Laboratories Cambridge, July 2009.

[12] Object Management Group (OMG), Object management architecture guide: revision 2.0

[13] Corba 3 fundamentals and programming, John Wiley & Sons, 2000 - Computers.

[14] M. K. Pawar, Dr. Ravindra Patel, Dr. N. S. Chaudhari; "Way to Component-based Vending Machine," CIIT International Journal of software engineering, 2012 , Vol.4, no.10. , pp. 447-451, Nov. 2012.

M. K. Pawar, Assistant Professor, Department of Information Technology at Rajiv Gandhi Proudyogiki Vishwavidyalaya (State Technological University of Madhya Pradesh), Bhopal, India. He has M. Tech. Degree in Information Technology. He posses more than 12 years of experience in the industry as well as teaching of graduate and postgraduate classes. He has published 02 papers in international journals and conference proceedings. He is a member IEEE.

Dr. Ravindra Patel, Associate Professor and Head, Department of Computer Applications at Rajiv Gandhi Proudyogiki Vishwavidyalaya (State Technological University of Madhya Pradesh), Bhopal, India. He has been awarded Ph.D. degree in Computer Science. He posses more than 12 years of experience in teaching postgraduate classes. He has published more than 15 papers in international journals and conference proceedings. He is a member of the International Association of Computer Science and Information Technology (IACSIT) & IEEE.

Dr. Narendra S. Chaudhari Professor, Department of Computer Science, Indian Institute of Technology (IIT) Indore, MP and Member - Advisory Board, ITM University, Gwalior (M.P.). He has been referee and reviewer for a number of premier conferences and journals including IEEE Transactions, Neurocomputing, etc. Dr. Chaudhari is Fellow of the Institution of Engineers, India (IE- India), as well as Fellow of the Institution of Electronics and Telecommunication Engineers (IETE) (India), senior member of Computer Society of India, Senior Member of IEEE, USA, Member of Indian Mathematical Society (IMS), Member of Cryptology Research Society of India (CRSI), and many other professional societies.