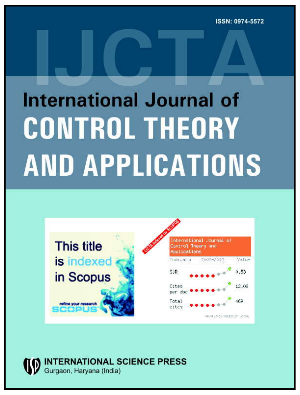


International Journal of Control Theory and Applications

ISSN : 0974-5572

© International Science Press

Volume 10 • Number 19 • 2017

A Framework for Heterogeneous Environment to Build an Application using CORBA

Mahesh Pawar[1], Anjana Pandey[1], Sachin Goyal[1] and Ratish Agrawal[1]

[1] Department of Information Technology, University institute of Technology, State Technological University (RGPV), Bhopal, Madhya Pradesh, India, Emails: mkpawar24@gmail.com, anjanapandey@rgtu.net, sachingoyal@rgtu.net, ratish@rgtu.net

Abstract: The main intention of object oriented development is to facilitate the independent flexible composition of software components for software reuse artifacts. The major component based software technologies such as CORBA, EJB, DotNet have shifted the attention from conventional programming to component based programming approach that enables rapid development of applications. However, achieving component interoperability in a fully automatic manner is still an intimidating job. In this paper, a robust framework has been proposed that handles compatibility issues between incompatible software components and let them communicate to each other. In this work several tools and techniques have been developed to make a component interoperable in a practical sense. This work extends the interoperability between various technologies with reasonably complex software tools. This framework generates a generalized code that enables component interoperability through an IDP (Interface, Data type and Protocol) tool. This work contributes to common understanding of IDL (Interface Definition Language), IDL compiler and modeling of components to build an interoperable application. The framework also possesses the capability to act as an intermediate interface for clients that are incompatible with server to enable communication between them. To test the framework, a case study of vending machine application has been developed whose different components were implemented using different languages and platforms i.e. Python, C++, Java and C#.Net for creating a mixed environment with CORBA as middleware. we have used omniORB for C++ to IDL mapping, omniORBpy for Python to IDL mapping, idlj for JAVA to IDL mapping and IIOP.NETIDLtoCLS for C#.NET to IDL mapping.

Keywords: Component Interoperability, mixing of Component objects, component cross communication, Components Heterogeneity

1. INTRODUCTION

In traditional software engineering, software development process comprises a sequence of activities such as analysis, design, coding, testing and integration. The major role of software engineering is to come up with a standard set of tasks, standard techniques, and standard tools that have a considerable effect on the primary dimensions that all engineering disciplines address: cost, quality and time to market.

About 70 to 80%of software engineering deals with changing the existing software. It is still in its relative infancy as compared to more traditional engineering disciplines like Mechanical engineering, Civil engineering and so on. Traditional software engineering is facing various challenges raised by software industry and end user, which includes rapid increase in size and complexity of software etc. There is a rapidly increasing demand for development of component based technology in software industries [1].

It is because industries are moving from traditional domain specific programming approach towards the software systems that are built up from platform and language independent components. There are many other engineering disciplines where the components are well developed and acceptable according to the system because of standards and rules are predefined.

Similarly, component based software engineering (CBSE) offers an opportunity to programmers through which, they can employ the existing plug incompatible software components rather than implementing everything from scratch. CBSE also offers significant benefits to component based systems such as enforcing management of dependencies, making system flexible, lowering down maintenance cost and strengthening the software. The proposed model is based on CORBA middleware technology.

The main reasons that CORBA offers several benefits [2]:

- CORBA middleware has very powerful features, which support various programming languages and operating systems.
- CORBA is an open standard technology in which developer can select an implementation from various CORBA vendors.
- CORBA standards support mapping of a variety of programming languages, such as C, C++, LISP, Python, JAVA, COBOL, Ada, PL/I etc.
- CORBA makes it feasible to develop servers that handle unlimited number of objects.

2. RELATED WORK

Interoperability can be well-defined as the ability of two or more entities to communicate and cooperate with each other regardless of variation in implementation language and execution environment.

Traditionally there are two levels of interoperability: *The Signature Level:* specifies the names and signatures of operations and *The Semantic Level:* specifies the significance of operations. The syntactic interoperability is now well defined and at this stage, various architects and middleware vendors are trying to establish interoperation standards. An architecture description languages (ADLs) are used to express and manipulate the structure of a component based system [3] [4] [5]. It also establishes the relationship and interaction of components. Most ADLs also represent a formal basis for interoperation of components. The CAPE-OPEN (CO) is a major technology for enablement of interoperability. The CO currently does not include standard warning concept. The Model based computing contains a similar high level description of a system [6] but this explanation is more commonly leveraged to generate a subsequent application.

Another view is computing model often depends upon the structure within components. Model based computing is commonly used in embedded system programming. Rastofer proposed a meta-model, which is suitable for both component instances and component model [7]. Re-modeling of an abstract component model is simple, effective and popular. This approach focuses on modeling the component model structure but does not consider the behavior. A formal model that defines behavior and in depth information of Enterprise JavaBeans (EJB) component model was presented in by Sousa et al.[8]. Authors performed EJB formal recognition of the structure and established many inconsistencies.

Ptolemy's component based system is also intended for embedded systems [9]. It permits use of several models of computation that provides a wide array of possibilities to express a problem domain. Ptolemy's computational models fundamentally differ from each other and include dataflow, time triggered, synchronous, isolated events and process networks etc.

They aim to combine various component models that can rely on different models of computation. It is a very low level tool and we are more interested in alliance of component models. The originators of ASSIST [10] presented a parallel and distributed high performance computation framework that sets up a binding between ASSIST programs and the CORBA component framework [11]. The bridge between the two different environments is generated by means of a conventional compiler. ASSIST program considers a component as independent executable method within a CORBA framework. The CORBA components can invoke this stand-alone method to interface with parallel ASSIST program. The interoperability of ASSIST and CORBA is an illustration of a hand written bridge scenario for which, they try to provide abstractions that facilitate easier conception. ASSIST allows fair interoperability levels with respect to the CORBA framework.

3. COMPONENT INTEROPERABILITY FRAMEWORK

The proposed interoperability framework binds various parameters with IDL generated code that combines two or more components. With a binding standard language and the IDP (Interface, data type and protocol) mapping, this work focuses to evaluate various aspects. Regularity of all aspects of the software component lies within the framework.

By exploiting this regularity, we enable the rapid development and evaluation of generalized code. Once the component model differences are understood and expressed in a standard format, they can easily be coupled with other component models. This framework is found effective in creating code that communicates between two different sets of interfaces and data types.

Our approach introduces a binding standard language based upon the meta-modeling techniques that adds semantic information to mapping and code generation process. This framework can be considered significant because it addresses several issues regarding the data type, interfaces, and protocol differences between component models. The main contribution of this framework is mapping between IDL generated code and IDP, which is augmented with data models.

The purpose of IDP mapping is to create ontology between component objects. The data models, assisted by IDP mapping, create a definition of data types and their handling for individual component model. The output of this meta-model, binding language as input to IDL tool and the generated code is mapped with IDP to produce the generalized version of Interface and a data type.

This framework mainly deals with the creation of a static, direct and generalized code between two or more existing component objects.

This section describes our approach to component interoperability, and demonstrates a framework that makes the interoperability practically possible. This work begins the discussion with IDL tool, which is augmented with meta-modelling, standard language binding and data model. The semantic tool (IDP) that we have designed for achieving the component interoperability. The component of framework is shown in Figure 1.

An IDL is a specification language used to is a specification language used to interpret software components' interfaces.

It describes an interface in a language-independent manner, which enables us to access those objects across the network. But it should be noted that IDL does not deal with details about how to traverse through the network.

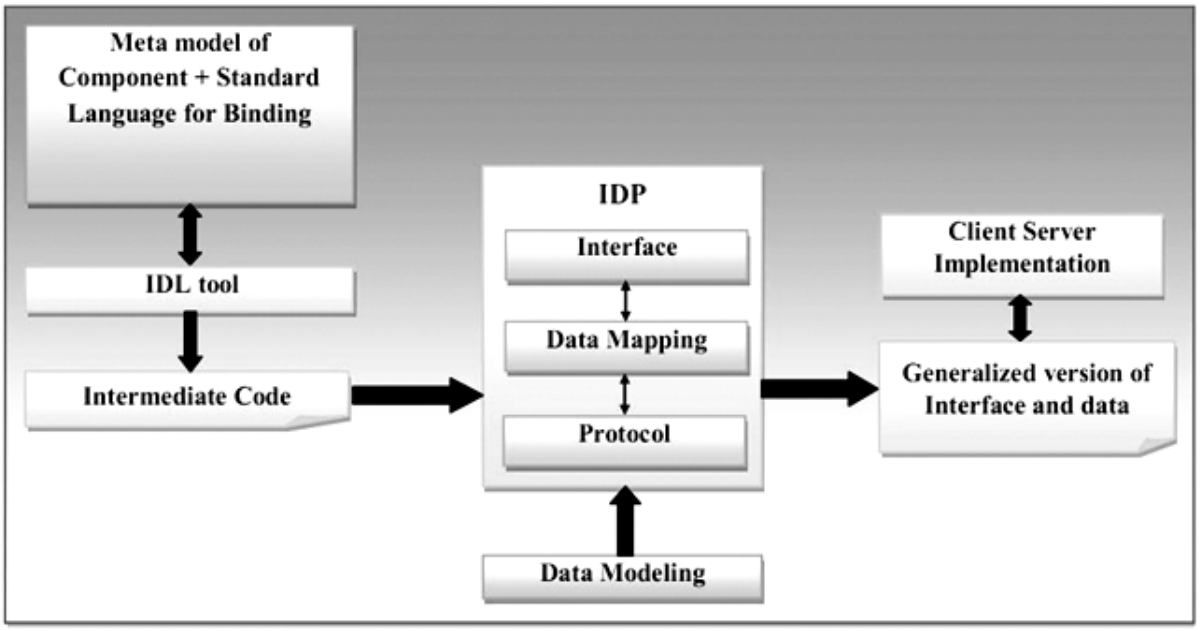

Meta model of Component + Standard Language for Binding
IDL tool
Intermediate Code
IDP
Interface
Data Mapping
Protocol
Data Modeling
Client Server Implementation
Generalized version of Interface and data

Figure 1: Framework for Interoperable version of Interface and data

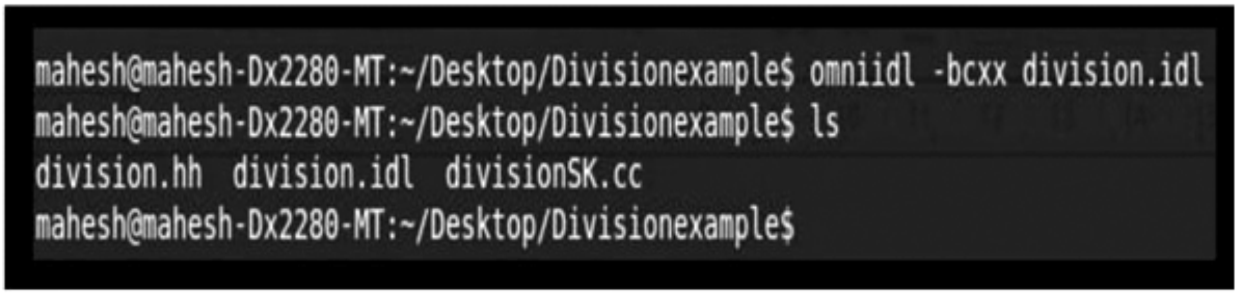
An IDL is compiled using IDL compiler (an IDL tool), which converts the definition of an input interface to a high level language. Another compiler integrates the component's source code with the converted interface source code andproduces the distributed component object.

This process is explained with the help of a simple example below:

//Interface Definition Language for Division interface (Division.idl)
interface Division
{
double div(in double a, in double b); };

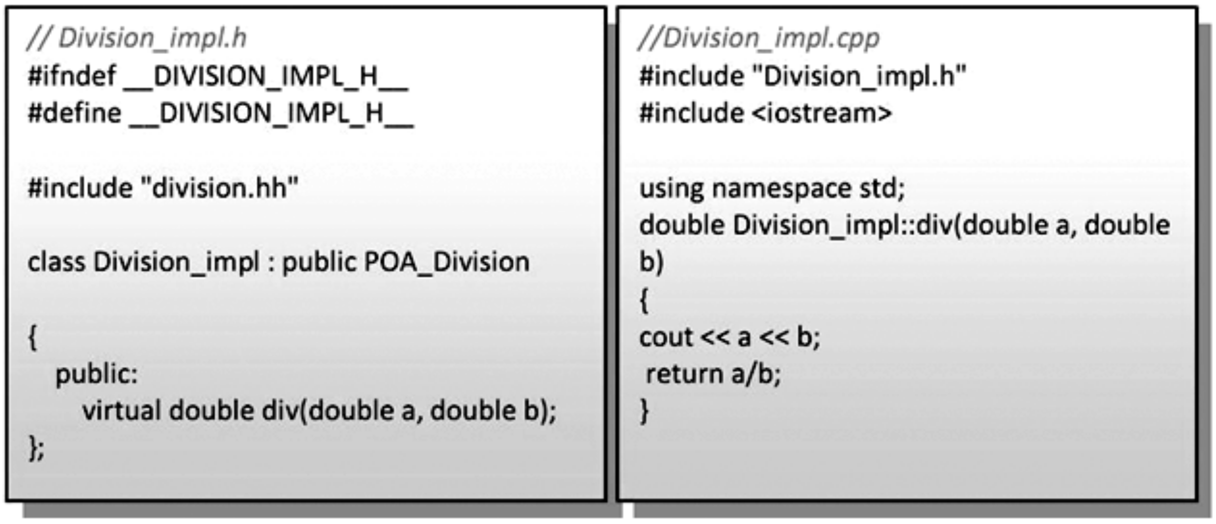
The above IDL contains the definition of division operation and an interface containing the method to compute this operation. This IDL definition is compiled with the help of a language specific IDL compiler.

To illustrate the compilation process of this IDL definition, an IDL to C++ mapping compiler (omniORB) is used. After compilation the division.idl (as shown in Figure: 2), it generates the division.hh and divisionSK.cc files in C++ language, which act as stub and skeleton respectively. To further process these IDL generated files for performing division operation, it requires implementing the following files (as illustrate in Figure..3) in C++.



```
mahesh@mahesh-Dx2280-MT:~/Desktop/Divisionexample$ omniidl -bcxx division.idl
mahesh@mahesh-Dx2280-MT:~/Desktop/Divisionexample$ ls
division.hh  division.idl  divisionSK.cc
mahesh@mahesh-Dx2280-MT:~/Desktop/Divisionexample$
```

Figure: 2 Division IDL processing using OmniORB



```
// Division_impl.h
#ifndef __DIVISION_IMPL_H__
#define __DIVISION_IMPL_H__

#include "division.hh"

class Division_impl : public POA_Division

{
   public:
      virtual double div(double a, double b);
};
```

```
//Division_impl.cpp
#include "Division_impl.h"
#include <iostream>

using namespace std;
double Division_impl::div(double a, double
b)
{
cout << a << b;
 return a/b;
}
```

Figure 3: Implementation of division operation in C++

In addition, to write the client and server programs to invoke the division operation. These files (i.e. client.cpp and server.cpp) are required to implement the client and server.

4. A CASE STUDY OF COMPONENT BASED VENDING MACHINE

This section applies the component interoperability framework to build an application model of well-known vending machine [12]. It is generally used to dispense various items such as Cold-drinks, coffee, Ice-creams etc. to buyers.

In this application model (as shown in Figure: 4), various components are implemented in C++, JAVA, PYTHON and C#.Net [13]. The vending machine accepts coins as user inputs in any succession and dispenses the products. If input amount is more than the cost of item, which user is currently buying then machine returns back the change.

It also supports an abort feature, which means a user can abort the transaction at any time and get the inserted money back without buying any item. Our vending machine consists of following major components, which have been implemented using different programming languages:

- ***Select Item:*** It is used to select the item(s) from item list. The items and their costs are verified by this C++ Component.
- ***CoinCheker:*** It checks the validity of inserted coin. This Python Component displays the values of coin inserted by the client for buying the item(s).
- ***ComputeChange:*** It compares the item cost with value of coin(s) inserted by the user, and returns the value of remaining amount. This task is performed by Java Component.
- ***Dispenser:*** It returns the item and change (if any). This component is implemented in C#.Net.

5. RESULTS

There are four different scenarios in which we have evaluated the result of different components they performed the task to dispense the item form vending machine as required by the user.

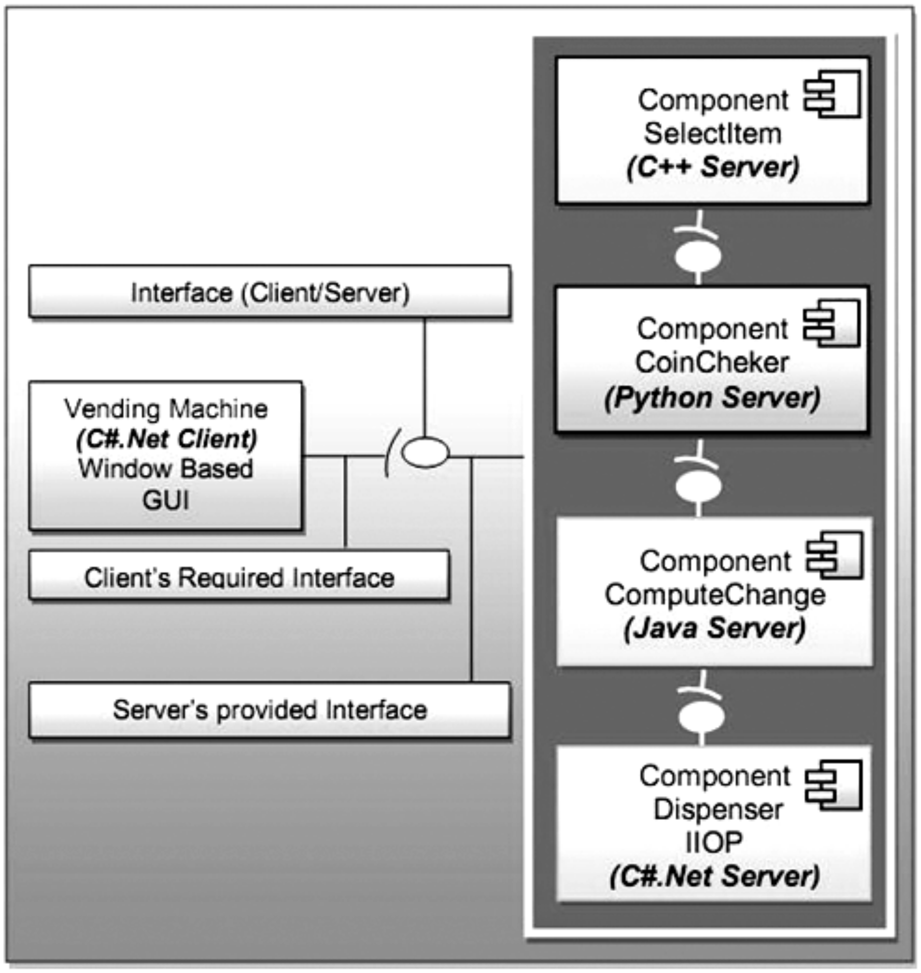


Figure 4: Vending Machine client-server communications in heterogeneous environment

Scenario1: *omniORB C++ SelectItem Component and C#.NET client*

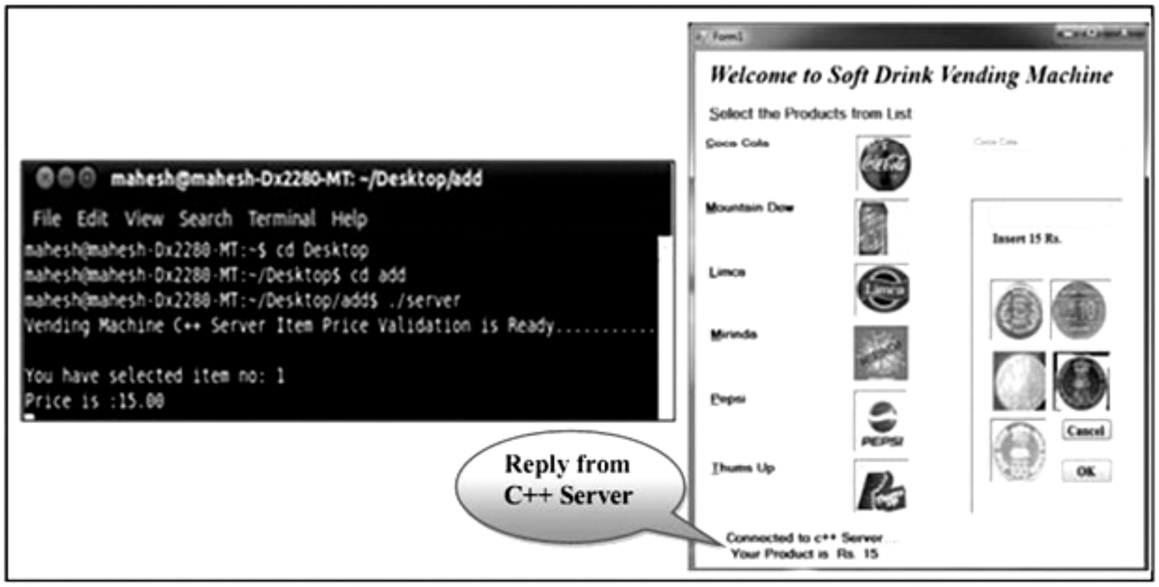
In this omniORB C++ server (as shown in Figure:5) and C#.NET client communication, he/she selects the soft drink item from the list of products, which is displayed in the client application. The selected item will go to the C++ server, where it verifies the selected item and send their price to the client. In our example user select the product coca cola from the client application, it will reflect to the C++ server. The server will display the user choice and the price of the item (Rs.15) will send to the client.

Scenario2: *omniORBpy Python CoinCheker Component and C#.NET client*

In this scenario, he/she insert the Coin of Rs.20, which is validated by the python server (as shown in Figure: 6). Python server will send the valid value of the coin to the C#.NET client. Here for coin checker purpose we simply display the message for the validity of the coin.

Scenario3: *idlj Java ComputeChange Component and C#.NET client*

In the third scenario, user has inserted the Coin of Rs.20, and price of product is Rs.15, now the Java server plays the role of comparison (as shown in Figure 7) between product price and user inserted money. Java server will get these values and perform the ComputeChange operation and return the value of Rs.5.

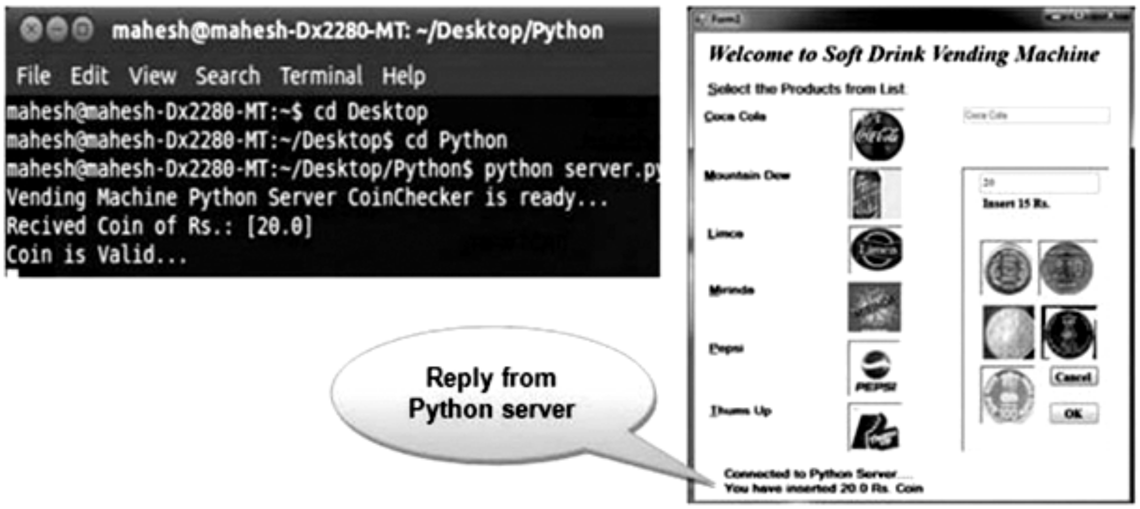


```
mahesh@mahesh-Dx2280-MT: ~/Desktop/add
File  Edit  View  Search  Terminal  Help
mahesh@mahesh-Dx2280-MT:~$ cd Desktop
mahesh@mahesh-Dx2280-MT:~/Desktop$ cd add
mahesh@mahesh-Dx2280-MT:~/Desktop/add$ ./server
Vending Machine C++ Server Item Price Validation is Ready..........

You have selected item no: 1
Price is :15.00
```

Connected to c++ Server...
Your Product is Rs. 15

Figure 5: Snapshots of C++ server component (left) and C#.Net client for SelectItem component (right)



```
mahesh@mahesh-Dx2280-MT: ~/Desktop/Python
File  Edit  View  Search  Terminal  Help
mahesh@mahesh-Dx2280-MT:~$ cd Desktop
mahesh@mahesh-Dx2280-MT:~/Desktop$ cd Python
mahesh@mahesh-Dx2280-MT:~/Desktop/Python$ python server.py
Vending Machine Python Server CoinChecker is ready...
Recived Coin of Rs.: [20.0]
Coin is Valid...
```

Connected to Python Server...
You have inserted 20.0 Rs. Coin

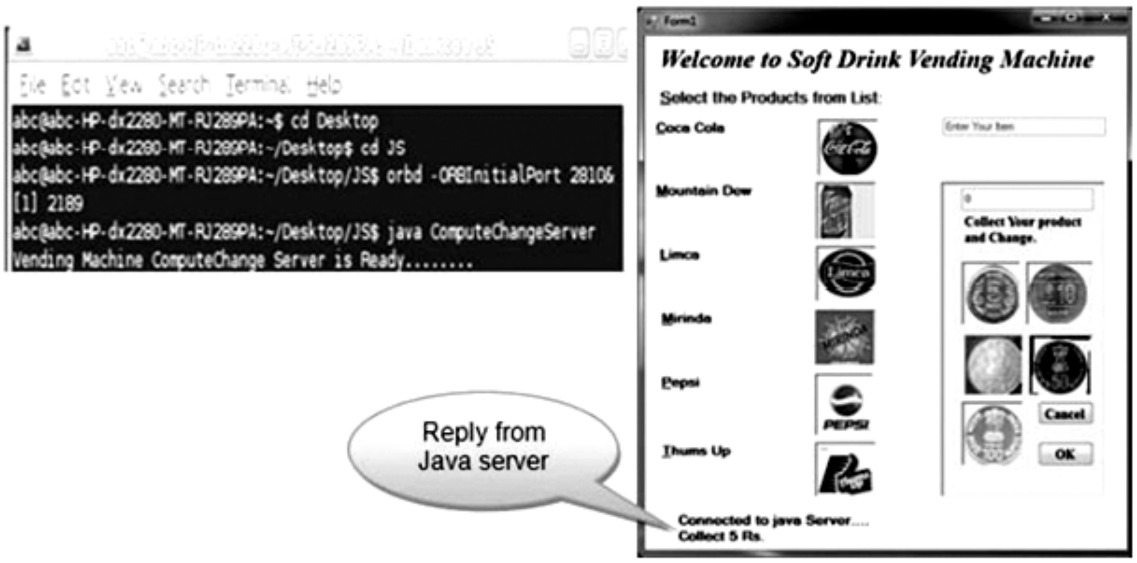
Figure 6: Python Server (left) and C#.Net client (right) for CoinChecker Component

Scenario4: *IIOP C#.NET Dispenser Component and C#.NET client*

In this scenario, the communication between client and server is robust due to same environment of client and server. The dispenser component will dispense the item and return change (if any).

Dispenser component also perform the task of difference money (for example if user insert the coin of Rs. 10 and the product price is Rs. 15 then Dispenser server display the message to insert Rs.5 more (as shown in Figure: 8) to purchase the item.) Here we perform all the validations of interface level and application level so that the overall performance of the application is in a robust way.

All the component of vending machine tested well and every component performs their task in an appropriate way.

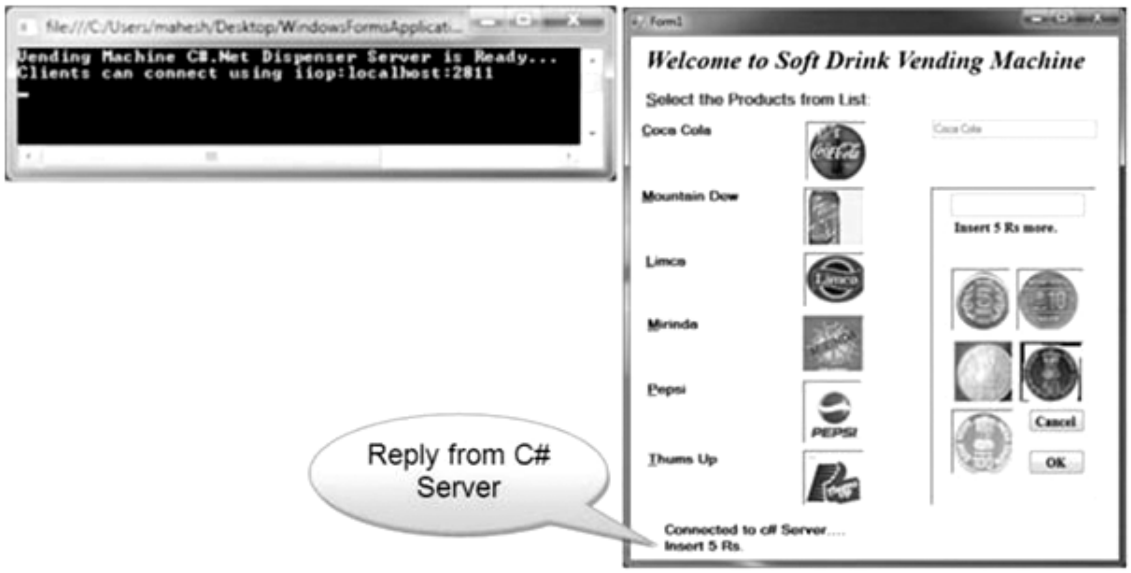

File Edit View Search Terminal Help
Form1
Welcome to Soft Drink Vending Machine
Select the Products from List:
Coca Cola
Mountain Dew
Limca
Mirinda
Pepsi
Thums Up
Enter Your Item
0
Cancel
OK
Reply from Java server

```
abc@abc-HP-dx2280-MT-RJ289PA:~$ cd Desktop
abc@abc-HP-dx2280-MT-RJ289PA:~/Desktop$ cd JS
abc@abc-HP-dx2280-MT-RJ289PA:~/Desktop/JS$ orbd -ORBInitialPort 28106
[1] 2189
abc@abc-HP-dx2280-MT-RJ289PA:~/Desktop/JS$ java ComputeChangeServer
Vending Machine ComputeChange Server is Ready........
```

Collect Your product and Change.

Connected to java Server....
Collect 5 Rs.

Figure 7: Java server (left) and C#.Net client (right) for ComputeChange component


file:///C:/Users/mahesh/Desktop/WindowsFormsApplicati...
Form1
Welcome to Soft Drink Vending Machine
Select the Products from List:
Coca Cola
Mountain Dew
Limca
Mirinda
Pepsi
Thums Up
Coca Cola
Cancel
OK
Reply from C# Server

```
Vending Machine C#.Net Dispenser Server is Ready...
Clients can connect using iiop:localhost:2811
_
```

Insert 5 Rs more.

Connected to c# Server....
Insert 5 Rs.

Figure 8: C#.Net Server (left) and C#.Net client (right) for Dispenser component

6. CONCLUSION AND FUTURE WORK

The proposed framework has opportunities of using CORBA middle-ware standard and its services for integrating the component objects developed in different programming languages. The works have been tested, in which a C#.Net client written in C#, running on Microsoft Windows can communicate directly with a C++ server by using the IIOP protocol. Additionally, It is also explained how to write the C#.Net client using IIOP.NET, which can communicate with C++ server, running on Linux. The benefits of channel IIOP.NET should be recognized by C#.Net programmers, who still interoperate with OmniORB code. CORBA is a benchmark that supports the architecture of various programming languages, thus making it very reasonable means to develop component based applications. The major limitations to implement

CORBA based component applications include the need to learn variety of programming languages, IDL language and data types for mapping.

Under this work, a framework is designed, which is based on standard library files for data types, interfacing, syntax and semantics to build component based applications. One aspect of the future work is that CORBA services are used with various external tools such as IDL compilers of various vendors. The IDL compilers pose challenges for configuring, versioning problem and interoperability with standard language compiler.

Second aspect of this component Interoperability framework is that the IDL compiler does not generate all syntactical constructs, which are described in the IDL specification. The third aspect is to use the component interoperability framework in a variety of programming languages and operating systems to build applications. This will prove its applicability and any mismatch of the framework.

REFERENCES

[1] M. K. Pawar, R. Patel, and N. S. Chaudhari, "Survey of Integrating Testing for Component-based System," International Journal of Computer Applications, vol. 57, pp. 21–25, November 2012. Published by Foundation of Computer Science, New York, USA.

[2] CORBA explained Simply at http://www.ciaranmchale.com/corba-explained-simply.

[3] R. Allen and D. Garlan, "Formalizing architectural connection," in Proceedings of the 16th international conference on Software engineering, ICSE '94, (Los Alamitos, CA, USA), pp. 71–80, IEEE Computer Society Press, 1994.

[4] E.M. Dashofy, A. van der Hoek, and R. N. Taylor, "An infrastructure for the rapid development of xml-based architecture description languages," in Proceedings of the 24th International Conference on Software Engineering, ICSE '02, (New York, NY, USA), pp. 266–276, ACM, 2002.

[5] D. C. Luckham, "Rapide: A language and toolset for simulation of distributed systems by partial orderings of events," in Princeton University, 1996.

[6] M. Fromherz and V. Saraswat, "Model-based computing: Using concurrent constraint programming for modeling and model compilation," in Principles and Practice of Constraint Programming CP '95 (U. Montanari and F. Rossi, eds.), vol. 976 of Lecture Notes in Computer Science, pp. 629–635, Springer Berlin Heidelberg, 1995.

[7] U. Rastofer, "Modelling with components - towards a unified component meta-model," in Proceedings of the Model-based Software Reuse Workshop, Malaga, Spain: Springer-Verlag, June 2002.

[8] J. a. P. Sousa and D. Garlan, "Formal modeling of the enterprise javabeanstm component integration framework," in Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume II, FM'99, (London, UK, UK), pp. 1281–1300, Springer-Verlag, 1999.

[9] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Readings in hardware/software co-design," ch. Ptolemy: a framework for simulating and prototyping heterogeneous systems, pp. 527–543, Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[10] M. Vanneschi, "The programming model of assist, an environment for parallel and distributed portable applications," Parallel Comput., vol. 28, pp. 1709–1732, Dec. 2002.

[11] S. Magini, P. Pesciullesi, and C. Zoccolo, "Parallel software interoperability by means of corba in the assist programming environment," in Euro-Par 2004 Parallel Processing (M. Danelutto, M. Vanneschi, and D. Laforenza, eds.), vol. 3149 of Lecture Notes in Computer Science, pp. 679–688, Springer Berlin Heidelberg, 2004.

[12] M. K. Pawar, R. Patel, and N. S. Chaudhari, "Way to Component based Vending Machine," CiiT, International Journals of Software engineering, vol. 4, no. 10, pp. 447–451, 2012.

[13] M. K. Pawar, R. Patel, and N. S. Chaudhari, "Interoperability between .Net framework and Python in Component Way," IJCSI, International Journals of Computer Science issues, vol. 10, no. 03, pp. 165–170, 2013.