SCHEDAE INFORMATICAE

VOLUME 20 (2011) DOI 10.4467/20838476SI.11.010.0296

The Efficiency Analysis of the Object Oriented Realization of the Client-Server Systems Based on the CORBA Standard[1]

Zdzisław Onderka
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: *onderka@ii.uj.edu.pl*

Abstract. The aim of this work was to analyze the cooperation efficiency of the distributed objects based on the CORBA standard. The obtained results show the possibilities of application to the object-relational databases or object oriented computation of data received via the network from the object managed data base. The distributed objects for the client-server model were implemented in Java and C++ languages. All possible configurations of the implementation for the client and the server were analyzed. Best results were received for the client and the server implementation in C++ language. The worst results were received for the client and the server implementation in Java.

Keywords: distributed objects, CORBA, object-relational database, client-server.

1. Distributed objects

Because of the growing importance of the object technology, and global access to computer networks, more and more frequently computations are realized in the form of distributed objects which communicate with one another in the network. It could be any client-server application, for example the client application which performs the computation of data received from the management server of the object-oriented database [1, 2, 5].

[1]Topic realized within statutory work: 11.11.140.561.

Other applications of the distributed objects are the distributed computations in the heterogeneous local (mainly) networks [13]. In this case, it could be a master-slave application or an application consisting of a collection of equivalent processes. In practice frequently applied models are as follows: one server and several clients or one client and several servers. In each case it is possible to use different programming languages to implement distributed objects.

Appropriate standards such as the distributed object and heterogeneous systems were defined by the OMG consortium (*Object Management Group*). The main known advantages of the development of a homogeneous architecture based on the object technology for the integration of distributed applications are: the possibility of reusing software components and data processing, portability, the possibility of using the abstract data types, modularization and faciliting interaction.

OMG defined the OMA architecture (*Object Management Architecture*) [6] with its most important component which is CORBA (Common Object Request Broker) [8] based on the object exchange mechanism defined in the middleware software ORB (*Object Request Broker*) [7]. It can be seen as an intermediary interface between hardware and software components for different manufacturers. Another important result from the work of the consortium OMG, was to define the standard of communication between the ORB-s that is IIOP protocol (Inter ORB Protocol), based on TCP/IP and applied in the CORBA standard since version 2.0.

One of the advantages of CORBA applications to achieve interoperability of distributed objects, among others, is the possibility of implementing objects in different object-oriented programming languages, among which Java and C++ are the most popular [3, 4]. Of course, the software engineers may apply other programming languages or communication standards (regardless of the level of abstraction) but most of them are dedicated for specific programming languages or system environments such as DCOM [10], .NET Remoting, WCF (*Windows Communication Foundation*) [11, 12], EJB, RMI etc. The text-based protocols were not taken into account due to the need of high communication efficiency for high performance computing.

Both Java and C++ are often used to implement the calculations in a network environment. For example, it could be the database that shares data over the network (the server application) or the client-side processing the data received from the database via a network (numerical or simulation calculations based on data received from the database).

Therefore, it seems important in terms of efficiency of communication between distributed objects, to test interoperability of these objects in different systems, namely the client and server implemented in C++, client and server implemented in Java and integrated mixed C++ – Java and Java – C++.

Another advantage of the CORBA standard is the ability to use different system platforms (such as Windows, UNIX etc.), as opposed to a dedicated technology, for example distributed communication technologies on the .NET.

2. The structure of the test application

To test the efficiency of interaction of distributed objects, based on the CORBA standard an application was implemented with the aim to calculate the so-called perfect numbers in a chosen numerical range. A perfect number is the natural number equal to the sum of its divisors, excluding this number. The greatest perfect even number was calculated in 2001, and its value is $2^{3466917} - 1$ [9].

This problem, despite its simplicity is a good example to show the possibility of scattering calculations and collaboratively distributed objects. The perfect number search algorithm is very simple, which allows to minimize the differences between the way of its implementation in C++ and Java, and thus better test the communications of the distributed objects (implemented in different programming languages) using the CORBA standard. The functional diagram of the computation of the perfect number (at the server side) is shown in Fig. 1.

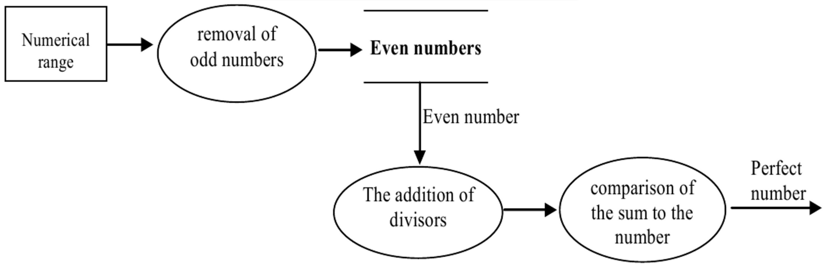

Fig. 1. The scheme of the algorithm of the perfect number computation

To communicate with the ORB core, the test application uses a static IDL stub on the client side and the static IDL skeleton on the server side, which involves the implementation of an IDL file that contains the primary interface. The IDL file content (Fig. 2) is not complicated, has one definition of sequences and an interface containing the method to calculate the perfect number. The method is passed two parameters, one specifying the current scope of the calculation, the second is the number of the thread, which is the number of the server. Simplicity of the IDL file is the intended treatment, this facilitates testing and leads to the transparent code.

```
module DoskMod{
        typedef sequence<long> LongSeq;

        interface Dosk{
                LongSeq  Oblicz(in long zakres,in long ns);
        };
};
```

Fig. 2. The IDL interface

During the startup, the client receives the three input parameters: the range of numbers, the number of available (running) servers and the number of threads. Threads have been used for parallelization of calculations. The numerical scope is divided into approximately equal sub-numbers, dependent on the number of servers (or clients or threads – depending on the test scenario – Section 3).

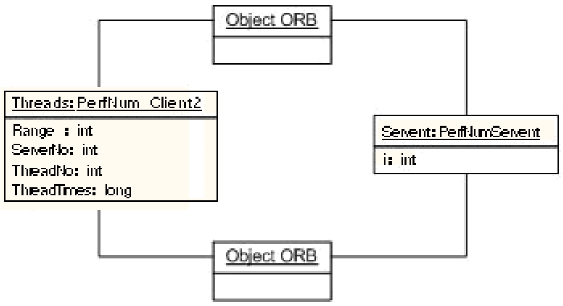

Fig. 3. The diagram of the objects for the test application

The application object diagram of the test application was presented in Fig. 3. Here the object representing a thread which initiates the execution of the program refers to the ORB object to send requests of the appointment of a perfect number by the server.

The object servant realizes the task of finding the perfect numbers in a given numerical input range and after the calculations forwards the result (the array of the found perfect numbers) through the ORB object to the thread object, which expects the solution.

3. The research methodology

All tests were carried out on a network (switch 1GB, Ethernet 1GB) of computers with the same hardware platform and configuration shown in Tab. 1.

Tab. 1. The hardware platform of computers

Itemt	**Parameter**
Procesor type	Intel Pentium 4, 2800MHz
RAM	1014 MB
Disk	ST380013AS (80GB, 7200 RPM, SATA)
Network Card	Realtek RTL8139/810x Family Fast Ethernet

The tests were run on the operating system Linux Fedora Core kernel-6.2.1912, Sun Microsystems Java Virtual Machine version 1.6.0_14 and the environment MICO

2.3.13. to run applications written in C++. The built-in library org.omg.CORBA was used to run the CORBA mechanism implemented in Java. The general schema for tests was as follows:

- launch the server application,
- remote the IOR file copy to the client,
- launch the client application with the respective switches.

Each of the following scenarios was carried out for all the implementations on average ten times. This was to calculate the average result of the test and to eliminate errors caused by the packed collision over the network.

3.1. Test scenarios

The scenario "1 on 1": Two computers were used. The client application was run on the first one and the server application on an other one (Tab. 2).

The scenario configuration "X on 1" also consists of two computers, where on one of them was running the server, while the client program on the second one, but through the threads (2, 4, 6, 8 (Tab. 3)) more clients were simulated. This scenario was designed to check the server performance during the calculation demands of a higher number of clients at the same time – each thread sends other sub-range of numbers.

Tab. 2. Scenario "1 on 1"

Server number	1		
Thread number	1		
Range	1000	10000	1000000

Tab. 3. Scenario "X on 1"

Server number	1		
Thread number	2, 4, 6, 8		
Range	1000	10000	1000000

Tab. 4. Scenario "1 on X"

Server number	2, 4, 6, 8, 10, 12			10, 12
Thread number	2, 4, 6, 8, 10, 12			10, 12
Range	1000	10000	100000	1000000

However, in the other scenario, the test designated as "1 in X", there is only one client and a variable number of servers. The scenario has been tested for 2, 4, 6, 8, 10 and 12 computers that are running servers (Tab. 4). Also in this case, the client uses threads in the same quantities as the number of servers in order to send a range of calculations in approximately the same time. The purpose of this scenario was to present the speedup of calculation with respect to the number of servers.

All scenarios were carried out for two implementations of C++ and JAVA. The task of scenarios was also checking the implementation functionality between client and server programming in these languages. Therefore, all these scenarios have been run in client-server patterns presented in Tab. 5.

Tab. 5. Client-server patterns (programming languages)

Client	**Server**
C++	C++
C++	Java
Java	C++
Java	Java

4. Obtained results and discussion

This chapter presents a comparison of the results, visually presented in charts, structured according to the scenarios presented in the previous chapter. The measured time covers: the clients execution time depends on the clients threads execution times, communication time and remote object execution time. The startup time of a Java virtual machine was not included. The client (or each client in the scenario "X on 1") made only one synchronous call of the remote object (the client waits for results).

Additionally, the following figures (Figs 4, 5 and 6) present the execution times for the client and for the method of the remote object (at the server side).

The scenario "1 on 1"

After comparing the above three charts (Figs 4, 5 and 6), one can say that the most optimal configuration is such, when the server and the client are implemented in C++. The differences are almost double for the 10000 range for the implementation of the client. One can also conclude that when the server is implemented in C++ calculations are performed relatively faster than when it is implemented in Java.

In Figs 4 and 5 the difference between the client time and the server time is shown. This is due to the initialization time of ORB (function CORBA::ORB_init()). For

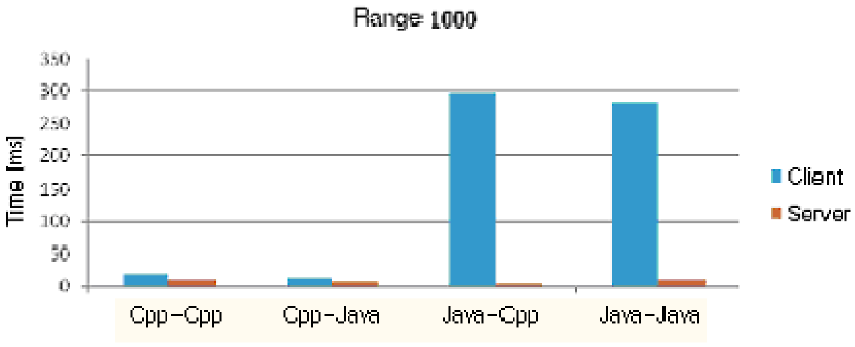

Fig. 4. The calculation time (ms) for the range 1000, the scenario 1-1

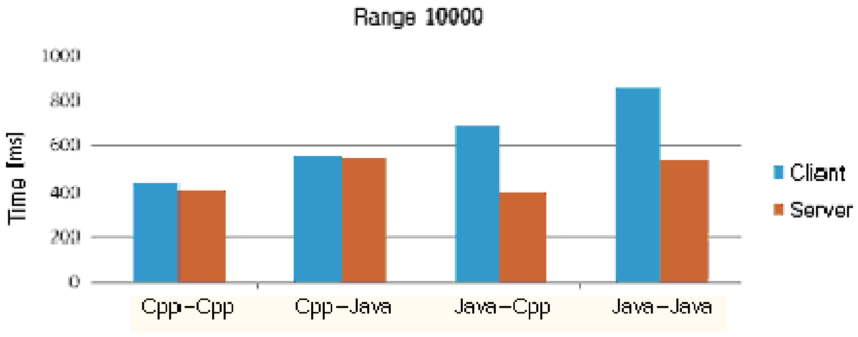

Fig. 5. The calculation time (ms) for the range 10000, the scenario 1-1

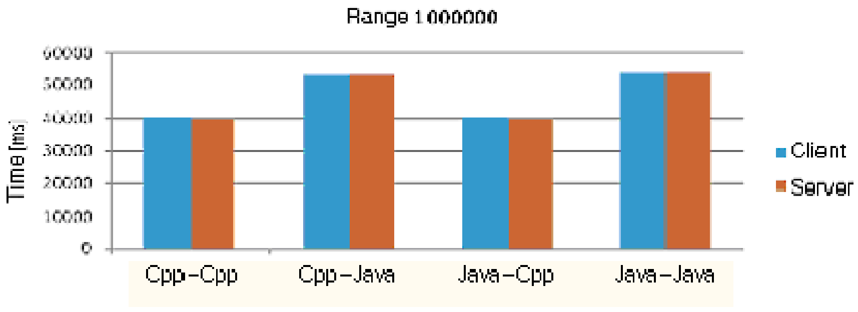

Fig. 6. The calculation time (ms) for the range 1000000, the scenario 1-1

the range 1000000 (Fig. 6) the ORBs initialization time is not significant. Especially for the case of the client implementation in Java the ORBs initialization time is dominant.

Scenario "X on 1"

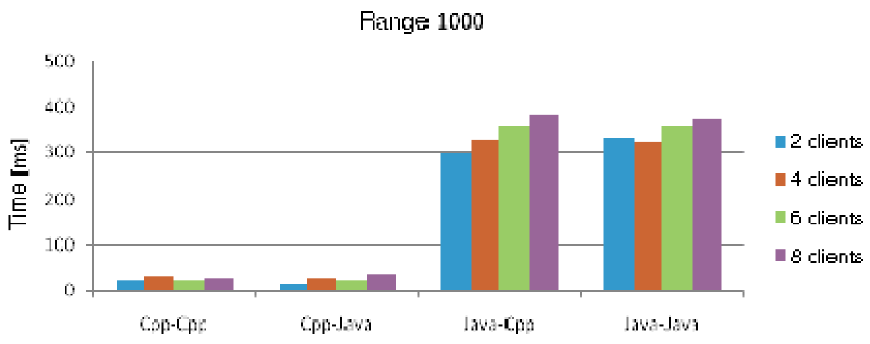

Fig. 7. The calculation time (ms) for the range 1000, scenario X-1

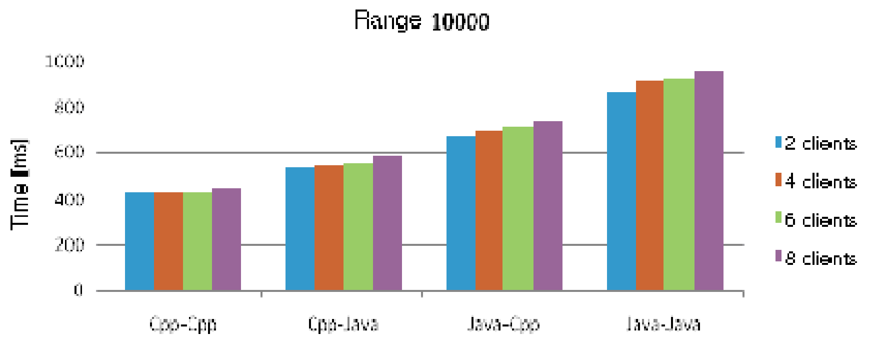

Fig. 8. The calculation time (ms) for the range 10000, the scenario X-1

In this scenario the range of numbers was divided into the approximately equal sub-ranges which number depends on the number of threads (the whole range of numbers is divided by the number of threads).

The implementation in C++, in all three ranges is the quickest. Charts (Figs 7, 8 and 9), however, did not show a clear conclusion about the mixed implementation. In the case of the range 1000000 one can see that the calculation time for the tested configuration client in Java and the server in C++ is comparable with the implementation of the two components in C++. And when the 10000 range is used

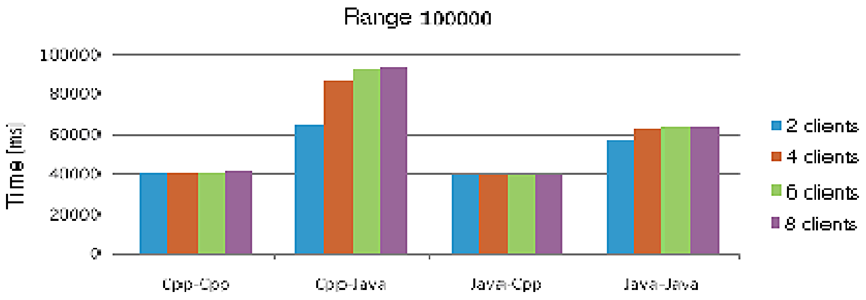

Fig. 9. The calculation time (ms) for the range 1000000, the scenario X-1

calculations in the schema JAVA-C++ for example for eight clients are about 288 [ms] slower than in C++.

Moreover, for the range 1000 and 10000 calculations in the schema C++-Java are comparable with these when the two components are implemented in C++. The implementation of the client and server in Java is still the slowest of the tested ones.

In Figs 7 and 8 the ORBs initialization time is still significant (like in the scenario “1 on 1”). Additionally, the number of threads decreases efficiency because of the startup time of threads and the scheduling time of threads. For the range 1000000 (Fig. 9) for the three configurations: C++-C++, Java-C++ and Java-Java the most significant is the calculation of the perfect numbers in so large range of numbers, that the server in C++ is the quickest. Only for the case of Java-Java the impact of the threads startup time is slightly visible.

Unexpected results were obtained for the case of C++-Java. Probably the long execution time at the server side resulted in the fact that the clients processes went into the sleeping mode. In the case of Java-Java the similar effect is not observed probably because of a different execution way of Java threads in the environment of the operating system.

The scenario “1 on X”

The fact that the implementation in C++ is faster than any other described in the paper, was predictable. All of the graphs (Figs 10, 11, 12 and 13) presented below confirm this fact. However, the difference in the 1000, between the implementation in C++ and Java, using the twelve servers, amounts to almost 500 [ms]. This demonstrates that the Java program is 34 times slower.

For the ranges of 1000 and 10000 implementations for the schemas C++-C++ and C++-Java are the fastest. Moreover, for these schemas (for the ranges 10000 and 100000) the efficiency increases with the number of servers as well as for the range 100000 for the schemas Java-C++ and Java-Java. Even for the range of 1000 the increase in the number of servers decreases the efficiency. In this case, the greatest impact on the performance has the ORBs initialization time at the servers

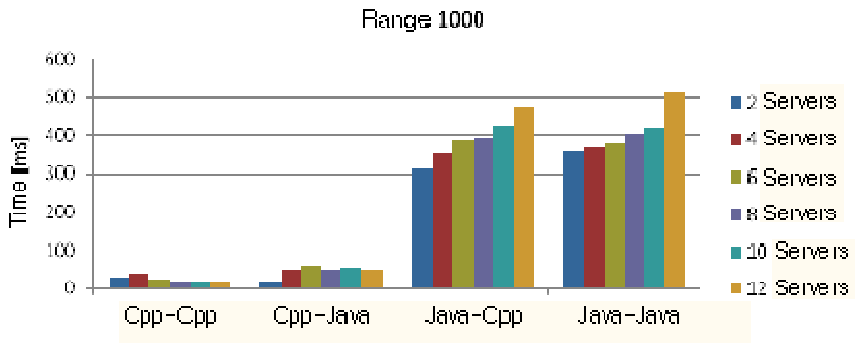

Fig. 10. The calculation time (ms) for the range 1000, the scenario 1-X

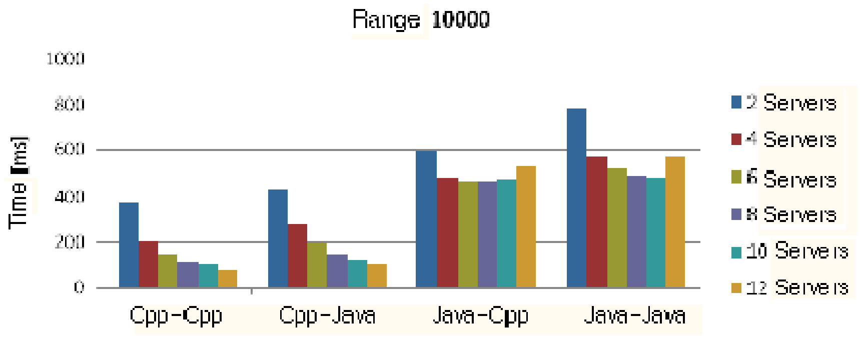

Fig. 11. The calculation time (ms) for the range 10000, the scenario 1-X

side. Moreover, over a short time of calculation (the small range of numbers) the communication overhead is significant too. Therefore, the increase in the number of servers increases the execution time.

Fore the range of 10000 (Fig. 11) for the schemas Java-C++ and Java-Java first the increase of server numbers causes the increase of the application performance but for 12 servers, causes a sudden decrease of performance. Too many servers cause the increase in the communication overhead which is significant in comparison with the perfect numbers calculation.

For small ranges (Figs 10 and 11), similar results can be observed in the case of the schemas C++-C++ and C++-Java. By contrast, for greater ranges (Figs 11 and 12) the similar results can be observed for the schemas C++-C++ and Java-C++. It should be also mentioned that for small ranges a significant impact on the obtained results has the time of initialization of ORB objects, and for large ranges it has a minimal impact.

The application written in Java for almost all charts achieves the highest values. The implementation in C++ is the opposite, as almost all the tests show that it is the fastest in execution. However, for the range of 100000 (Fig. 12) the performance of

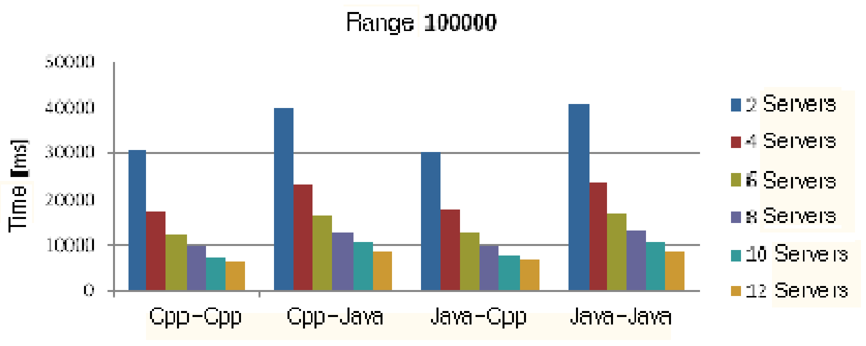

Fig. 12. The calculation time (ms) for the range 100000, the scenario 1-X

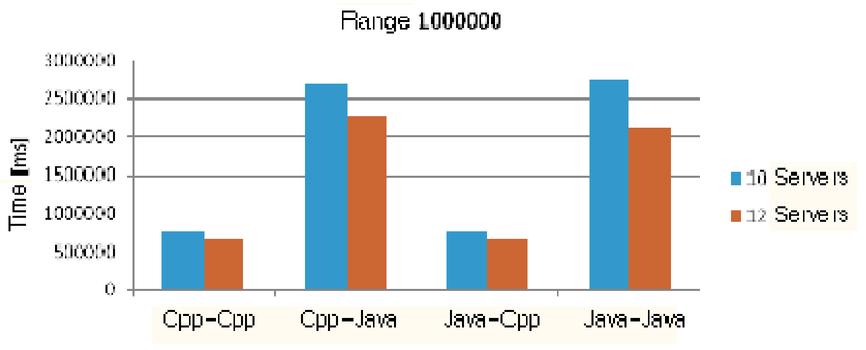

Fig. 13. The calculation time (ms) for the range 1000000, the scenario 1-X

Java-Java definitely improves (is similar to C++-C++) which is particularly evident for the 10 and 12 servers. By contrast, for the range 1000000 (Fig. 13) a significant deterioration of the performance for the schema Java-Java relatively to C++-C++ can be seen. This may imply that Java begins to lose its effectiveness with too large computational tasks regardless of the number of servers.

5. Summary and conclusions

In this work opportunities of the usage of the CORBA middleware for distributed objects for programming languages Java and C++ were presented, with particular emphasis on the joint implementation. It may be important for the implementation of object-oriented or object-relational databases and distributed computing object-oriented applications. CORBA is a standard independent of a hardware architecture or programming language, which makes it a very affordable tool to implement distributed computing projects. But it is not a simple mechanism. Its independence is overpayed by the need to use external tools, such as MICO, or omniORB for C++. In the alternative mechanisms such as RMI, EJB, DCOM, .NET Remoting or WCF there are no such complexities, but in these standards there is no so great freedom to use a programming language and hardware as well as system platform.

After conducting all the test scenarios (Chapter 3), analyzing the results shown in the charts (Chapter 4), it can be concluded that the implementation of the server and the client in C++ is the most effective method of the distributed object interaction, for example, in determining perfect numbers. However, in the respective ranges, the mixed implementations: C++-Java and Java-C++ matched the efficiency of C++-C++.

Mixed implementations can be used in client server systems in which numerical calculations are performed in the language C++. For example the server implemented in Java provides data from a database to the client – implemented in C++ performing numerical calculations (for example some simulations).

Easiness of writing in Java has been fraught with a high computation time, but the use of CORBA in C++ was necessary to use an external server MICO. Each schema has its own advantages and disadvantages. The use of a programming language depends on the type and size of calculations, as is well illustrated by implementations of the mixed schemes.

6. References

[1] Lausen G., Vossen G.; *Models and Languages of Object Oriented Databases*, WNT, Warszawa 2000.

[2] Vossen G.; *Bibliography on Object-Oriented Database Management*, Technical report No. 9301, Computer Science Group, Univ. of Giessen, Germany, 1993.

[3] Onderka Z.; *Computer Aided Network Access to Data from Geological and Geophysical Databases* (in Polish), in: Kozielski S., Małysiak B., Kasprowski P., Mrozek D. (ed.), *Bazy Danych, Rozwój Metod i Technologii II*, WKiŁ, Warszawa 2008, pp. 219–229.

[4] Piórkowski A., Onderka Z.; *Project and Implementation of the Geological Data Base in the Internet Network* (in polish), in: Pochopień B., Kwiecień A., Grzywak A., Klamka J. (ed.), *New Technologies for the Computer Networks*, WKiŁ, Warszawa 2006.

[5] Stonebraker M.; *Object-Relational DBMSs – The Next Great Wave*, Morgan Kaufmann, CA, 1996.

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

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

[8] Siegel J.; *CORBA Fundamentals and Programming* J. Wiley, New York 1996.

[9] Wibig T.; University of Łódź Physics Dept. Perfect Numbers. Available via `http://www.u.lodz.pl/~wibig/hieronim/hie15pok.htm`.

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

[11] Gupta S.; *A Performance Comparison of Windows Communication Foundation (WCF) with Existing Distributed Communication Technologies*, Available via `http://msdn.microsoft.com/en-us/library/bb310550.aspx`.

[12] *What Is Windows Communication Foundation?*, Available via `http://msdn.microsoft.com/en-us/library/ms731082.aspx`.

[13] Onderka Z., Uhruski P.; *The SDK for the Process Migration in the Heterogeneous Computer Networks*, Schedae Informaticae, 11, 2002, pp. 99–114.

Received September 22, 2010