STUDIA INFORMATICA 2011
Volume 32 Number 3A (98)

Zdzisław ONDERKA
Akademia Górniczo-Hutnicza,
Katedra Geoinformatyki i Informatyki Stosowanej
Marek CICHY
Bielska Wyższa Szkoła im. Tyszkiewicza

PORÓWNANIE WYDAJNOŚCI KOMUNIKACJI DLA STANDARDÓW CORBA I DCOM W SYSTEMACH TYPU KLIENT-SERWER[1]

Streszczenie. W artykule przeanalizowano i porównano wydajność komunikacji rozproszonych obiektów, realizujących funkcjonalności klienta i serwera na podstawie standardów CORBA i COM/DCOM, w celu pokazania możliwości ich zastosowania do realizacji systemów typu klient-serwer. Przedstawiono obiektowo zorientowany model aplikacji testowej. Przeanalizowano czasy transmisji dla danych typów podstawowych, dla tablic jednowymiarowych oraz struktur danych.

Słowa kluczowe: obiekty rozproszone, CORBA, COM/DCOM, klient-serwer, obliczenia rozproszone

THE COMPARISON OF THE COMMUNICATION EFFICIENCY FOR THE CORBA AND DCOM STANDARDS IN THE CLIENT-SERVER SYSTEMS

Summary. In this work there was analyzed and compared the communication efficiency of the distributed objects which were realized the client and the server functionalities based on the CORBA and the COM/DCOM standards in order to show the possibilities of its application to the client-server systems implementation. The object-oriented model was presented for the testing application. It was analysed transmission times for the basic data types, the tables and the structures.

Keywords: distributed objects, CORBA, COM/DCOM, client-server, distributed calculations

[1] Temat realizowany w ramach prac statutowych Katedry Geoinformatyki i Informatyki Stosowanej WGGiOŚ AGH.

1. Obiekty rozproszone

Wobec rosnącego znaczenia technologii obiektowej od dłuższego czasu prowadzi się prace nad realizacją obiektowości w systemach rozproszonych, począwszy od rozproszonych aplikacji obliczeniowych, intensywnie wykorzystujących zaawansowane metody numeryczne, aż po obiektowe bądź obiektowo-relacyjne systemy bazowo-danowe.

Dla potrzeb realizacji obiektów rozproszonych zdefiniowano wiele standardów, z których ważniejsze to standard CORBA, zrealizowany przez konsorcjum OMG [1], EJB (ang. *Enterprise JavaBeans*) [9] i RMI (ang. *Remote Method Invocation*) [10] dedykowane dla Javy, standardy dedykowane COM/DCOM [6, 7, 8], .NET Remoting [11] oraz WCF (ang. *Windows Communication Foundation*) [4,5] firmy Microsoft oraz inne. W standardach tych obiekty komunikują się poprzez warstwę pośrednią, a lokalizacja obiektów jest przeźroczysta. Ponadto, technologie te silnie bazują na różnych odmianach interfejsów.

1.1. CORBA

W 1989 roku powstało konsorcjum OMG (ang. *Object Management Group*), skupiające producentów sprzętu i oprogramowania. Zadaniem konsorcjum było między innymi zdefiniowanie standardów dotyczących zastosowania metod obiektowych w heterogenicznych systemach rozproszonych. Efektem działalności OMG było zdefiniowanie architektury OMA (ang. *Object Management Architecture*) [1] z jej najważniejszym składnikiem, jakim jest CORBA (ang. *Common Object Request Broker*) [3, 7], opartym na zdefiniowanym w warstwie pośredniej oprogramowania mechanizmie wymiany obiektów ORB (ang. *Object Request Broker*) [2]. Można go też traktować jako interfejs pośredniczący między składnikami sprzętowymi a programowymi dla różnych producentów. Innym istotnym efektem pracy konsorcjum OMG było zdefiniowanie standardu komunikacji pomiędzy ORB-ami, czyli protokołu IIOP (ang. *Inter ORB Protocol*), bazującego na TCP/IP i zastosowanego w standardzie CORBA od wersji 2.0.

Jedną z najważniejszych zalet standardu CORBA jest możliwość implementacji obiektów rozproszonych w różnych językach programowania (np. C/C++, Java, Ada, Pyton, Smalltalk) oraz możliwość zastosowania na różnych platformach systemowych (np. Windows, UNIX), w odróżnieniu od standardów dedykowanych dla konkretnych języków programowania (np. RMI, EJB) lub środowisk systemowych (np. .NET Remoting, COM/DCOM, WCF). Ponadto, CORBA poprzez serwisy i usługi dostarcza udogodnień związanych z obiektami, jak usługi zdarzeń, transakcji i bezpieczeństwa.

1.2. COM/DCOM

Standard COM (ang. *Component Object Model*), dedykowany dla środowiska Windows definiuje API pozwalające na tworzenie złożonych aplikacji z gotowych komponentów oraz na współdziałanie takich komponentów w środowisku. Warunkiem współpracy jest stosowanie się do binarnej struktury wyspecyfikowanej przez Microsoft. DCOM (ang. *Disributed Component Object Model*) [6] natomiast definiuje protokół pozwalający komponentom COM na współpracę w środowisku sieci komputerowych. Stanowi, więc zbiór dodatkowych funkcji, pozwalających na połączenie ze zdalnymi komputerami. COM oraz DCOM są dostarczane w postaci zestawu bibliotek, oferującego wszystkie te możliwości [8].

Obiekt COM ukrywa przed klientem szczegóły implementacji. Klient komunikuje się z obiektem wyłącznie za pomocą interfejsów. Sam interfejs nie jest klasą ani obiektem, ale jest zbiorem funkcji, za pomocą których można uzyskać dostęp do metod obiektu COM. Każdy obiekt COM musi zostać utworzony wewnątrz serwera COM, przy czym jeden serwer może utworzyć wiele takich obiektów.

Dla niektórych aplikacji COM został zastąpiony przez mechanizmy środowiska .NET Framework oraz przez wsparcie dla usług sieciowych (ang. *Web Services*) przez WCF (ang. *Windows Communication Foundation*). Jednakże obiekty COM/DCOM mogą być używane ze wszystkimi językami środowiska .NET, poprzez technologię .NET COM Interop.

2. Projekt aplikacji testowej

Projekt zakładał stworzenie dwóch aplikacji typu klient-serwer przesyłających między sobą zestaw danych testowych, przy czym w celu uzyskania jak najbardziej miarodajnych wyników, obie aplikacje powinny być w jak największym stopniu podobne do siebie. W praktyce oznacza to budowę programów serwerów i klientów na podstawie identycznych szablonów projektowych środowiska programistycznego.

Do oceny efektywności komunikacji przewidziany jest pomiar czasu, który jest potrzebny do przesłania ustalonej ilości danych wybranych typów prostych i strukturalnych (tablic oraz struktur).

2.1. Analiza zapotrzebowań

2.1.1. Wymagania funkcjonalne aplikacji testowej

Wymagania użytkownika w stosunku do aplikacji testowej:

- wysyłanie testowego zestawu zmiennych wybranych typów,

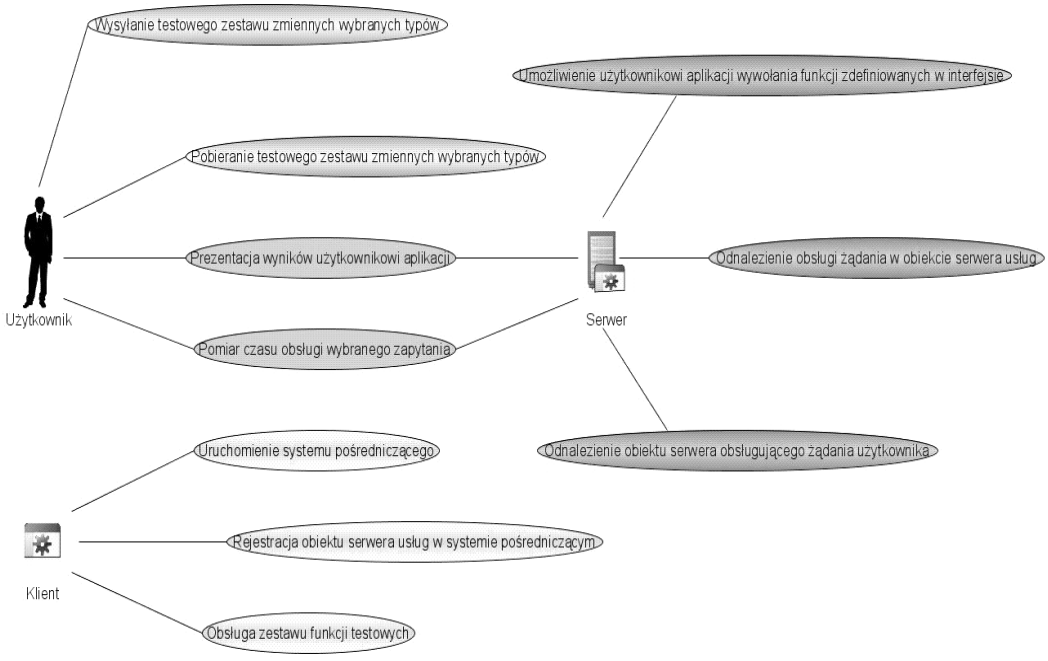
- pobieranie testowego zestawu zmiennych wybranych typów,
- pomiar czasu obsługi wybranego zapytania,
- prezentacja wyników użytkownikowi aplikacji.

Wymagania programu klienta w stosunku do programu serwera usług:

- uruchomienie systemu pośredniczącego,
- rejestracja obiektu serwera usług w systemie pośredniczącym,
- obsługa zestawu funkcji testowych.

Wymagania programu serwera usług w stosunku do programu klienta:

- umożliwienie użytkownikowi aplikacji wywołania funkcji zdefiniowanych w interfejsie,
- odnalezienie obiektu serwera obsługującego żądania użytkownika,
- odnalezienie obsługi żądania w obiekcie serwera usług,
- pomiar czasu obsługi żądania użytkownika,
- prezentacja wyników użytkownikowi aplikacji.



Rys. 1. Diagram przypadków użycia
Fig. 1. Use-case diagram

2.1.2. Minimalne wymagania niefunkcjonalne

- Sprzęt: Komputer klasy PC z procesorem Intel Pentium 2, 256 MB RAM.
- Oprogramowanie: system operacyjny MS Windows oparty na jądrze NT, pakiet oprogramowania Microsoft Visual Studio 2005, pakiet oprogramowania omniORB ver. 4.1.2.

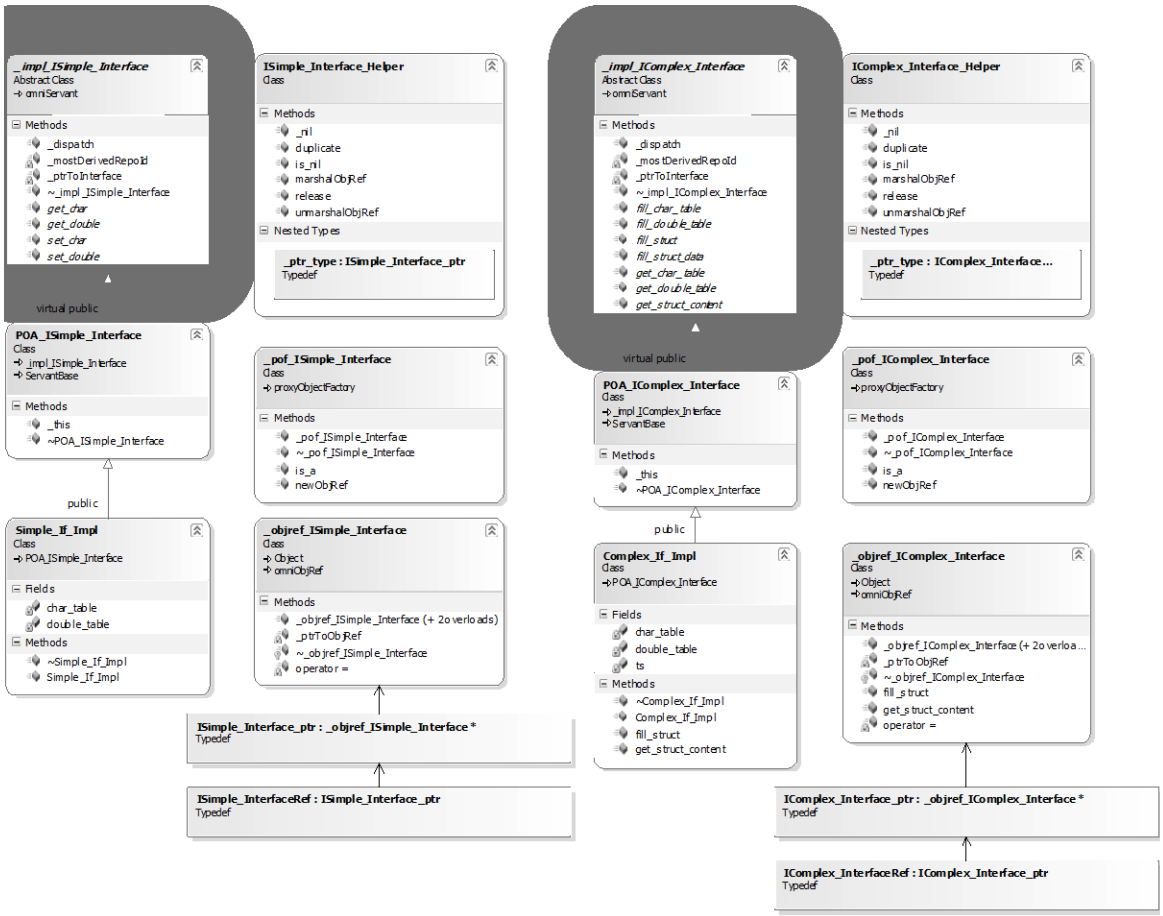
2.2. Opis funkcji testowych (w języku IDL)

- `char getChar(in unsigned short _index)` – funkcja pobierająca daną typu `char` z określonego indeksu tablicy jednowymiarowej;
- `double getDouble(in unsigned short _index)` – funkcja pobierająca daną typu `double` z określonego indeksu tablicy jednowymiarowej;
- `void setChar(in char c,in unsigned short _index)` – funkcja umieszczająca daną typu `char` na określonej pozycji tablicy jednowymiarowej;
- `void setDouble(in double d, in unsigned short _index)` – funkcja umieszczająca daną typu `double` na określonej pozycji tablicy jednowymiarowej;
- `void getCharTable(inout ct charTable)` – funkcja pobierająca tablicę jednowymiarową danych typu `char`;
- `void getDoubleTable(inout dt doubleTable)` – funkcja pobierająca tablicę jednowymiarową danych typu `double`;
- `void fillCharTable(in ct charTable)` – funkcja wypełniająca tablicę jednowymiarową danymi typu `char`;
- `void fillDoubleTable(in dt doubleTable)` – funkcja wypełniająca tablicę jednowymiarową danymi typu `double`;
- `Test_Structure getStructContent()` – metoda pobierająca zawartość struktury;
- `void fillStructData(in char c, in double d, inout ct charTable, inout dt doubleTable)` – funkcja umieszczająca dane wewnątrz struktury obiektu serwera;
- `void sendStruct(in Test_Structure ts)` – funkcja wysyłająca strukturę do obiektu serwera.

2.3. Hierarchia klas serwera w technologii CORBA

Interfejs obiektów klienta i serwera dla aplikacji testowej, aby działać w różnych środowiskach programistycznych został zdefiniowany w języku opisu obiektów IDL (ang. *Interface Definition Language*), niezależnym od języka programowania. Pliki wygenerowane przez pakiet omniORB po kompilacji tego interfejsu zawierają definicje klas (rys. 2) modułów *proxy* i *stub*, czyli obiektów pośredniczących w wymianie danych pomiędzy obiektami serwera usług i klienta oraz definicje funkcji, pozwalających na przesyłanie komunikatów pomiędzy obiektami CORBA (ang. *marshalling/unmarshalling*). Wywołania są kodowane do postaci strumienia bajtów przesyłanych od klienta do serwera poprzez kanał transmisyjny. Odpowiedź serwera jest również kodowana i odsyłana do klienta. Procesy kodowania i dekodowania odbywają się w modułach *proxy* i *stub*, odpowiednio klienta i serwera. Poza tym

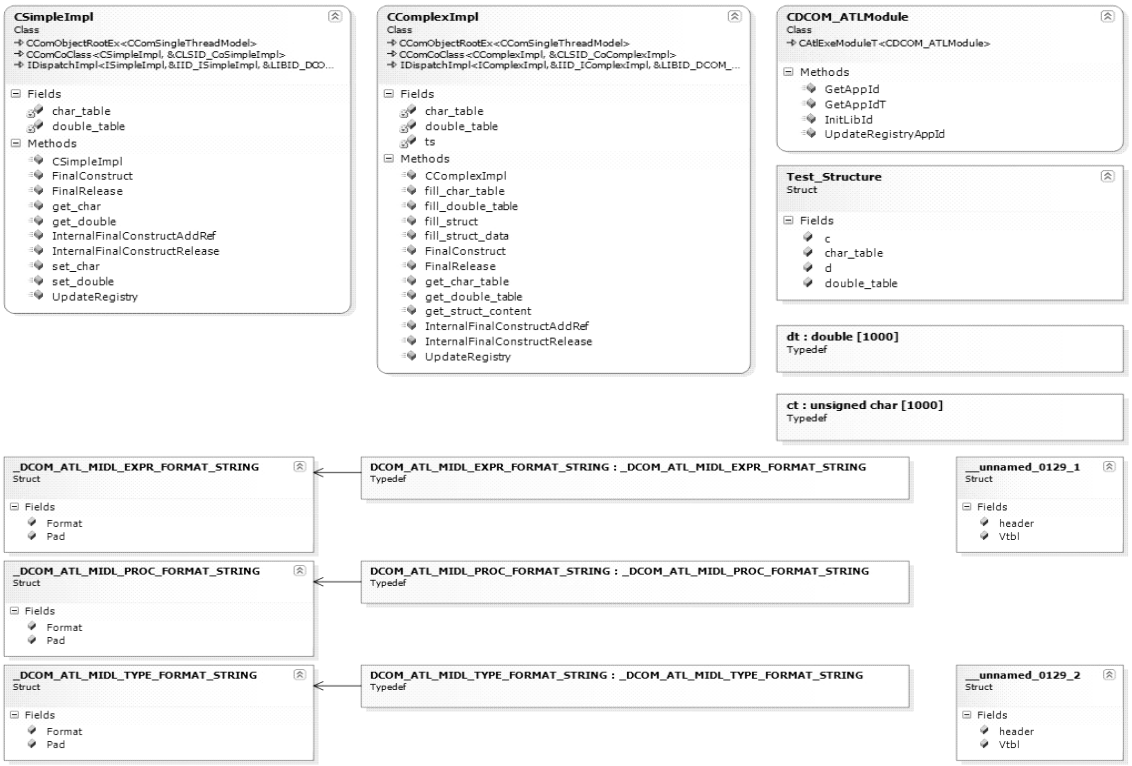
w plikach zawarte są również deklaracje i szkielety definicji funkcji klas dziedziczących z interfejsu, które programista sam wypełnia definicjami funkcji.



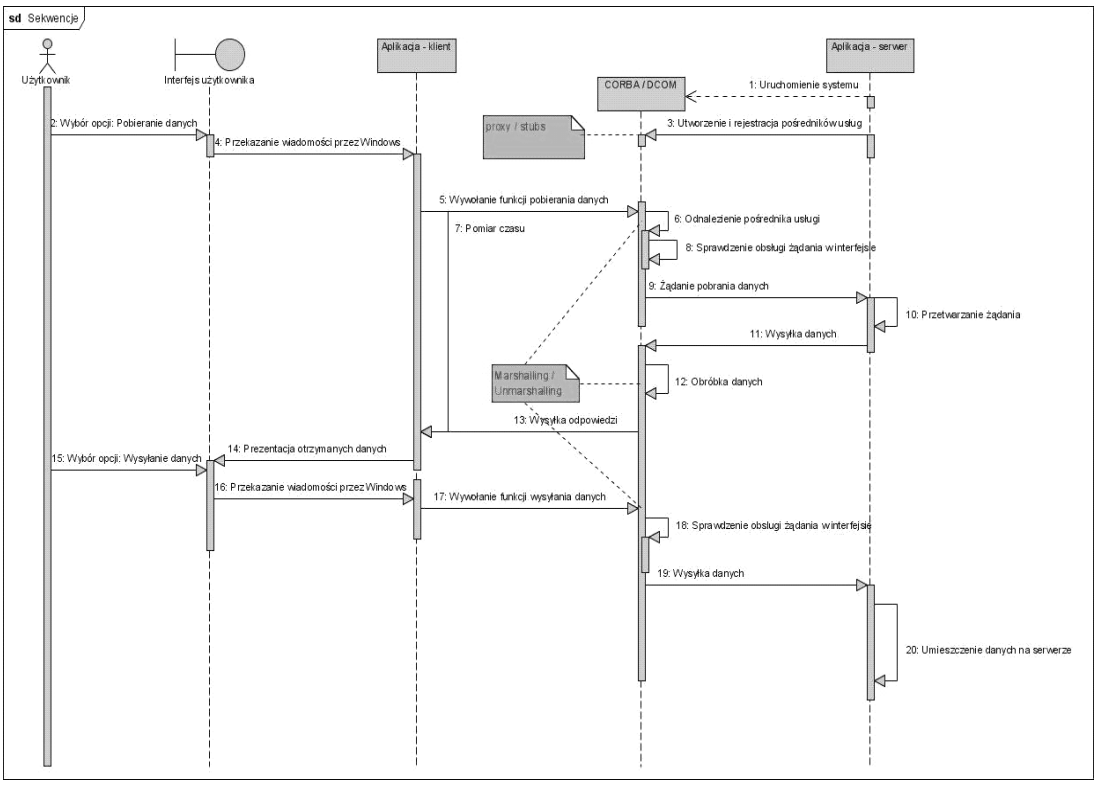
Rys. 2. Diagram klas serwera dla standardu CORBA
Fig. 2. Class diagram for server application for CORBA standard

2.4. Hierarchia klas serwera w technologii DCOM

Po kompilacji pliku interfejsu (w języku IDL) przez program MIDL.exe zestaw bibliotek DCOM generuje pliki zawierające identyfikatory klas i typów, używane potem do rejestracji w systemie Windows oraz definicje modułów *proxy* i *stub*, czyli obiektów pośredniczących w wymianie danych pomiędzy obiektami serwera usług i klienta (ang. *marshalling/unmarshalling*). Wszystkie deklaracje dotyczące interfejsów są napisane w postaci struktur, a klasy implementujące funkcje dziedziczą obowiązkowo z interfejsu *IUnknown* (rys. 3).



Rys. 3. Diagram klas serwera dla standardu DCOM (ATL)
Fig. 3. Class diagram for server application for DCOM (ATL) standard

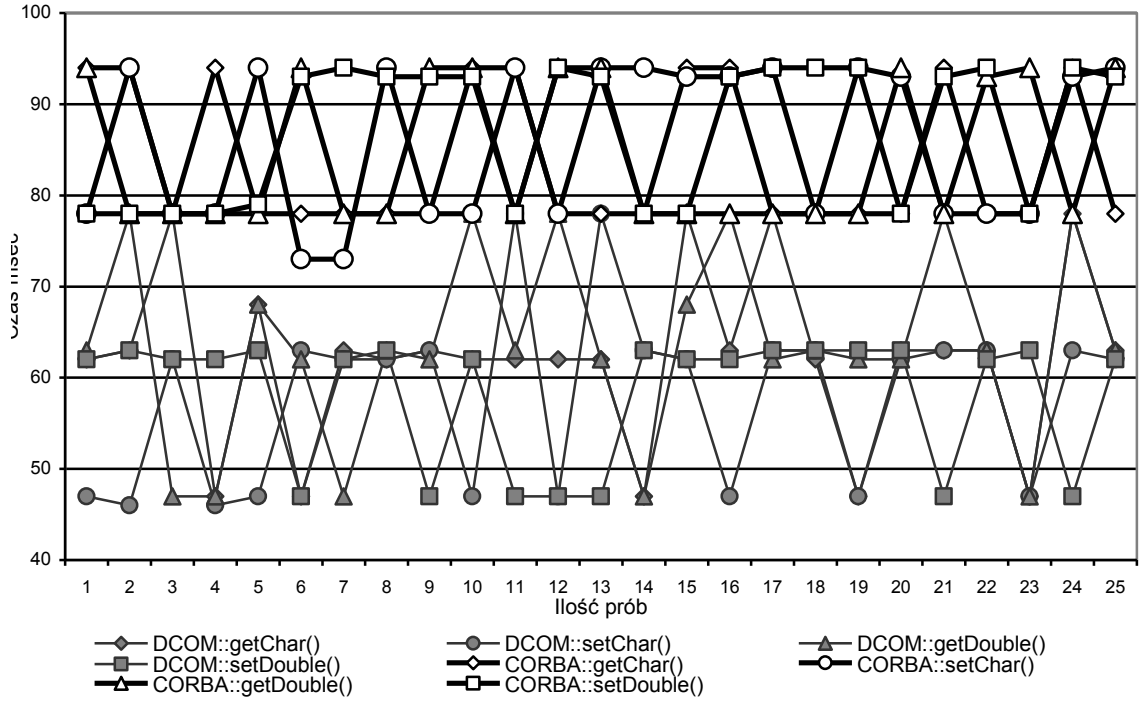


Rys. 4. Diagram sekwencji przepływu komunikatów
Fig. 4. Sequence diagram for message flow

3. Testy aplikacji

Testy wydajności przesyłania danych pomiędzy programem serwera a programem klienta aplikacji zostały przeprowadzone lokalnie na komputerze przenośnym. W celu otrzymania policzalnych wyników w implementacji klienta zastosowane zostało wielokrotne wywołanie metod zdalnego obiektu w pętli (1000 razy). Dla zapewnienia względnej stałości środowiska testowego i uniezależnienia się od chwilowych wahań obciążenia procesora i pamięci, dla każdego testu przeprowadzonych zostało 25 prób, a następnie obliczono średnie wartości. Przesyłane tablice znakowe i zmiennoprzecinkowe miały rozmiar 1000 elementów, natomiast struktura składała się z czterech pól: składowej znakowej (`char`), składowej zmiennoprzecinkowej (`double`), tablic 1000 elementowych znakowej i zmiennoprzecinkowej.

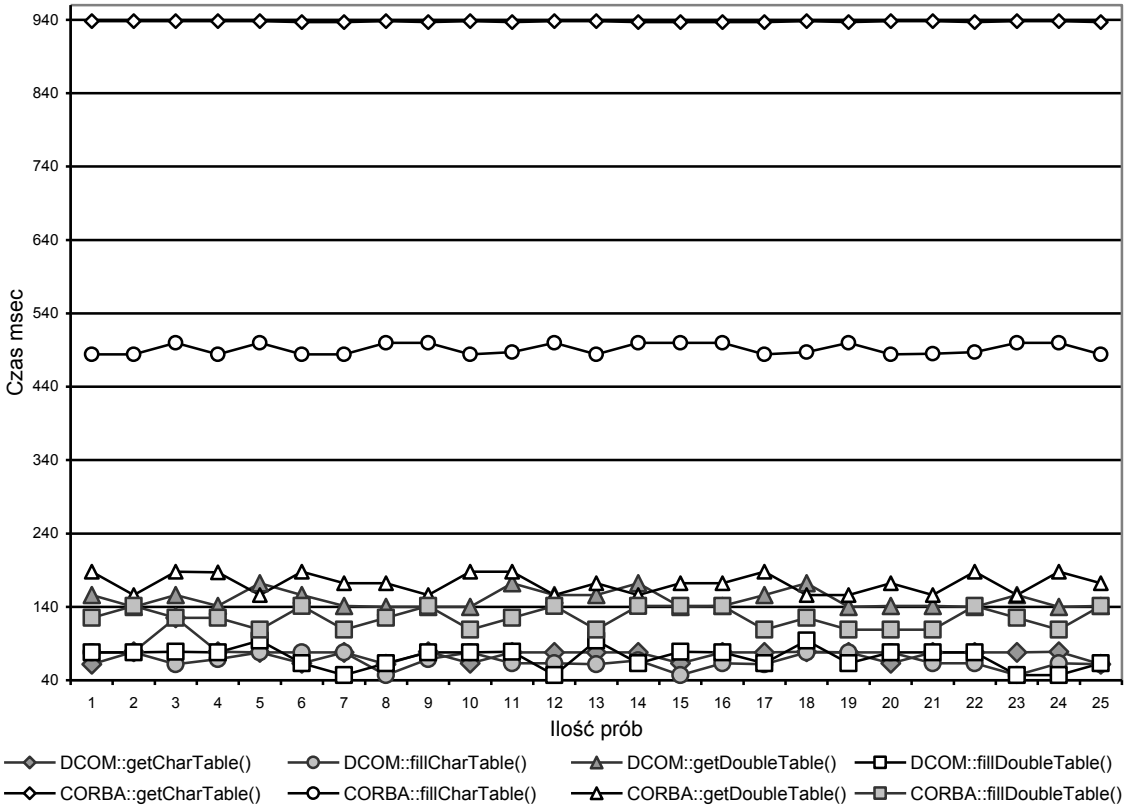
Wyniki pomiarów (czasy w milisekundach) przedstawione są w tabelach 1-3.



Rys. 5. Testy przesyłania danych typów podstawowych
Fig. 5. The basic data types transfer tests

Tabela 1
Średnie czasy przy przesyłaniu danych typów podstawowych [ms]

	getChar()	setChar()	getDouble()	setDouble()
DCOM	62,12	58,12	63,56	58,16
CORBA	87,76	85,76	84,36	87,32



Rys. 6. Testy przesyłania tablic
Fig. 6. The table transfer tests

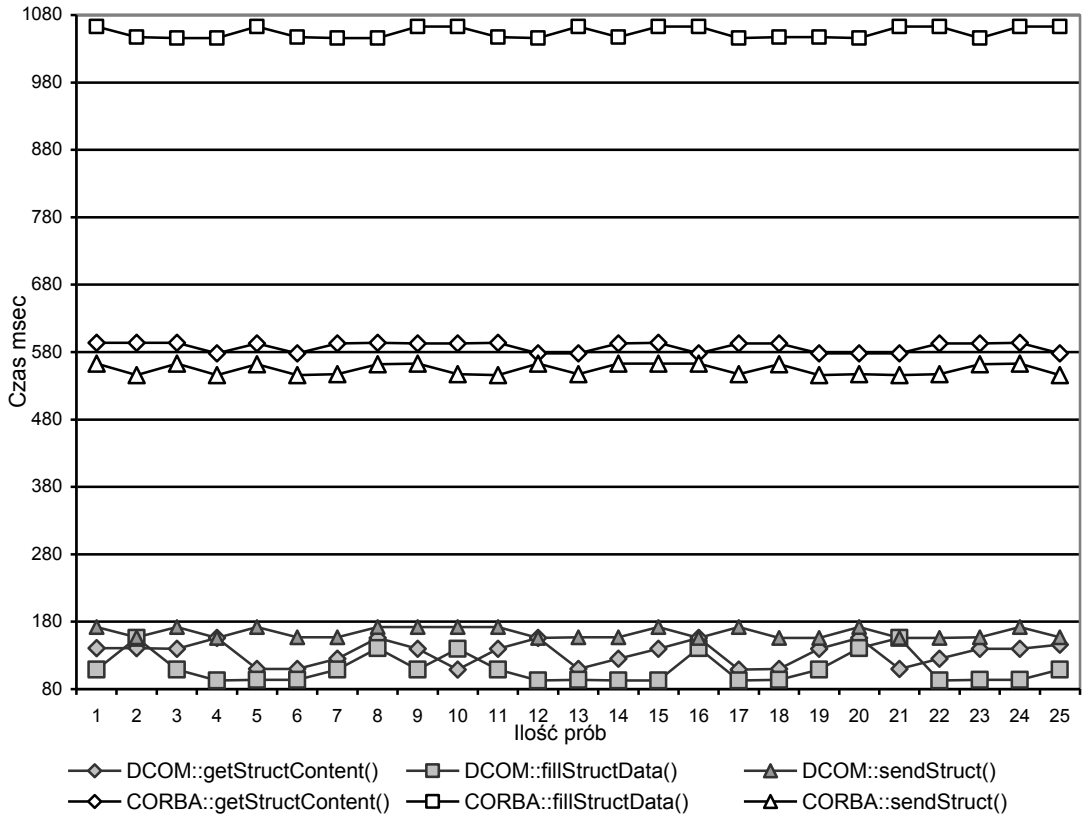
Tabela 2

Średnie czasy przy przesyłaniu tablic [ms]

	getCharTable()	fillCharTable()	getDoubleTable()	fillDoubleTable()
DCOM	75,76	66,96	149,84	71,48
CORBA	937,56	491,44	171,96	125,0

4. Podsumowanie i wnioski

Stworzenie od podstaw aplikacji typu klient serwer dla środowiska rozproszonego jest zadaniem pracochłonnym, nawet pomimo tego, że dużą część pracy wykonuje za nas kompilator interfejsu (języka IDL), dostarczony z pakietem oprogramowania omniORB. Natomiast, implementacja projektu w technologii DCOM została przeprowadzona za pomocą szablonu projektu wykorzystującego ATL (ang. *Active Template Library*), z uwagi na zalecenia firmy Microsoft dotyczące tworzenia obiektów COM.



Rys. 7. Testy przesyłania struktur
Fig. 7. The structure transfer tests

Tabela 3

Średnie czasy przy przesyłaniu struktur [ms]

	getStructContent()	fillStructData()	sendStruct()
DCOM	133,24	110,4	163,32
CORBA	587,88	1053,22	554,24

Test przesyłania typów podstawowych wykazuje niewielką przewagę technologii Microsoft pod względem szybkości transferu, natomiast ORB działa stabilniej i rozrzut wyników pomiaru czasu jest znacznie mniejszy (rys. 5, tab. 1).

Dla tablic typów podstawowych różnice są już znaczne, przy czym widać, że ORB znacznie wolniej przesyła dane typu znakowego. Jeżeli chodzi o stabilność pracy, sytuacja wygląda podobnie jak przy pierwszym teście (rys. 6, tab. 2).

Przesyłanie struktur oraz wypełnianie ich pól przez program klienta daje podobne rezultaty. Implementacja DCOM przoduje pod względem wydajności, program korzystający z ORB jest za to bardziej stabilny (rys. 7, tab. 3). Dodatkowo, czas wypełniania struktury z użyciem CORBA jest prawie dwukrotnie dłuższy niż czas pobrania jej zawartości, który jest zbliżony do czasu jej przesłania. Takiego efektu nie daje użycie obiektów DCOM.

Należy pamiętać jednak, że projekt ten symuluje tylko pracę w rzeczywistym środowisku rozproszonym. Jako, że do transportu danych w sieci używane są dwa odrębne protokoły

(DCE RPC oraz IIOP), dlatego aktualnie realizuje się implementację aplikacji klienta i serwera dla wykonania testów na dwóch niezależnych komputerach, połączonych siecią bezprzewodową (również dla porównania z siecią przewodową). Oczywiście ważne jest też to, że technologia CORBA jest dużo bardziej uniwersalna i dostępna w ogromnej większości systemów operacyjnych. Ponadto, do projektowania aplikacji użyto kompletu produktów Microsoft (system operacyjny i środowisko programistyczne), można by więc pokusić się o próbę stworzenia aplikacji w innym IDE.

Podsumowując, testy pokazały przewidywane, lepsze rezultaty dla standardu dedykowanego do środowisk systemowego i programistycznego, niż standard niezależny i ogólny jakim jest CORBA. Uniwersalność CORBY opłacana jest mniejszą efektywnością choć możliwość implementowania obiektów w różnych językach programowania (różnych producentów środowisk programistycznych czy kompilatorów) jest tu bardzo dużą zaletą.

Obie technologie należą do raczej skomplikowanych i wymagają gruntownego zapoznania się z dokumentacją, jeśli programista nie zamierza ograniczyć się do kompilacji plików z przykładami, aczkolwiek użycie szablonu projektu ATL oszczędza konieczności zapoznania się ze wszystkimi szczegółami implementacji.

BIBLIOGRAFIA

1. Object Management Group: Object Management Architecture Guide. OMG Document Number 92.11.1 Revision 2.0, 1992.
2. Object Management Group: The Common Object Request Broker: Architecture and Specification, Version 2.0. OMG Document, 1995.
3. Siegel J.: CORBA Fundamentals and Programming. J. Wiley, New York 1996.
4. Gupta S.: A Performance Comparison of Windows Communication Foundation (WCF) with Existing Distributed Communication Technologies: http://msdn.microsoft.com/en-us/library/bb310550.aspx.
5. What Is Windows Communication Foundation?: http://msdn.microsoft.com/en-us/library-/ms731082.aspx.
6. Opis technologii DCOM http://devcentral.iftech.com/articles/DCOM/intro_DCOM/part1-/4.php.
7. http://neo.dmcs.p.lodz.pl/rso – CORBA, DCOM: Podstawy, zasada działania.
8. Zestaw bibliotek dla standardów COM/DCOM http://www.microsoft.com/downloads/en/
9. Monson-Haefel R., Burke B.: Enterprise JavaBeans3.0, Fifth Edition. O'Reilly Publisher.
10. Bielecki J.: Java 3 RMI. Podstawy programowania rozproszonego. Helion, Gliwice 1999.

11. .NET Remoting Overview http://msdn.microsoft.com/en-us/library/kwdt6w2k(v=vs.71)-.aspx.

Recenzenci: Prof. dr hab. inż. Bolesław Pochopień
Dr inż. Piotr Pikiewicz

Wpłynęło do Redakcji 12 marca 2011 r.

Abstract

In this work was presented the efficiency comparison of the communication between distributed object using standard CORBA and COM/DCOM. The CORBA standard is independent on the different programming languages and system platforms. COM/DCOM is dedicated to the Microsoft operating systems and Mictrosoft programming environments.

After executing all testing scenario (chapter 3) i.e sending the data of the simple types (char and double), sending the 1000 elements character table and floating point table and sending the structure consists of one character data, one floating point data and two 1000 elements tables (character and floating point) the received results where presented on the graphic chart (Chapter 3). The results shows that the COM/DCOM standard is quicker but CORBA is more stable (smaller differences in obtained results in identical tests).

Both technologies are rather complex and require a thorough look at the documentation although the use of ATL project template saves you having to get acquainted with all the details of implementation.

Adres

Zdzisław ONDERKA: Akademia Górniczo-Hutnicza, Katedra Geoinformatyki i Informatyki Stosowanej, WGGiOŚ, Al. A. Mickiewicza 30, 30-059 Kraków, Polska, onderka@ii.uj.edu.pl, zonderka@agh.edu.pl