JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 18 (2021) 37–41
Received: 04 December 2020
Accepted: 06 January 2021

Comparison of frameworks for creating web services using the Axis2/C and gSOAP examples

Porównanie technologii tworzenia usług sieciowych na przykładzie Axis/C i gSOAP

Roman Bondarev*, Beata Pańczyk

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The main purpose of this paper was to compare two frameworks (Axis2/C and gSOAP) that implement the SOAP protocol, a protocol for the exchange of structured messages in a computer environment. The paper describes experiments showing the level of SOAP performance depending on the components of each framework and the methods of their implementation, such as: parsing, serialisation, working with various message formats, the time effectiveness of protocol, the efficiency of message processing models, etc. After the practical part was completed, performance studies of each of the frameworks were carried out, the collected data helped to define a more efficient and faster tool for transferring data between the server and the client.

Keywords: web-services; serialisation; deserialisation; SOAP

Streszczenie
Głównym celem tego artykułu było porównanie dwóch frameworków (Axis2/C and gSOAP), które implementują protokół SOAP do wymiany ustrukturyzowanych komunikatów w środowisku komputerowym. W artykule opisano eksperymenty badające wydajność SOAP dla wskazanych frameworków oraz metody ich implementacji, takie jak: parsowanie, serializacja, praca z różnymi formatami wiadomości, efektywność czasowa protokołu, efektywność modeli przetwarzania wiadomości itp. Do wykonania badań przygotowano aplikacje testowe, które umożliwiły porównanie wydajności każdego z frameworków. Analiza zebranych danych pozwoliła wskazać wydajniejsze i szybsze narzędzie do przesyłania danych pomiędzy serwerem a klientem, którym okazał się gSOAP.

Słowa kluczowe: web-serwisy; serializacja; deserializacja; SOAP

*Corresponding author
Email address: **roman.bondarev@pollub.edu.pl** (R. Bondarev)

1. Wstęp

W systemach informatycznych przez wiele lat istniał wymóg komunikowania się przez Internet, niezależnie od platformy, języka, lokalizacji lub technologii, w której zostały tworzone aplikacje. Rozwój XML, powszechnego formatu wymiany danych oraz akceptacja architektury zorientowanej na usługi, ze strony największych liderów rynkowych, sprawiły, że usługi sieciowe, to dziś zjawisko wszechobecne. Przez dwadzieścia lat rozwoju webserwisów najbardziej rozbudowanymi i popularnymi narzędziami do tworzenia usług sieciowych w języku C++ stały się gSOAP i Axis/C, oparte na protokole SOAP [1].

Wymagania aplikacji opartych na architekturze web-serwisowych są zróżnicowane, a dostępne frameworki są zaprojektowane tak, żeby spełniały różne wymagania programistów. Każdy z nich posiada określoną wydajność „end-to-end", wydajność serializacji i deserializacji, określony poziom wykorzystania pamięci, skalowalność itp.

Obecnie szybkość realizacji usług sieciowych to główne kryterium dla użytkowników. Jeżeli wymiana danych zajmuje zbyt dużo czasu, to spowoduje, że klienci będą szukać innych sposobów komunikowania. Taka teza jest właściwa w odniesieniu do wszelkich nowoczesnych aplikacji: bankowych, aplikacji przeznaczonych do przesyłania multimediów i informacji tekstowych (serwisy społecznościowe), aplikacji obsługujących sklepy internetowe, czy rezerwacji miejsc [2].

W ciągu ostatnich lat, zwłaszcza w związku z rewolucją mobilną, wiele działań gospodarczych i społecznych zostało przeniesionych do sieci. Z tego powodu postrzegana przez użytkowników wydajność sieci jest obecnie podstawowym miernikiem nowoczesnych usług sieciowych. Ponieważ przepustowość pozostaje stosunkowo tania, opóźnienie w sieci jest aktualnie główną przeszkodą w poprawie wydajności. Co więcej, wiadomo, że opóźnienie sieci powoduje zmniejszenie zysku. Na przykład Amazon szacuje, że każde 100 ms opóźnienia zmniejsza zyski o 1% [3].

Głównym celem niniejszego artykułu było przeprowadzenie analizy porównawczej dwóch popularnych frameworków implementujących SOAP w języku C++ (Axis/C i gSOAP) oraz wskazanie bardziej wydajnego narzędzia do przesyłania danych w sieci.

2. gSOAP i Axic/C

Protokół SOAP jest protokołem, czyli mechanizmem transportu informacji. Przenoszone informacje są uporządkowane (posiadają strukturę i typy). Dane zapisane są w języku XML. Za pomocą SOAP można przenosić różnego rodzaju dane, które są kodowane w razie potrzeby do postaci wymaganej przez XML.

W przypadku przesyłania danych protokołem SOAP, stosuje się następujący algorytm:

1. Analiza struktury przesyłanych danych - określenie typu i rozmiaru zmiennych do przesłania przez sieć.
2. Konwersja danych na format ASCII/UTF (niezbędne dla XML). W przypadku ciągów ASCII konwersja jest prosta i szybka. Konwersja liczb całkowitych na ASCII obejmuje konwersję binarną na dziesiętną. Konwersja liczb zmiennoprzecinkowych na ASCII wymaga również konwersji binarnej na dziesiętną, ale konwersja zmiennoprzecinkowa jest znacznie bardziej skomplikowana niż konwersja liczb całkowitych.
3. Zapis danych ASCII w formacie XML w pamięci. Sposób przechowywania ciągu znaków, może wpływać na liczbę wymaganych operacji z pamięcią.
4. Wysłanie komunikatu HTTP i XML przez system operacyjny.

W przypadku odbierania danych postępuje się analogicznie, tylko poszczególne kroki algorytmu są wykonywane w odwrotnej kolejności.

Operacje te wpływają na działanie całego systemu. W każdym z dwóch analizowanych frameworków etapy te realizowane są w różny sposób.

gSOAP obsługuje serializowanie podstawowych typów danych C/C++ do składni XML, co oznacza, że dowolny typ danych, obiekt, struktura, zdefiniowane przez użytkownika, muszą być konwertowane do postaci najprostszych typów przed ich transmisją. Narzędzia gSOAP zapewniają powiązanie SOAP/XML z językami C i C++, aby ułatwić rozwój usług internetowych i dostępność aplikacji klienckich na szerokim zakresie urządzeń. Framework zawiera kompilator, za pomocą którego automatycznie mapuje natywne i zdefiniowane przez użytkownika typy danych C i C++ do typów danych XML i odwrotnie. gSOAP wykorzystuje techniki „xml-predictive" i „pull-based", co oznacza, że podczas serializacji/deserializacji koperty XML, w zależności od poziomu jej komplikacji, odbywa się podwyższenie lub obniżenie częstotliwości procesora, niezbędnej do realizacji konkretnej operacji. Wykorzystanie takiej techniki parsowania XML pozwala osiągnąć większą wydajność i zmniejszyć wykorzystanie pamięci RAM.

Z kolei Axis korzysta z własnego modelu obiektowego AXIOM i API do parsowania XML (StAX) w celu zwiększenia wydajności [4]. Biblioteka AXIOM zapewnia implementację modelu obiektowego zgodnego z XML, który obsługuje budowę drzewa obiektów na żądanie. Również obsługuje model, który umożliwia wyłączenie budowania drzewa i bezpośredni dostęp do strumienia za pomocą interfejsu API StAX. Posiada również wbudowaną obsługę pakietu XML Optimized Packaging (XOP) i MTOM, których kombinacja pozwala XML na wydajną i przejrzystą transmisję danych binarnych. Połączenie tych czynników stanowi łatwy w użyciu interfejs API o wysokiej wydajności. Opracowany jako część Apache Axis2, Apache AXIOM jest rdzeniem Apache Axis2, ale również może być używany niezależnie od Apache Axis2 [5].

Jedną z kluczowych zalet Axis jest to, że przechowuje logikę i dane w oddzielnych komponentach, co pozwala zoptymalizować wydajność podczas przesyłania danych.

Protokół transportowy w omawianych frameworkach też jest wykorzystywany w różny sposób. Axis korzysta z pamięci RAM do zapisywania informacji w postaci drzewa DOM, po czym, w porządku kolejki, wysyła dane do klienta. W tym względzie gSOAP jest zupełnie inny od Axis. Korzysta z gniazd API i częściowo zapisuje dane bezpośrednio do strumienia wyjściowego tych gniazd.

Wykorzystanie pamięci może być czynnikiem decydującym o wdrożeniu aplikacji usług internetowych na urządzeniach wbudowanych i serwerach o dużym obciążeniu. Duże zużycie pamięci wpływa na wydajność wielowątkowego serwera, który musi obsługiwać wiele jednoczesnych połączeń. Ponadto wzrost wykorzystania pamięci dynamicznej może negatywnie wpłynąć na efektywność pamięci podręcznej. Chociaż alokator i dealokator ułatwiają alokację pamięci z punktu widzenia programisty, może to stanowić problemem dla wydajności aplikacji.

Podejście Axis do rozwiązania tego problemu, to możliwość użycia obiektu WSDLWrapper, który pozwala zoptymalizować zużycie pamięci. Niestety nie wiadomo jakie techniki wykorzystuje w celu zmniejszenia tego zużycia, a w dokumentacji można znaleźć tylko "Implementacja WSDL Definition wykorzystuje różne strategie do kontrolowania zużycia pamięci" [6].

gSOAP unika kosztownych operacji kopiowania buforów przy użyciu algorytmu dwóch iteracji. Pierwsza iteracja ocenia strukturę danych i oblicza rozmiar danych do przesyłania (w bajtach), druga iteracja ładuje dane bezpośrednio do gniazda. Także wykorzystuje dodatkowe techniki zmniejszające zużycie pamięci, m.in. technikę parsowania "schema-specific" w celu zmniejszenia wymagań dotyczących konsumpcji pamięci podczas pracy z XML przy użyciu jednego, wstępnie przydzielonego bufora dla wejścia/wyjścia, wykorzystanie TLB (ang. translation lookaside buffer) dla szybkiego dostępu do zawartości XML oraz kompresji HTTP w celu zmniejszenia rozmiaru wiadomości.

3. Przegląd literatury

Bardzo podobna tematyka do yej poruszanej w niniejszym artykule, została podjęta w pracy [7]. Prace dotyczyły procesorów XML i były prowadzone przez pracowników Uniwersytetu Stanu Nowy Jork we

współpracy z Uniwersytetem Stanu Florydy. Autorzy sugerują, aby wybór frameworka uzależniać od wydajności procesora XML [7]. Wbrew tej opinii, nie można ignorować innych modułów, które wpływają na działanie SOAP, np.: protokoły TCP/IP i HTTP, zarządzanie kolejkami, wielowątkowość, automatyczna dealokacja pamięci (ang. *garbage collector*), przepustowość sieci itp.

W pracy profesorów Helsińskiego Instytutu Technologii Informacyjnych [8] skupiono się na wykorzystaniu protokołów transportowych i ich zorganizowaniu tak, żeby były maksymalnie wydajne w zestawieniu z pozostałymi elementami SOAP. Wówczas nie istniał jeszcze osobny i powszechnie akceptowany standard komunikacyjno-transportowy. W pracy tej określono i omówiono kwestie wykorzystania SOAP: możliwość asynchronicznych zapytań, wykorzystanie web-serwisów na telefonach komórkowych, wykorzystanie pamięci głównej, metody kompresji danych i metody ulepszenia istniejących protokołów do przesyłania danych. Wyniki przedstawionych tam badań nie określają wydajności SOAP jako monolitycznego, wyspecyfikowanego i ogólnie przyjętego narzędzia. W badaniu tym SOAP jest narzędziem, a nie przedmiotem badania.

4. Metoda badań

W środowisku programistycznym Visual Studio stworzono cztery aplikacje: serwerowe i klienckie, przy użyciu każdego z porównywanych frameworków.

Badania były przeprowadzone na komputerze o następujących parametrach:

- system operacyjny: Microsoft Windows 10 Pro x64;
- CPU: Intel Core i5-6200U Skylake;
- GPU: AMD Radeon R5 M330;
- RAM: Samsung 4 GB.

Wykorzystano oprogramowanie:

- IDE: Microsoft Visual Studio Community 2017 15.9.23;
- Build acceleration: IncrediBuild 1.5.0.3;
- Platform toolset: Visual Studio 2015 (v140).

Testowano frameworki w wersjach:

- Apache Axis2/C (axis2c-src-1.6.0);
- gSOAP 2.8.101.

Aplikacje wykorzystano do testowania:

- opóźnienia czasowe w przypadku wysyłania i odbierania wywołań SOAP;
- serializacji/deserializacji dla różnych typów danych (Void, Double, String, Integer, Struct, obrazów typu MIME image/jpg);
- wykorzystania pamięci RAM.

Badania przeprowadzone zostały dla różnych typów danych, począwszy od tych najmniej obciążających protokół transportowy i silnik XML. Dzięki temu można było łatwo zauważyć wzrost czasu i mocy obliczeniowych podczas zwiększania rozmiaru i złożoności danych przesyłanych przez sieć. Rezultaty otrzymano po przeprowadzeniu dziesięciu pomiarów wydajności dla zbioru danych liczących od dziesięciu do miliona zmiennych, obiektów.

5. Wyniki testów

W tabelach 1, 3 oraz na rysunkach 1, 3 porównywano wyniki pomiarów czasu przesyłania danych różnego typu. W tabelach 2, 4 i na rysunkach 2,4 przedstawiono wykorzystanie pamięci.

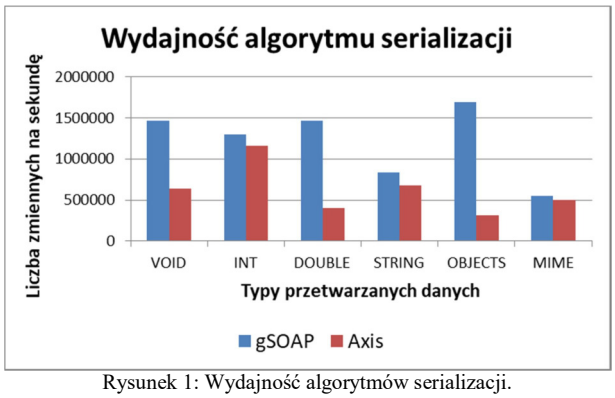
Axis miał problemy ze stabilnością, które uniemożliwiały ukończenie testów dla danych składających się z ponad 5000 zmiennych.

Model obiektowy realizowany w Axis, w porównaniu z gSOAP, wypada gorzej prawie we wszystkich testach, oprócz przetwarzania danych typu string. Jednak dane w postaci ciągu znaków są najprostsze do przetwarzania, ponieważ przekształcenie do postaci ciągu znaków jest jednym z etapów serializacji [9].

Tabela 1 i rysunek 1 pokazują, że w przypadku serializacji, we wszystkich przypadkach gSOAP jest szybszy niż Axis. W przypadku deserializacji (Tabela 3, Rys. 3), gSOAP działa również szybciej. Skrótem [zm/s] określono liczbę przesyłanych zmiennych w ciągu sekundy.

Tabela 1: Porównanie wydajności algorytmu serializacji.

Serializacja			
Typ danych	**gSOAP [zm/s]**	**Axis [zm/s]**	**Różnica [%]**
VOID	1,461,965	639,146	128.74%
INT	1,301,413	1,163,028	11.90%
DOUBLE	1,466,225	400,229	266.35%
STRING	832,806	681,776	22.15%
OBJECTS	1,688,468	313,886	437.92%
MIME	549,936	495,487	10.99%

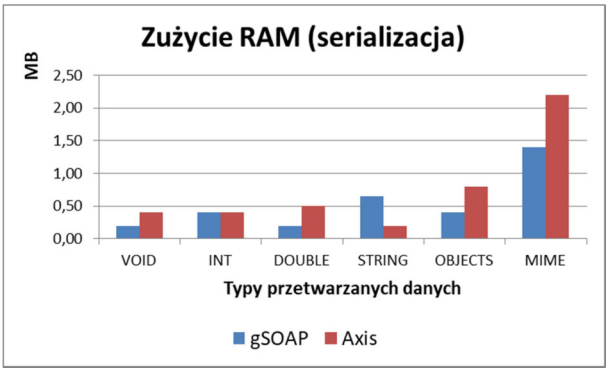


Rysunek 1: Wydajność algorytmów serializacji.

Tabela 2: Porównanie zużycia RAM algorytmem serializacji.

RAM (serializacja)			
Typ danych	**gSOAP (MB)**	**Axis (MB)**	**Różnica (%)**
VOID	0.2	0.4	50.00%
INT	0.4	0.4	0.00%
DOUBLE	0.2	0.5	60.00%
STRING	0.65	0.2	225.00%
OBJECTS	0.4	0.8	50.00%
MIME	1.4	2.2	36.36%

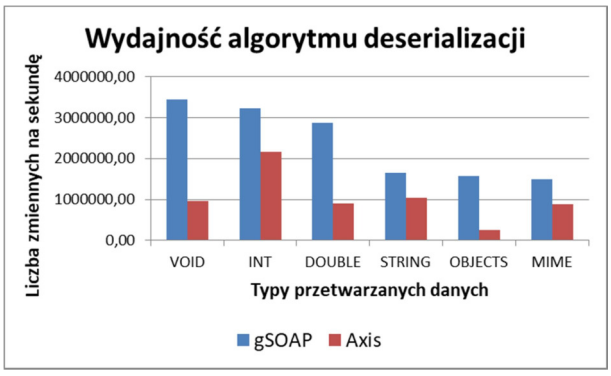
W przypadku przekształcania danych typu string z punktu widzenia zużycia pamięci (Rys. 2) jest lepszy Axis. W tym przypadku algorytm nie przekształca danych do postaci ASCII\UTF a są one bezpośrednio mapowane do składni XML.



Rysunek 2: Zużycie pamięci podczas serializacji.

Tabela 3: Porównanie wydajności algorytmów deserializacji.

Deserializacja			
Typ danych	**gSOAP (zm/sek)**	**Axis (zm/sek)**	**Różnica (%)**
VOID	3,441,712	963,241	257.31%
INT	3,233,707	2,159,858	49.72%
DOUBLE	2,863,494	899,128	218.47%
STRING	1,644,745	1,047,961	56.95%
OBJECTS	1,577,937	244,914	544.28%
MIME	1,490,912	875,489	70.29%

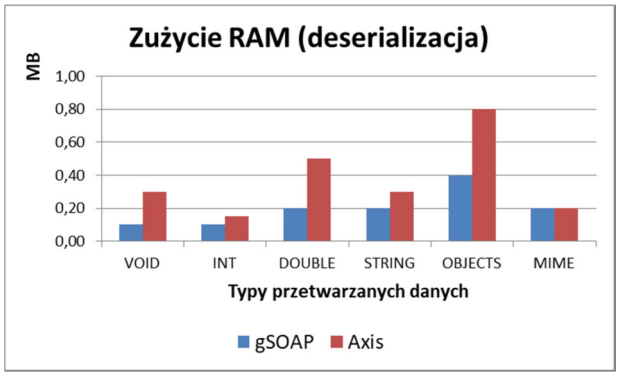


Rysunek 3: Wydajność algorytmów deserializacji.

Tabela 4: Porównanie zużycia RAM algorytmem deserializacji.

RAM (Deserializacja)			
Typ danych	**gSOAP (MB)**	**Axis (MB)**	**Różnica (%)**
VOID	0.1	0.3	66.67%
INT	0.1	0.15	33.33%
DOUBLE	0.2	0.5	60.00%
STRING	0.2	0.3	33.33%
OBJECTS	0.4	0.8	50.00%
MIME	0.2	0.2	0.00%

W przypadku wytwarzania aplikacji, które muszą przetwarzać duże zbiory danych liczbowych, rozsądniej będzie korzystać z gSOAP. Jeżeli architektura serwera jest oparta na paradygmacie programowania OOP i transmisji danych w postaci obiektów to gSOAP również jest wydajniejszym rozwiązaniem. Gdy głównym założeniem aplikacji jest wymiana plików multimedialnych - gSOAP zużywa mniej pamięci.



Rysunek 4: Zużycie pamięci podczas deserializacji.

Zużycie pamięci przez aplikację SOAP jest zwykle niskie. Rozmiar kodu zależy liniowo od rozmiaru schematów WSDL i/lub XML, które są mapowane na typy C/C++. Wykorzystanie pamięci podczas realizacji zależy od rozmiaru danych, które mają być przechowywane w czasie wykonywania aplikacji, które obejmują rozmiar zmiennych C/C++ serializowanych i deserializowanych w XML. Dane są bezpośrednio serializowane w formacie XML. Pewna ilość pamięci stosu i sterty jest używana przez kontekst silnika gSOAP, który ma wewnętrzny bufor IN/OUT (rozmiar bufora to 64 KB), przechowywany na stosie w celu optymalizacji transferu danych (z wykorzystaniem gniazd). W rezultacie standardowej kompilacji aplikacji klienckiej, jej całkowity rozmiar to 224 Kb (20 Kb jest przydzielane na stercie i 66 Kb na stosie).

Optymalizacja wydajności w Axis odbiła się na zużyciu pamięci podczas działania aplikacji. Możliwość wysłania wielu buforów danych za pośrednictwem pojedynczego wywołania skutkuje tym, że niezbędne jest przechowywanie wszystkich danych w pamięci. Podobnie przesyłanie dużej ilości danych przez sieć jednocześnie znacznie zwiększa czas transmisji danych.

Po zbadaniu przepustowości łącza dla danych potrzebujących fragmentacji (tabela 5) pomiary pokazały, że w przeciwieństwie do Axis, gSOAP eliminuje opóźnienie w czasie potrzebne do rozbicia danych, co wskazuje na różne realizacje protokołów wykorzystywanych podczas działania frameworków, a także na różne sposoby ich wykorzystania. gSOAP, podobnie jak Axis oferuje standardowy zakres narzędzi do sterowania protokołem TCP.

Tabela 5: Porównanie wydajności TCP+HTTP dla różnych rozmiarów danych.

Ping			
Fragmentacja danych	**gSOAP (pak/sek)**	**Axis (pak/sek)**	**Różnica (%)**
Tak	930	616	50.92%
Nie	968	342	183.26%

gSOAP, korzysta z TCP Hybla, który ma na celu wyeliminowanie opóźnienia połączenia TCP. Ten protokół zamyka połączenie z dużym opóźnieniem (na podstawie średniego czasu opóźnienia) i i ponownie próbuje nawiązać połączenie. Protokół osiąga to poprzez analityczną ocenę dynamiki przeciążenia, która steruje połączeniami i eliminuje niski RTT [10].

Przesyłanie danych przez sieć jest kosztowne w przypadku, gdy należy zoptymalizować wydajność pod względem zużycia pamięci. Architektura Axis pozwala na wysłanie wielu buforów danych za pośrednictwem pojedynczego wywołania systemowego, co spowalnia przepustowość łącza. Natomiast gSOAP wysyła dane podczas ich przetwarzania, bezpośrednio z użyciem gniazd. W przypadku gSOAP wykorzystanie tej techniki i tryb wykorzystania protokołu TCP jest bardziej wydajny zarówno pod względem przesyłania danych, jak i zużycia pamięci (tabela 6).

Tabela 6: Porównanie zużycia RAM protokołem TCP+HTTP dla danych różnych rozmiarów.

RAM			
Fragmentacja danych	**gSOAP (MB/1000 pak)**	**Axis (MB/1000 pak)**	**Różnica (%)**
Tak	7.6-50	10-50	~4.16
Nie	7.6-50	25-70	~36.84

6. Wnioski

W pracy przeanalizowano wydajność przesyłania danych za pomocą protokołu SOAP i porównano zużycie pamięci przez oba frameworki. Wyniki badań mogą pomóc w projektowaniu i opracowaniu nowych zestawów narzędzi SOAP oraz mogą stanowić wskazówki dla użytkowników co do wyboru frameworkam, czy sposobu jego dopasowania do szczegółowych wymagań aplikacji.

Uzyskane wyniki pomiarów pozwalają stwierdzić, że w przypadku wytwarzania aplikacji, które muszą przetwarzać duże zbiory danych liczbowych, rozsądniej będzie korzystać z gSOAP. Prędkość przekształcania danych typu string jest szybsza w gSOAP. Jeżeli architektura serwera jest oparta na paradygmacie programowania OOP i transmisji danych w postaci obiektów - gSOAP ponownie jest wydajniejszym rozwiązaniem. Gdy głównym założeniem aplikacji jest wymiana plików multimedialnych, gSOAP także ma przewagę pod względem zużycia pamięci. Podsumowując, wybierając narzędzie do dokonania wydajnej transmisji danych gSOAP jest lepszym wyborem.

Axis jest prostszym narzędziem i pomimo tego, że jest mniej wydajnym rozwiązaniem, to można go polecić programistom, którzy nie posiadają doświadczenia w budowaniu web-serwisów.

Literatura

[1] Lista popularnych frameworków protokołu SOAP, https://en.wikipedia.org/wiki/List_of_web_service_frameworks, [4.09.2020].

[2] Szkody spowodowane opóźnieniem przesyłania usług internetowych, https://www.coxblue.com/7-ways-slow-internet-speeds-are-hurting-your-business/, [6.09.2020].

[3] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, R. Govindan, Reducing Web Latency: The Virtue of Gentle Aggression, In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (2013) 159-170.

[4] AXIOM, https://ws.apache.org/axiom/, [10.10.2020].

[5] Model obiektowy AXIOM, https://www.ibm.com/developerworks/ru/library/ws-java2/index.html, [25.10.2020].

[6] Dokumentacja Axis, https://axis.apache.org/axis2/java/core/apidocs/org/apache/axis2/jaxws/util/WSDL4JWrapper.html, [25.10.2020].

[7] M. R. Head, Benchmarking XML processors for applications in grid web services, In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (2006) 121-es.

[8] J. Kangasharju, S. Tarkoma, K. Raatikainen, Comparing SOAP Performance for Various Encodings, Protocols, and Connections, In IFIP International Conference on Personal Wireless Communications, Springer, Berlin, Heidelberg, (2003) 397-406.

[9] Serializacja i deserializacja, https://isocpp.org/wiki/faq/serialization, [1.10.2020].

[10] Opisy różnych realizacji protokołu TCP/IP, http://book.itep.ru/4/44/tcp.htm#8, [10.10.2020].