JCSI 1 (2016) 14-19
Journal of Computer Sciences Institute

Wysłane: 2016-09-15
Przyjęte: 2016-09-26

Porównanie wydajności wybranych technologii tworzenia usług sieciowych w aspekcie zastosowań w aplikacjach internetowych

Artur Gdula*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł przedstawia analizę porównawczą wybranych technologii tworzenia usług sieciowych. Analiza polega na porównaniu usług wykorzystujący protokół SOAP z usługami typu RESTful oraz szablonów wspomagających tworzenie usług w języku Java: Apache CXF, Spring WS, Jersey. W pierwszej kolejności analiza bazuje na metrykach statycznych porównując przykładowe aplikacje. W drugim etapie analiza skupia się na kwestii wydajności i porównuje wybrane technologie pod kątem czasu odpowiedzi usługi.

Słowa kluczowe: usługi sieciowe; wydajność, metryki

*Autor do korespondencji.

Adres E-mail: artur.gdula@pollub.edu.pl

Performance comparison of selected web service development technology in web applications

Artur Gdula*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents a comparative analysis of selected web service development technology. The analysis consists on a comparison of services using SOAP with RESTful web service and frameworks support the creation of web service in Java: Apache CXF, Spring WS, Jersey. In the first the analysis is based on a static metrics, compares the sample applications. In the second stage, the analysis raise the issue of performance, and compares the selected technologies in terms response time service.

Keywords: web services, performance; metrics

*Corresponding author.

E-mail address: artur.gdula@pollub.edu.pl

1. Wstęp

Usługi Web Service to współcześnie termin pojawiający się bardzo często. Jest to spowodowane postępem w technologiach informatycznych oraz ciągle rosnącymi wymaganiami biznesowymi. Usługi sieciowe mogą być rozwijane w oparciu o dwa style SOAP i REST [1]. Ten artykuł skupia się na zestawieniu szablonów aplikacji wspomagających tworzenie usług sieciowych wykorzystujących protokół SOAP i styl architektury oprogramowania REST. Ponadto zawiera zestawienie cech, wad i zalet porównywanych szablonów. Lista szablonów dostępnych w internecie, wspierających tworzenie usług sieciowych nie jest zbyt obszerna, najważniejszymi pozycjami są: Apache Axis2, Apache CXF, Java Web Services Development Pack / GlassFish, Jersey, Spring WS, Web Services Interoperability Technology. Analizie poddano trzy szablony Apache CXF, Spring WS i Jersey. Na wybór szablonów miało wpływ kilka istotnych kwestii. Apache CXF umożliwia proste i intuicyjne tworzenie usług, wspiera większość standardów specyficznych dla usług sieciowych, coraz częściej zastępuje poprzedniego lidera Apache Axis2. Spring bardzo często jest używany we wszelkiego rodzaju aplikacjach, ze względu na wiele modułów dostarczających rozwiązania dla różnych zagadnień technicznych. Szablon Jersey został wybrany, ze względu na swoją popularność, którą zyskał miedzy innymi dzięki swojej lekkości i prostocie. W procesie badawczym użyto metryk statycznych oraz dynamicznych, dlatego badania oraz analiza została podzielona na dwa etapy. W pierwszej kolejności analiza bazuje na metrykach statycznych, porównuje przykładowe aplikacje dostarczające usługi. Badania wydajnościowe zostały przeprowadzone z wykorzystaniem aplikacji klienckiej oraz programu SoapUI, umożliwiającego wykonywanie różnego rodzaju testów. Dodatkowym celem badań jest potwierdzenie lub obalenie prawdziwości tez: SOAP jest wolniejszą, ale bardziej ustandaryzowaną technologią w odróżnieniu od REST, JSON jest dużo lepszym rozwiązaniem we współczesnych aplikacjach od języka znaczników XML. Wybór stylu to ważna decyzja dla projektantów i programistów podczas wdrażania systemów [1]. Artykuł pomaga określić która z technologii oraz który z użytych szablonów sprawuje się najlepiej w określonych warunkach.

1.1. Style REST i SOAP

SOAP jest protokołem komunikacyjnym opartym o język znaczników, którego celem jest zastąpienie innych protokołów. SOAP określany jest następcą XML-RPC [2]. Początkowo protokół był wykorzystywany do wywołań RPC w sieci, jednak protokół był ciągle rozwijany i od wersji SOAP 1.1 zyskał własną strukturę komunikacyjną. „Interfejsy

usług SOAP są zwykle definiowane przez plik Web Service Description Language (WSDL), opisujący metody, parametry wejściowe i wyjściowe oraz schematy XML dla tych parametrów" [2]. SOAP wykorzystuje paradygmat koperty opartej na standardzie XML do przesyłania informacji i zdalnego wywoływania metod [3]. Sam protokół definiuje reguły przetwarzania przy pomocy węzłów oraz format wiadomości. Wiadomość SOAP – koperta może składać się z elementów: *header*, *body*, *fault.* "Specyfikacja SOAP definiuje węzeł SOAP, jako jednostkę, która implementuje reguły przetwarzania komunikatów SOAP" [4].

W przeciwieństwie do SOAP, REST nie jest protokołem komunikacyjnym. REST jest najczęściej definiowany jako styl architektoniczny budowy oprogramowania. Usługi sieciowe tworzone zgodnie z tym stylem są często nazywane RESTful Web Services. W ostatnich czasie gwałtowny rozwój aplikacji mobilnych przyczynił się do wzrostu popularności technologii REST. Technologia umożliwia tworzenie lekkich i prostych interfejsów API, ułatwiających integracje zarówno dużych systemów jak i małych aplikacji. Systemy REST do komunikacji pomiędzy serwerem a klientem najczęściej wykorzystują protokół HTTP. „Usługi REST zapewniają dostęp do zasobów poprzez czasowniki protokołu HTTP, wykorzystywane metody to: PUT,DELETE, POST, GET, HEAD" [5]. Klient który chcę uzyskać dostęp do zasobu musi wysłać odpowiednie żądanie z identyfikatorem URI. Identyfikatory pełnią kluczową role w architekturze REST, są to unikatowe adresy wskazujące na zasób, za ich pomocą odbywa się komunikacja pomiędzy dostawcą a odbiorcą usługi [4]. Wyżej wymienione operacje odpowiadają zaimplementowanym metodom na serwerze, które pozwalają na odczytanie, usunięcie oraz modyfikacje zbioru danych. Usługi REST mogą korzystać z różnych formatów wymiany danych ale najczęściej spotykanymi są pliki XML i JSON. W przeciwieństwie do SOAP przesyłana wiadomość zawiera mniej metadanych miedzy innymi dlatego usługi RESTful są określane jako prostsze i lżejsze. Mimo to usługi te pozwalają na operacje na obiektach zawierających dane nawet do 5 terabajtów [6]. W przypadku REST do dostarczenia opisu usługi można skorzystać z bazującym na XML-u standardzie WADL(Web Application Desciption Language). Nie zawsze w praktyce jest on stosowany, jednak bardzo ułatwia integracje aplikacji oraz podobnie jak WSDL pozwala na automatyczne tworzenie usług.

1.2. Apache CXF, Jersey, Spring WS

Apache CXF jest szablonem wspomagającym tworzenie usług sieciowych dostępnym na licencji open-source. Usługi sieci Web z użyciem Apache CXF mogą być implementowane z wykorzystaniem różnych protokołów, takich jak SOAP, XML, JSON, RESTful HTTP i obsługiwać różne protokoły komunikacyjne, takie jak HTTP lub JMS (Java Message Service) [7]. Apache CXF pozwala na integracje ze Springiem, w dużej mierze ułatwia to tworzenie usług wykorzystując pliki konfiguracyjne Springa.

Jersey to lekki i prosty szablon służącym do tworzenia usług typu RESTful. Podobnie jak w przypadku Apache CXF, Jersey może być zintegrowany ze Springiem, wspiera JAX-RS, pozwala na łatwe i szybkie tworzenie usług RESTful przy użyciu adnotacji. Ponadto szablon pozwala na negocjacje treści, dzięki temu komunikacja z usługą może odbywać się w dowolnie wybranym formacie.

Biblioteka Springa jest bardzo obszerna, ponadto jest rozszerzana przez poboczne projekty, obejmując: bezpieczeństwo, przepływy WWW, usługi sieciowe SOAP (usługi sieciowe REST są częścią głównej biblioteki), integracja w przedsiębiorstwie , przetwarzanie wsadowe, integracja z portalami społecznościowymi itd. Spring WS pozwala na stworzenie *contract-first* SOAP Web-Services, które w pierwszej kolejności wymagają stworzenia pliku WSDL lub XSD [8].

2. Metryki i metody

2.1. Metryki

Porównanie wybranych technologii zostało przeprowadzone w oparciu o metryki statyczne oraz metryki dynamiczne. Metryka oznacza dowolną wartość liczbową charakteryzującą oprogramowanie [9]. Według standardu IEE 1061-1998 metryka to funkcja odwzorowującą jednostkę oprogramowania w wartość liczbową, ta wyliczona wartość jest interpretowalna jako stopień spełnienia pewnej własności jakości jednostki oprogramowania [10]. Metryka oprogramowania jest ważnym czynnikiem pozwalającym odróżnić podobne produkty oferowane przez różnych dostawców [11]. Podstawowy podział metryk wyróżnia metryki statyczne i dynamiczne. W metrykach statycznych wyróżnia się metryki rozmiaru, obiektowe, złożoności, pakietów. Metryki statyczne odnoszą się do analizy kodu, są szczególnie przydatne w trakcie implementacji aplikacji. Do metryk dynamicznych zalicza się metryki pozwalające określić zachowanie już działającego produktu, pozwalają określić jak działa napisana usługa, czy spełnia wymagania wydajnościowe i oczekiwania klienta. Wszystkie metryki aby spełniały swoją role powinny charakteryzować się prostotą, niezależnością od języka, powinny odzwierciedlać przydatną i konkretną informacje oraz powinny dać się obliczyć przez komputer. Poniżej znajdują się definicje poszczególnych metryk.

2.1.1. Metryki statyczne

Liczba linii kodu- Najprostszą miara oprogramowania, należąca do metryk rozmiaru, jest metryka rozpoznawana pod skrótem LOC(*lines of code*) lub SLOC(*sources lines of code*). Metryka NCLOC jest liczbą linii kodu wykluczając puste linie i linie komentarzy. Na podstawie liczby linii kodu można wyliczyć inne metryki pochodne takie jak jakość, ilość dokumentacji. W części badawczej została użyta do porównania tej samej usługi napisanej przy użyciu różnych technologii, liczba linii kodu liczona będzie zarówno dla całych klas, interfejsów, metod oraz plików konfiguracyjnych.

Liczba klas i interfejsów - Metryka statyczna pozwalająca na określenie liczby klas i interfejsów w aplikacji. Metrykę

wykorzystano do zbadania liczby klas i interfejsów trzech aplikacji spełniających tą samą funkcjonalność.

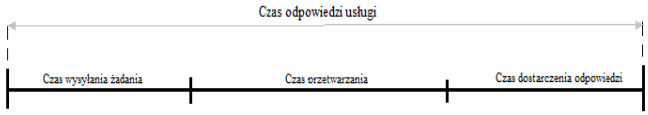
Liczba metod - Prosta metryka pozwalająca określić ilość metod w klasie, interfejsie czy też całej aplikacji. Metryka liczby metod podobnie jak metryka liczby klas i interfejsów została wykorzystana do porównania aplikacji spełniających tą samą funkcjonalność.

Liczba niezbędnych bibliotek - Metryka porównuje liczbę niezbędnych bibliotek potrzebnych do wystawienia usługi w porównywanych szablonach aplikacji. Liczba wymaganych bibliotek w każdym szablonie może być różna, oddziałuje to na inne metryki.

Rozmiar aplikacji - Porównanie aplikacji pod względem rozmiaru na dysku, ilości plików i folderów. W części badawczej zestawiono porównania kodu źródłowego oraz otrzymanego artefaktu w postaci pliku war. Na rozmiar aplikacji może wpływać ilość wykorzystywanych bibliotek, ilość napisanego kodu, ilość folderów i plików. Ta miara została wyrażona w KB.

2.1.2. Metryki dynamiczne

Czas odpowiedzi usługi - *„Service Response Time (SRT)* jest to czas odpowiedzi usługi jaki upłynął pomiędzy wysłaniem żądania a dostarczeniem usługi" [12]. Wyniki badań zostały zaprezentowane w milisekundach(ms). W związku z tym w celu dokładniejszego zbadania problemu czas odpowiedzi podzielono na trzy etapy. Badanie czasu odpowiedzi serwera przeprowadzono przy użyciu aplikacji klienckiej, testów funkcjonalnych, testów obciążeniowych oraz skryptów w narzędziu SoapUI. Rysunek 1 poniżej obrazuje badany czas odpowiedzi z podziałem na czas wysyłania żądania, czas przetwarzania żądania oraz czas dostarczenia odpowiedzi.



Rys 1. Czas odpowiedzi usługi

Obciążenie serwera - metryka pozwalająca określić obciążenie komputera np. zajętość procesora, zajętość pamięci operacyjnej. W tym celu wykorzystano zaawansowane funkcje testów obciążeniowych w programie SoapUI oraz księgarnie internetową.

2.2. Metody pomiaru

Badania wydajnościowe zostały przeprowadzone w lokalnej sieci WIFI na dwóch komputerach (dostawca i konsument usług). W celu otrzymania jak najdokładniejszych wyników, badania przeprowadzono na komputerach przy jak najmniejszej aktywności pamięci i procesora, wszystkie zbędne procesy i zadania systemu zostały wyłączone. Ponadto sieć została udostępniona jedynie dla dwóch komputerów, a czas obydwu komputerów został zsynchronizowany. Aplikacje dostarczające usługi uruchamiano kolejno na „czystym" kontenerze aplikacji webowych Tomcat 7.0.68. Do badania metryk dynamicznych z wykorzystaniem aplikacji klienckiej użyto metod *nanotime, currentTimeMillis* z pakietu JDK oraz biblioteki Apache log4j do logowania czasu w plikach tekstowych. Metoda *currentTimeMillis()* zwraca aktualny czas w milisekundach. Dodatkowo testy funkcjonalne i wydajnościowe zostały przeprowadzone przy użyciu narzędzia SoapUI 5.2.1.

3. Badania

Badania zostały przeprowadzone na trzech aplikacjach działających po stronie serwera, opartych na szablonach Apache CXF, Jersey, Spring WS. Aplikacje dostarczają usługi SOAP i REST. Początkowo aplikacje dostarczały jedynie usługi REST, natomiast po zaimplementowaniu usług SOAP aplikacje dostarczały zarówno usługi REST jak i SOAP. Klientami a zarazem narzędziami pomiarowymi są: aplikacja księgarni oraz program SoapUI. W pierwszej kolejności porównaniu poddane zostały aplikacje dostarczające usługi oferujące tą samą funkcjonalność w aspekcie metryk statycznych. Tabela 1 przedstawia rezultat porównania trzech aplikacji pod względem wykorzystania zasobów komputera. W przypadku pierwszych metryk odnoszących się do kodu źródłowego(liczba linii kodu, liczba metod itd.), najlepiej wypadł szablon Jersey, natomiast najbardziej rzuca się w oczy słaby wynik aplikacji Spring WS. Początkowo wydawać by się mogło, że to Apache CXF ze względu na swoją złożoną strukturę, wielkość bibliotek oraz przeznaczenie głównie dla usług sieciowych w tych metrykach będzie sprawował się najsłabiej. Mimo że, różnica pomiędzy projektami Apache CXF i Spring WS jest dość spora, zaimplementowanie dodatkowej usługi SOAP w każdym z projektów wymaga dodania podobnej ilości plików i dodanych linii kodu. Wartym zauważenia jest fakt, że aplikacja zaimplementowana z użyciem Spring WS potrzebuje najmniej bibliotek oraz po zbudowaniu i umieszczeniu w serwerze Tomcat zajmuje najmniej miejsca. Spowodowane jest to tym, że aplikacje oparte o Apache CXF i Jersey poza własnymi bibliotekami, korzystają dodatkowo z modułów szablonu Spring takich jak Spring Web, Spring Context, natomiast aplikacja oparta o Spring WS korzysta głównie z modułów Spring. Ponadto moduły w Springu są podzielone na wiele komponentów oraz zostały zaimplementowane w ten sposób aby móc działać samodzielnie. Natomiast Apache CXF posiada mniej modułów od Spring WS, są one specyficzne dla usług sieciowych, obszerniejsze, bardziej złożone oraz wspierają więcej standardów usług sieciowych.

Tabela 1. Porównanie stworzonych aplikacji z wykorzystaniem wybranych szablonów.

	Apache CXF		**Spring WS**		**Jersey**
Rodzaj dostarczanych usług:	REST	REST/SOAP	REST	REST/SOAP	REST
Liczba linii kodu	893	968	1282	1367	865
Liczba klas	15	17	22	24	15
Liczba interfejsów	5	6	5	6	5
Liczba metod	96	99	112	115	96
Rozmiar kodu źródłowego w kB	76	84	108	116	72
Ilość plików	23	25	33	36	25
Ilość plików konfiguracyjnych(xml, properties)	6	6	7	8	7
Ilość folderów	20	20	19	20	18
Liczba niezbędnych bibliotek(po skompilowaniu)	63	63	42	45	72
Rozmiar niezbędnych bibliotek(po skompilowaniu) w kB	26214	26214	20992	21504	25804
Rozmiar spakowanego programu(war) w kB	22732	23450	18739	19148	22835
Rozmiar aplikacji w tomcacie(folder w webapps) w kB	25498	26316	18739	19149	22835

Po porównaniu kodu aplikacji przeprowadzono testy wydajnościowe, wyniki testów znajdują się w tabelach 2-5. Badania przeprowadzono z użyciem narzędzia Soap UI oraz aplikacją kliencką. Narzędzia nie zawsze odwzorowują prawdziwego klienta dlatego też dla celów badań wydajnościowych zaimplementowano księgarnie internetową, konsumenta usług sieciowych. Jak można zauważyć w wynikach badań czasy odpowiedzi dla Soap UI różnią się od czasów odpowiedzi klienta-księgarni internetowej. Na podstawie tej obserwacji można dojść do wniosku, że nie warto polegać tylko i wyłącznie na różnego rodzaju narzędziach.

Tabela 2. Czas odpowiedzi usługi *getAllBooks* zmierzony aplikacją kliencką.

	Apache CXF	
Rodzaj dostarczanych usług:	REST	SOAP
Czas wysyłania żądania w ms	37	242
Czas przetwarzania w ms	36	34
Czas dostarczania odpowiedzi w ms	3548	362
Kompletny czas odpowiedzi usługi w ms	3621	3901
Kompletny czas odpowiedzi usługi w ms – wywołanie usługi 20 razy pod rząd	47153	73087

Tabela 3. Czas odpowiedzi usługi *getAllBooks* zmierzony narzędziem SoapUI.

	Apache CXF		**Spring WS**		**Jersey**
Rodzaj dostarczanych usług:	REST	SOAP	REST	SOAP	REST
Czas wysyłania żądania w ms	15	75	15	28	16
Czas przetwarzania w ms	48	29	59	38	38
Czas dostarczenia odpowiedzi w ms	1176	2358	1177	2830	1144
Kompletny czas odpowiedzi usługi w ms	1253	2470	1253	2899	1201
Kompletny czas odpowiedzi usługi w ms – wywołanie usługi 20 razy pod rząd	26866	48482	25506	56819	25468

Tabele 2 i 3 świadczą o tym że REST w każdym z badań przewyższają usługi SOAP. W przypadku jednokrotnego wywołania usługi REST najlepiej sprawuję się aplikacja oparta o szablon Jersey, niewiele mniejszy czas odpowiedzi mają pozostałe aplikacje. Ciekawa sytuacja występuje podczas wywołania usługi dwudziestokrotnie, aplikacja oparta o szablon Apache CXF spowalnia i znajduję się na

ostatnim miejscu. Analizując dane z tabeli 3 można stwierdzić, że aplikacja oparta o szablon Apache CXF doskonale radzi sobie z usługami SOAP, jednak jeśli chodzi o usługi REST im większy rozmiar danych, większe obciążenie czas odpowiedzi usługi wzrasta. Warto również wspomnieć o aplikacji wykorzystującej Spring WS, która nie ustępuje znacząco aplikacji wykorzystującej Apache CXF, w przypadku usług SOAP oraz aplikacji wykorzystującej szablon Jersey w przypadku usług REST. Na podstawie danych liczbowych zawartych w tabelach 2 i 3 można zauważyć, że bez względu na rodzaj usługi najwięcej czasu pochłania zwrócenie danych do klienta. Tabela 4 zawiera danę świadczące o obciążeniu serwera w trakcie korzystania z usług dostarczanych przez trzy aplikacje. Obciążenie serwera mierzono z wykorzystaniem programu Soap UI. W wyniku przeprowadzonych badań okazało się, że aplikacja oparta o szablon Apache CXF wykorzystuje w największym stopniu procesor i pamięć operacyjną zarówno w przypadku jednokrotnego wywołania usługi jak i przy pozostałych testach. Pozostałe aplikacje sprawują się lepiej, ale jest to mała różnica. Na obciążenie serwera nie wpływa również to czy wywołano usługę REST czy SOAP.

Tabela 4. Obciążenie serwera w trakcie wywoływania usługi *getAllBooks.*

	Apache CXF		**Spring WS**		**Jersey**
Rodzaj dostarczanych usług:	REST	SOAP	REST	SOAP	REST
Wykorzystanie procesora w procentach podczas wywoływania pojedynczej usługi	6	6	4	4	4
Wykorzystanie pamięci operacyjnej w procentach podczas wywoływania pojedynczej usługi	34	34	33	33	32
Wykorzystanie procesora w procentach podczas wywoływania 20 razy pod rząd tej samej usługi	7	8	5	6	5
Wykorzystanie pamięci w procentach podczas wywoływania 20 razy pod rząd tej samej usługi	35	36	34	34	33
Wykorzystanie procesora w procentach podczas testów obciążeniowych SoapUIw procentach	11	11	8	9	8
Wykorzystanie pamięci w procentach podczas testów obciążeniowych SoapUI	36	37	36	36	34

Analizując dane z tabeli 5 można dojść do wniosku, że warto dobrze dobrać format danych w używanych usługach. Na podstawie danych można wnioskować, że format JSON sprawuję się dużo lepiej od formatu XML. Natomiast nie widać znaczącej różnicy w czasach odpowiedzi w zależności od użytego szablonu.

Tabela 5. Wpływ formatu danych na przykładzie usługi *getAllBooks.*

	Apache CXF		**Spring WS**		**Jersey**	
Format danych:	XML	JSON	XML	JSON	XML	JSON
Minimalny czas odpowiedzi w ms	1020	827	960	931	972	835
Maksymalny czas odpowiedzi w ms	1196	864	1334	797	1099	858
Średni czas odpowiedzi w ms	1080	847	1101	855	1035	843

4. Wnioski

W analizie opisano czego może spodziewać się każdy programista wybierając konkretną technologie. Metryki proste wskazują programiście czego może się spodziewać podczas implementacji, natomiast metryki dynamiczne wskazują programiście jak dana aplikacja zachowa się w trakcie działania. W zależności od przeznaczenia tworzonej aplikacji wskazany może być wybór konkretnej technologii. Dla złożonych aplikacji biznesowych, wymagających dużego stopnia bezpieczeństwa, mogących w przyszłości rozrosnąć się o nowe funkcjonalności, zaleca używać się standardu SOAP. REST charakteryzuje się prostotą i sprawuje się dużo szybciej więc użycie go zaleca się w sytuacjach gdy priorytet ma szybkość działania.

Szablonem najlepiej sprawującym się w przypadku usług REST okazał się Jersey, jest lekkim, szybkim i prostym w użyciu szablonem. Z pewnością najlepiej sprawdził by się w prostych aplikacjach internetowych. W przypadku usług SOAP największe wsparcie dla innych technologii ma

szablon Apache CXF, ponadto okazał się najlepszy pod względem czasu odpowiedzi, jednak skutkowało to większym obciążeniem serwera. Dla programisty duże znaczenie może mieć technika tworzenia usług, szablon Spring WS wspiera jedynie technikę *conract-first*. Pomiar metryk dynamiczny wskazał że format JSON jest dużo wydajniejszy od formatu XML.

Badania metryk kodu wykazały, że aplikacja oparta Spring WS jest największa pod względem liczby metod i linii kodu. Jednak po zbudowaniu aplikacji liczba jej bibliotek i rozmiar jest najmniejszy spośród porównywanych aplikacji.

Nie jest zabronione stosowanie protokołu SOAP dla prostych aplikacji, a usług REST dla złożonych aplikacji, zależy to od przeznaczenia aplikacji, zespołu programistycznego itp. Wyników badań i wniosków nie należy traktować jako wytycznych podczas wyboru technologii, mogą one jedynie służyć jako wskazówki. Stworzone aplikacje oferowały jedynie proste funkcjonalności, dalszy rozwój aplikacji można stosować w celu rozbudowy o nowe funkcjonalności, moduły oraz przeprowadzenia dodatkowych badań.

Literatura

[1] K. Smita, S. Kumar Rath. Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration. Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on. IEEE, 2015.

[2] M. Lanthaler, C. Gütl. Towards a RESTful service ecosystem. Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on. IEEE, 2010.

[3] G. Mulligan, D. Gračanin. A comparison of SOAP and REST implementations of a service based interaction independence middleware framework. Simulation Conference (WSC), Proceedings of the 2009 Winter. IEEE, 2009.

[4] Z. Fryźlewicz, A.Salamon, Podstawy architektury i technologii usług XML sieci WEB, Warszawa : Wydawnictwo Naukowe PWN : Mikom, 2008

[5] A. Rodriguez. Restful web services: The basics. IBM developerWorks(2008).

[6] B. P. Upadhyaya, Rest client pattern. Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on. IEEE, 2014.

[7] N. Balani, R. Hathi. Apache CXF Web Service Development: Develop And Deploy SOAP And Restful Web Services. Birmingham: Packt Publishing, 2009. eBook Collection (EBSCOhost). Web. 6 May 2016.

[8] H. Sattari, S. Kunjumohamed. Spring Web Services 2 Cookbook : Over 60 Recipes Providing Comprehensive Coverage Of Parctical Real-Life Implementations Of Spring-WS. Birmingham: Packt Publishing, 2012. eBook Academic Collection (EBSCOhost). Web. 29 July 2016.

[9] L.Westfall, 12 Steps to Useful Software Metrics. Proceedings of the Seventeenth Annual Pacific Northwest Software Quality Conference. Vol. 57. 2005

[10] Software & Systems Engineering Standards Committee. IEEE Std 1061-1998—IEEE standard for a software quality metrics methodology. IEEE Computer Society, Tech. Rep (1998).

[11] N. Thio, S. Karunasekera, Automatic measurement of a qos metric for web service recommendation. Software Engineering Conference, 2005. Proceedings. 2005 Australian. IEEE, 2005.

[12] J.Sun Her ; S. Won Choi ; S. Hun Oh ; S. Dong Kim, A framework for measuring performance in service-oriented architecture. Next Generation Web Services Practices, 2007. NWeSP 2007. Third International Conference on. IEEE, 2007.