ZESZYTY NAUKOWE UNIWERSYTETU SZCZECIŃSKIEGO

NR 740 STUDIA INFORMATICA NR 31 2012

PIOTR CZAPIEWSKI
Zachodniopomorski Uniwersytet Technologiczny

STANDARDY USŁUG INTERNETOWYCH A INTEGRACJA ZE ŚRODOWISKAMI OBLICZEŃ NAUKOWYCH I INŻYNIERSKICH

Streszczenie

W pracy przedstawiono dyskusję możliwości integracji usług internetowych (Web Services) ze środowiskami obliczeń naukowych i inżynierskich, takimi jak MATLAB, GNU Octave czy SciLab. Ważniejsze standardy usług sieciowych (SOAP, REST, Hessian) poddano analizie pod kątem ich zastosowania w obliczeniach naukowych. Przedstawiono cele tego rodzaju integracji, przykładowe scenariusze wykorzystania oraz uwarunkowania techniczne.

Wprowadzenie

Najczęściej spotykany sposób wykorzystania środowisk obliczeń naukowych i inżynierskich, takich jak MATLAB, GNU Octave czy Scilab[1], zakłada działanie w trybie wsadowym. Do skryptów napisanych w jednym z języków czwartej generacji dostarcza się dane przygotowane uprzednio i zapisane

[1] Zob. *http://www.mathworks.com*, *http://www.gnu.org/software/octave, http://www.scilab.org*, dostęp 10.04.2012.

w postaci plików, ewentualnie składowane w relacyjnej bazie danych. Często tworzone są rozwiązania zamknięte, wykorzystujące wyłącznie funkcjonalność dostępną w ramach danego środowiska. Ta separacja od zewnętrznego świata i rozwiązań współczesnej inżynierii oprogramowania prowadzi do kilku patologii:

– jeśli określonej funkcjonalności brakuje w danym środowisku, często tworzona jest ona od podstaw, zamiast wykorzystania dostępnych gotowych rozwiązań;
– w przypadku konieczności aktualizacji danych, procedura ta wykonywana jest ręcznie (pobranie nowych danych, wstępne przetworzenie, uruchomienie od nowa analizy danych w sposób wsadowy);
– jakość tworzonych rozwiązań z punktu widzenia inżynierii oprogramowania jest niska: brak wykorzystania przyjętych wzorców projektowych, ścisłe wiązanie (*tight coupling*) i utrudnione ponowne użycie kodu;
– tworzone rozwiązania nie wykorzystują potencjału Internetu, ani w obszarze pozyskiwania i agregacji danych, ani przetwarzania chmurowego, ani interakcji w ramach serwisów społecznościowych.

W rozwiązaniu części z powyższych problemów pomóc może integracja środowisk obliczeniowych z usługami internetowymi różnego rodzaju. Integracja z systemami internetowymi, czy to prostymi aplikacjami webowymi, czy agentami, czy też rozwiązaniami typu *cloud computing*, może przynieść wiele korzyści. W dalszej części pracy przedstawione zostaną przykładowe scenariusze, w których tego rodzaju integracja może być wskazana. Omówione zostaną także najważniejsze standardy tworzenia usług sieciowych (*web services*), a także uwarunkowania techniczne istotne z punktu widzenia wykorzystania *web services* w praktyce obliczeń naukowych.

1. Standardy web services

Cechą wspólną wszystkich standardów *web services* jest ukierunkowanie na komunikację pomiędzy aplikacjami w środowiskach heterogenicznych i rozproszonych[2]. Standardy te oferują, choć w różnym stopniu, ujednolicenie

[2] G. Alonso, F. Casati, H. Kuno, V. Machiraju, *Web Services: Concepts, Architectures, Applications*, Springer 2004.

komunikacji bez względu na stosowane w aplikacjach technologie (języki programowania, mechanizmy składowania danych, itd.). Najważniejsze różnice pomiędzy nimi tkwią w takich aspektach, jak dopuszczalne formaty danych, zakres wsparcia ze strony technologii i języków programowania, wsparcie programisty przez narzędzia pomocnicze, wydajność czy narzuty związane z przesyłaniem metadanych. Poniżej przedstawiono analizę najważniejszych spośród dostępnych standardów.

1.1. SOAP

SOAP jest protokołem komunikacyjnym, który wyewoluował ze standardu XML-RPC i pozostaje silnie związany z językiem XML[3] . Zaprojektowany został w celu wymiany ustrukturalizowanej informacji w środowiskach rozproszonych. Bazując na XML i technologiach powiązanych, definiuje rozszerzalny szkielet komunikatów. Bazowa, zewnętrzna struktura komunikatu pozostaje stała (koperta, nagłówek, zawartość), zaś struktura zawartości definiowana jest każdorazowo dla konkretnej aplikacji. Chociaż pierwotna specyfikacja dopuszczała stosowanie różnych mechanizmów do określenia struktury komunikatu, to najczęściej stosowanym i obecnie jedynym zalecanym rozwiązaniem jest użycie języka XML Schema[4]. W ten sposób uzyskuje się neutralność technologiczną i niezależność typów danych od platformy, a także możliwość pełnej walidacji przesyłanych komunikatów. Do bibliotek i warstw wiążących SOAP z konkretnym środowiskiem programowym należy translacja z typów precyzyjnie zdefiniowanych w XML Schema na typy danego języka programowania.

Jedną z istotnych cech usług opartych na protokole SOAP jest wykorzystanie standardu WSDL (*Web Service Description Language*) do opisania

[3] World Wide Web Consortium: *SOAP Version 1.2 Part 0: Primer (Second Edition),* W3C Recommendation, *http://www.w3.org/TR/soap12-part0/*, dostęp 10.04.2012; D. Winer, *XML--RPC Specification*, *http://xmlrpc.scripting.com/spec.html*, dostęp 10.04.2012; D. Box, *A Brief History of SOAP*, *http://www.xml.com/pub/a/ws/2001/04/04/soap.html*, dostęp 10.04.2012.

[4] World Wide Web Consortium: *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures,* W3C Recommendation, *http://www.w3.org/TR/xmlschema11–1/*, dostęp 10.04.2012; World Wide Web Consortium: *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes,* W3C Recommendation, *http://www.w3.org/TR/xmlschema11–2/*, dostęp 10.04.2012.

interfejsu konkretnej usługi[5]. Dokument WSDL zawiera wszelkie informacje niezbędne do wykorzystania danej usługi, w tym listę dostępnych operacji, ich parametry oraz stosowane typy danych. Rola WSDL zbliżona jest do IDL znanego ze standardu CORBA[6]. Wykorzystanie WSDL znacząco ułatwia tworzenie aplikacji klienckich oraz współdzielenie usług. Między innymi językowi WSDL usługi SOAP zawdzięczają najlepsze spośród omawianych standardów wsparcie ze strony narzędzi programistycznych, takich jak biblioteki, środowiska GUI, generatory kodu, narzędzia do testowania ręcznego i automatycznego.

Podstawowym paradygmatem towarzyszącym usługom SOAP jest RPC – zdalne wywoływanie procedur (*Remote Procedure Call*). Co prawda możliwe jest tworzenie usług, których celem jest wymiana fragmentów dokumentów, jednak naturalną dla tego standardu perspektywą jest zdalne wywołanie funkcjonalności udostępnionej na serwerze. Listing 1 przedstawia przykładowy komunikat SOAP, zawierający polecenie wykonania obliczeń.

Za wadę SOAP można uznać wymuszenie stosowania w komunikacji języka XML oraz związane z tym narzuty. Narzut związany z elementami XML oraz metadanymi jest szczególnie znaczący przy częstym przesyłaniu niewielkich ilości danych oraz przy przesyłaniu dużych, homogenicznych zbiorów danych.

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:sciw="http://sciws.chpd.org/">
   <soapenv:Header/>
   <soapenv:Body>
     <sciw:calculateRsi>
        <quote>EUR/USD</quote>
        <period>14</period>
        <numPoints>32</numPoints>
        <delay>0</delay>
     </sciw:calculateRsi>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 1. Przykładowy komunikat SOAP

[5] World Wide Web Consortium: *Web Services Description Language (WSDL) Version 2.0*, W3C Recommendation, *http://www.w3.org/TR/wsdl20-primer/*, dostęp 10.04.2012.

[6] Object Management Group: *Common Object Request Broker Architecture (CORBA) Specification*, *http://www.omg.org/spec/CORBA/Current/*, dostęp 10.04.2012.

1.2. REST

Termin REST (*REpresentational State Transfer*), wprowadzony przez Fieldinga[7], wbrew powszechnej opinii nie odnosi się wyłącznie do usług sieciowych. Jest to styl architektoniczny, którego sieć WWW z protokołem HTTP jest jedną z realizacji. W przypadku usług zgodnych z wytycznymi REST często mówi się o *RESTful Web Services*.

Porównanie kluczowych cech usług typu REST oraz SOAP znaleźć można m.in. w pracy Pautasso i in.[8] Wśród najważniejszych kwestii odróżniających REST od SOAP wymienić można: paradygmat ROA zamiast SOA/RPC, możliwość użycia różnych formatów danych, lepsze wykorzystanie cech protokołu HTTP, brak sformalizowanego opisu usługi.

W usługach REST mamy do czynienia z odmiennym paradygmatem niż w przypadku SOAP, nazywanym ROA (*Resource Oriented Architecture*)[9]. Punktem centralnym jest tutaj zasób i dostęp do niego za pomocą zdefiniowanych w protokole HTTP czterech metod: GET, POST, PUT i DELETE, przy czym semantyka metod jest tu ściśle określona. Metody te bezpośrednio mapowane są na typowe operacje, jakich dokonuje się na elementach zbiorów danych: pobranie, zapisanie nowego zasobu, zapisanie zmian w istniejącym zasobie, usunięcie. Co za tym idzie, od strony koncepcyjnej bardziej zasadne jest stosowanie REST w przypadku usług oferujących dostęp do określonych danych, niż usług skupionych na udostępnianiu funkcjonalności (np. złożonej logiki biznesowej lub metod obliczeniowych).

Sama koncepcja RESTful Web Services nie wymusza formatu wymiany danych. W praktyce najczęściej spotyka się przesyłanie komunikatów w formacie XML lub JSON. Dodatkowo dzięki wykorzystaniu mechanizmu negocjacji

---

[7] R.T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, University of California, 2000, *http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf*, dostęp 10.04.2012.

[8] C. Pautasso, O. Zimmermann, F. Leymann, *Restful web services vs. "big" web services: making the right architectural decision*, Proceedings of the 17th international conference on World Wide Web, s. 805–814.

[9] D. Duggan, *Enterprise Software Architecture and Design: Entities, Services, and Resources*, Wiley 2012; D. Guinard, V. Trifa, F. Mattern, E. Wilde, *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices*, w: *Architecting the Internet of Things*, red. D. Uckelmann, M. Harrison, F. Michahelles, Springer 2011.

zawartości za pomocą nagłówków HTTP możliwe jest stosowanie w tej samej usłudze różnych formatów danych.

Komunikaty przesyłane w usługach typu REST są zwykle lżejsze niż w SOAP. Wynika to po części z faktu braku narzutów na metadane oraz maksymalnego wykorzystania cech protokołu HTTP (np. mapowanie adresu URL oraz metod HTTP na określoną semantykę, zamiast zawarcia jej w treści komunikatu), a po części z możliwości użycia lżejszego formatu wymiany danych niż XML (np. JSON, a w skrajnym przypadku nawet formatu binarnego, np. zakodowanego z użyciem Base64).

W przeciwieństwie do SOAP, brak tu jest sformalizowanego opisu usługi. Co prawda możliwe jest użycie standardu WADL[10], jednak w praktyce rzadko jest to spotykane, a możliwości są tu znacznie mniejsze niż w przypadku SOAP. Z tego powodu, a także z powodu braku ustalonego formatu komunikatu oraz specyficznego podejścia do mapowania adresów na zasoby, wsparcie ze strony narzędzi jest tu znacznie uboższe niż w przypadku SOAP. O ile zaimplementowanie obsługi komunikatów od podstaw byłoby w przypadku REST znacznie łatwiejsze, o tyle w przypadku użycia gotowych narzędzi SOAP pozwala na znacznie szybsze i bardziej niezawodne tworzenie kodu klienckiego (np. dzięki walidacji komunikatów, kontroli typów argumentów, automatycznemu generowaniu kodu).

Usługi REST z założenia są bezstanowe, co oznacza, że serwer usługi nie przechowuje informacji o stanie sesji. W konsekwencji możliwe jest lepsze wykorzystanie cech protokołu HTTP oraz duża skalowalność. Oznacza to także, że stan aplikacji musi być przechowywany w obrębie zasobów lub po stronie klienta.

Na rysunku 1 przedstawiono przykładowe API zgodnej z koncepcją REST usługi, pozwalającej na zdalny dostęp do urządzeń pomiarowych. Zdefiniowano w niej następujące zasoby: *Urządzenie*, *Konfiguracja* oraz *Pomiar*. Przy konstrukcji żądania do tego typu usługi istotne są trzy elementy, razem stanowiące o wykonywanej operacji: adres URL zasobu, użyta metoda HTTP oraz ciało żądania HTTP. Przykładowo, celem pobrania wyników wszystkich pomiarów wykonanych przez urządzenie o identyfikatorze 17, należy przesłać

[10] World Wide Web Consortium: *Web Application Description Language,* W3C Member Submission, *http://www.w3.org/Submission/wadl/*, dostęp 10.04.2012.

żądanie GET pod adres: *http://serwer/urzadzenie/17/pomiar*. Z kolei zmiana konfiguracji urządzenia może nastąpić przez wysłanie żądania PUT pod adres: *http://serwer/urzadzenie/17/konfiguracja*, gdzie w ciele komunikatu HTTP umieszczono nowe dane konfiguracyjne w odpowiednim formacie.

/urzadzenie/	
GET	– pobierz listę urządzeń
POST	– dodaj nowe urządzenie
PUT	– nieużywane
DELETE	– nieużywane

/urzadzenie/ {id}	
GET	– pobierz informację o urządzeniu o danym id
POST	– nieużywane
PUT	– aktualizuj informacje o urządzeniu
DELETE	– usuń urządzenie

/urzadzenie/ {id}/pomiar	
GET	– pobierz wszystkie wyniki pomiarów dla danego urządzenia
POST	– zapisz nowy wynik pomiaru
PUT	– nieużywane
DELETE	– nieużywane

/urzadzenie/ {id}/pomiar/ {id-pomiaru}	
GET	– pobierz wynik pojedynczego pomiaru
POST	– nieużywane
PUT	– aktualizuj wynik pomiaru
DELETE	– usuń wynik pomiaru

/urzadzenie/ {id}/konfiguracja	
GET	– pobierz dane konfiguracyjne urządzenia
POST	– dodaj nowy parametr konfiguracyjny
PUT	– aktualizuj dane konfiguracyjne
DELETE	– nieużywane

/urzadzenie/ {id}/konfiguracja/parametr	
GET	– pobierz wartość pojedynczego parametru konfiguracyjnego
POST	– nieużywane
PUT	– aktualizuj wartość parametru
DELETE	– usuń pojedynczy parametr

Rys. 1. Przykład API dla usługi typu RESTful

1.3. Hessian

Zarówno w przypadku usług SOAP, jak i REST, możliwe jest przesyłanie danych binarnych, jednak nie jest to ani szybkie, ani wygodne w użyciu. Jeżeli usługa wymaga transferu znacznej ilości danych binarnych, bądź też krytyczna jest wydajność transferu, bardziej odpowiedni do jej realizacji może się okazać protokół Hessian[11]. Z założenia jest to protokół binarny, przeznaczony do tworzenia lekkich usług sieciowych, bez konieczności użycia ciężkich stosów

[11] Caucho Technology, Inc., *Hessian Binary Web Service Protocol*, *http://hessian.caucho.com*, dostęp 10.04.2012.

programowych. Dane binarne mogą być w nim przesyłane bezpośrednio, nie zaś jak w przypadku SOAP jako załącznik[12].

W standardzie określono sposób serializacji podstawowych typów danych oraz format polecenia zdalnego wywoływania metod w stylu RPC. Brak tu zewnętrznego opisu API usługi (odpowiednika WSDL z SOAP).

Największą wadą protokołu Hessian jest jego niewielkie rozpowszechnienie, a co za tym idzie – najsłabsze wsparcie ze strony narzędzi. Dostępne są stosowne biblioteki dla większości popularnych, współczesnych języków programowania (w tym dla języków Java, C++, C#, PHP, Ruby, Objective C), jednak wykorzystanie protokołu Hessian w innych językach czy środowiskach (np. Fortran) może być kłopotliwe i wymagać implementacji od podstaw.

2. Integracja usług sieciowych ze środowiskami obliczeń naukowych

2.1. Przykładowe scenariusze i cele integracji

Większość scenariuszy integracji usług sieciowych ze środowiskami obliczeń naukowych można przypisać do jednej z dwóch kategorii:

– wykorzystanie zasobów internetowych jako źródła danych,
– zdalne wywołanie procedur przetwarzania danych.

W pierwszym przypadku dane pobierane mogą być np. z istniejących, działających *online* systemów, w których zgromadzone są dane związane z normalnym, codziennym funkcjonowaniem aplikacji. Dotyczyć to może zarówno systemów biznesowych, jak i np. monitorujących czy zarządzających produkcją przemysłową. Pobierane okresowo dane mogą zostać poddane analizie w środowisku obliczeń naukowych, a bezpośrednia integracja z użyciem *web services* pozwoli na znaczne skrócenie czasu pomiędzy kolejnymi analizami lub też na ich automatyzację, a także pomoże w zachowaniu spójności i uniknięciu błędów podczas aktualizacji danych. Punktem centralnym jest w tym przypadku system informacyjny, na potrzeby którego prowadzone są analizy. Wykorzystanie środowiska typu MATLAB, zamiast implementacji logiki analitycznej bezpośrednio w oprogramowaniu użytkowym, może być wskazane ze względu na łatwiejsze i szybsze prototypowanie, aktualizację

[12] World Wide Web Consortium, *SOAP Messages with Attachments,* W3C Note, *http://www.w3.org/TR/SOAP-attachments/*, dostęp 10.04.2012.

i modyfikację kodu, a także na chęć wykorzystania analityków ze znacznym doświadczeniem naukowym, a niewielkim programistycznym.

Z inną sytuacją mamy do czynienia, gdy punktem ciężkości stają się badania naukowe, a oprogramowanie działające w chmurze przygotowane zostało specjalnie na potrzeby ciągłego zbierania, magazynowania i agregowania danych. Dzięki temu działające w trybie *online* oprogramowanie agenckie może nieprzerwanie gromadzić dane i poddawać je wstępnej obróbce. Tego rodzaju zadania nie można zrealizować bezpośrednio we wspomnianych środowiskach obliczeniowych, ze względu na ich interaktywny i niejako wsadowy charakter. Przykładem może być budowa rozproszonej sieci agentów monitorujących serwisy społecznościowe, doniesienia prasowe czy notowania finansowe, które następnie pobierane będą okresowo do analizy.

Druga z kategorii zakłada wykorzystanie zdalnych zasobów obliczeniowych, co może być zasadne w dwóch ogólnych przypadkach. Po pierwsze, w celu przyspieszenia obliczeń – usługa internetowa może być fasadą dużego systemu obliczeniowego, czy to klasycznego klastra, czy specjalizowanego akceleratora obliczeń (np. NVidia Tesla), czy wreszcie rozwiązania chmurowego (np. wykorzystującego jedną z dostępnych technologii realizujących model MapReduce). Po drugie, zasadne może być wykorzystanie zaimplementowanej już w istniejących systemach logiki analitycznej – podstawową korzyścią jest wówczas nie czas przetwarzania, lecz przyspieszenie wytwarzania oprogramowania analitycznego.

2.2. Wybór standardu usług sieciowych

Biorąc pod uwagę przedstawione wcześniej aspekty poszczególnych standardów usług sieciowych oraz typowe scenariusze integracji, można zaproponować następujące preferencje odnośnie do wyboru wdrażanego typu usług.

W typowych przypadkach współdzielenia danych, zwłaszcza przy przewadze jednokierunkowego pobierania ich z usługi, korzystne będzie stosowanie usług typu REST. Dla danych o złożonej strukturze wskazane będzie użycie języka XML, wraz ze schematem XML Schema (opcjonalnym w przypadku REST). Pozwoli to na uzyskanie względnie wysokiej wydajności i prostej struktury kodu oraz na łatwą implementację serwera w dowolnej technologii

hostingowej czy chmurowej, przy zachowaniu możliwości walidacji przesyłanych komunikatów.

W przypadku użycia usług sieciowych w celu współdzielenia logiki obliczeniowej, wskazane będzie użycie usług SOAP lub Hessian. Wybór zależeć będzie od wymagań dotyczących wydajności i ilości przesyłanych danych. Jeśli szacowany czas obliczeń znacznie przeważać będzie czas transferu, należy wybrać SOAP. W przypadku konieczności częstego transferu dużej ilości danych binarnych, lepszym wyborem będzie Hessian.

2.3. Możliwości integracji środowisk obliczeniowych

Oprócz powyższych wskazań, zależnych wyłącznie od charakteru postawionego zadania, należy także wziąć pod uwagę możliwości integracyjne poszczególnych środowisk obliczeń naukowych.

Środowisko MATLAB oferuje pod tym względem bardzo szerokie możliwości[13]. Obsługa usług SOAP wbudowana jest w samo środowisko, co czyni je najłatwiej dostępnymi. Bezpośrednia obsługa usług REST czy Hessian nie jest co prawda możliwa, jednak dostępne są trzy mechanizmy pozwalające na stosunkowo łatwe skorzystanie z tych usług:

- możliwość wywoływania kodu Java;
- możliwość wywoływania kodu .NET i obiektów COM;
- możliwość wywoływania kodu z odpowiednio przygotowanych, skompilowanych bibliotek C/C++.

Ze względu na ścisłą integrację z maszyną wirtualną Java (JVM), ten wariant w praktyce daje najszersze możliwości i potencjalnie najlepszą wydajność. Dowolny kod napisany w języku Java może być bezpośrednio wywoływany z linii poleceń czy ze skryptów programu MATLAB. Biorąc pod uwagę szeroką dostępność bibliotek ułatwiających korzystanie z usług (np. RestEasy, Jersey, Apache CXF czy Restlet[14] dla REST), stworzenie stosownego adaptera dla skryptów MATLAB jest zadaniem stosunkowo prostym.

[13] The MathWorks, Inc., *MATLAB Documentation. External Interfaces*, *http://www.mathworks.com/help/techdoc/matlab_external/bp_kqh7.html*, dostęp 10.04.2012.

[14] Zobacz *http://www.jboss.org/resteasy, http://jersey.java.net, http://cxf.apache.org, http://www.restlet.org*, dostęp 10.04.2012.

Darmowe środowisko GNU Octave, rozwijane na licencji *open source*, oferuje uboższe możliwości integracji niż MATLAB. Tutaj także jest jednak możliwe wywoływanie kodu Java, co umożliwia wykorzystanie dowolnego typu usług sieciowych[15]. Ponadto w prostszych przypadkach można pokusić się o bezpośrednią obsługę usług typu REST za pomocą funkcji do obsługi protokołu HTTP. Oprogramowanie Scilab, również na licencji *open source*, wypada pod względem możliwości integracji podobnie jak GNU Octave, również oferując możliwość wywołania kodu Java (za pomocą dodatkowego rozszerzenia[16]). W obu rozwiązaniach *open source* brak bezpośredniej obsługi SOAP oraz możliwości wywołania kodu .NET.

2.4. Architektura proponowanego rozwiązania

Proponowane rozwiązanie zakłada obustronną komunikację z zasobami internetowymi za pośrednictwem dwóch modułów:

- interfejsu zewnętrznych usług sieciowych – modułu mającego na celu umożliwienie wykorzystania zdalnych zasobów, źródeł danych i usług sieciowych;
- interfejsu udostępniania usług sieciowych – modułu mającego na celu udostępnienie na zewnątrz zasobu środowiska obliczeniowego w postaci usług sieciowych.

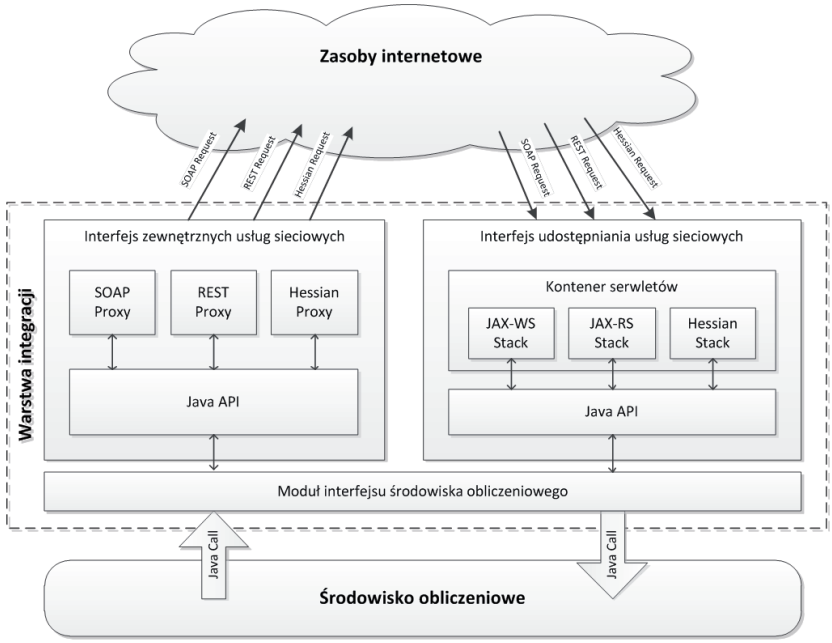
W przypadku obu modułów rodzaj wykorzystanych usług sieciowych zostanie ukryty przed środowiskiem obliczeniowym poprzez zastosowanie odpowiednich pośredników (*proxy*), których celem będzie komunikacja z użyciem bibliotek odpowiednich dla danego standardu usług (SOAP Proxy, REST Proxy, Hessian Proxy). Usługi udostępnione zostaną środowisku obliczeniowemu w postaci API w języku Java, do bezpośredniego wywołania.

Moduł udostępniający usługi sieciowe zawiera wbudowany lekki kontener serwletów oraz stosy programowe dla poszczególnych standardów usług sieciowych. W tym module przychodzące z zewnątrz żądania wywołania usług zostają tłumaczone na wywołania stosownych procedur i funkcji środowiska obliczeniowego.

---

[15] *GNU Octave Wiki: Java Package*, *http://www.octave.org/wiki/index.php?title=Java_package*, dostęp 10.04.2012.

[16] C. Denizet, *Java Interaction Mechanism in Scilab*, *http://atoms.scilab.org/toolboxes/JIMS*, dostęp 10.04.2012.



Rys. 2. Ogólna architektura proponowanego rozwiązania integracyjnego

Oba moduły komunikują się ze środowiskiem obliczeniowym, korzystając z modułu pośredniczącego. Adaptacja rozwiązania do różnych środowisk obliczeniowych wymagać powinna jedynie wymiany lub modyfikacji modułu pośredniczącego.

2.5. Ograniczenia proponowanego rozwiązania

Pewne ograniczenia czy wady rozwiązania polegającego na bezpośredniej integracji aplikacji czy zasobów internetowych ze środowiskiem obliczeń naukowych po części wynikają z inherentnych cech aplikacji i usług internetowych, a po części z odmiennej natury produkcyjnych środowisk programowych i środowisk obliczeń naukowych.

W przypadku wykorzystania zewnętrznych zasobów internetowych poważnym ograniczeniem jest fakt pracy synchronicznej wszystkich wymienionych standardów usług sieciowych. Wywoływana usługa, czy też w ogólności – zasób, do którego próbujemy uzyskać dostęp z poziomu skryptu środowiska

obliczeniowego, musi być w danym momencie dostępny i odpowiadać na żądania. W sytuacjach gdy takie odwołania zewnętrzne są częste i powtarzalne, może to prowadzić do niepotrzebnego wstrzymania procesów obliczeniowych w oczekiwaniu na zdalne zasoby. Do pewnego stopnia można temu zaradzić, wprowadzając np. mechanizmy cache'owania danych, nie rozwiązuje to jednak problemu dostępu do zdalnych zasobów obliczeniowych.

W sytuacji odwrotnej, kiedy to zewnętrzna aplikacja ma uzyskać dostęp do stworzonych w ramach środowiska obliczeniowego procedur za pomocą interfejsu usług sieciowych, problemem może się okazać niższa niż wymagana w warunkach produkcyjnych stabilność. Środowiska obliczeniowe przystosowane są raczej do pracy interaktywnej z niewielkim obciążeniem, z nastawieniem na łatwość prototypowania i implementacji, niż na stabilność funkcjonowania. Bezpośrednie wywoływanie procedur obliczeniowych z poziomu serwera aplikacji, zwłaszcza przy dużym obciążeniu, może doprowadzić do niestabilności środowiska obliczeniowego. Pewnym rozwiązaniem może być użycie równolegle kilku środowisk pracujących w ramach odseparowanych maszyn wirtualnych oraz częsta kontrola stabilności tychże maszyn, z równoważeniem obciążenia i możliwością automatycznego restartu.

Podsumowanie

Integracja środowisk przeznaczonych do obliczeń naukowych i inżynierskich z usługami sieciowymi może być przeprowadzona w celu eksploracji rozproszonych, internetowych źródeł danych, zdalnego wykorzystania specjalizowanych zasobów obliczeniowych czy też ponownego użycia logiki analitycznej zaimplementowanej w istniejących systemach.

W zależności od przypadku użycia, wskazany może być wybór jednego z kilku popularnych standardów *web services*. Dla typowego przypadku dzielenia funkcjonalności, zwłaszcza przy złożonych API analitycznych, zalecić można standard SOAP. W sytuacji gdy kluczowe staje się współdzielenie zasobów, nie zaś funkcji, bardziej zasadne będzie użycie prostszych usług typu REST. W zastosowaniach, gdzie konieczne jest częste przesyłanie dużej ilości danych, przy zachowaniu maksymalnej wydajności, wskazane będzie zastosowanie standardu Hessian.

Większość stosowanych obecnie środowisk obliczeniowych oferuje mechanizmy integracji z zewnętrznymi zasobami programowymi. Zarówno w przypadku środowiska MATLAB, jak i darmowych GNU Octave i Scilab, możliwe jest wywoływanie kodu napisanego w języku Java, co daje dostęp do wszelkich standardów usług sieciowych za pośrednictwem dodatkowych bibliotek bądź mechanizmów wbudowanych w JDK. Ponadto w środowisku MATLAB dostępny jest bezpośredni interfejs do usług typu SOAP oraz bibliotek stworzonych w językach .NET/C#, C/C++, Fortran.

Przedstawiona w artykule dyskusja oraz rozwiązanie architektoniczne stanowi wstęp do dalszych prac nad realizacją uniwersalnego modułu integrującego różne środowiska obliczeń naukowych i inżynierskich z serwerami aplikacji, usługami sieciowymi i internetowymi zasobami danych.

Literatura

1. Alonso G., Casati F., Kuno H., Machiraju V., *Web Services: Concepts, Architectures, Applications*. Springer 2004.
2. Box D., *A Brief History of SOAP*, *http://www.xml.com/pub/a/ws/2001/04/04/soap.html*, dostęp 10.04.2012.
3. Caucho Technology, Inc., *Hessian Binary Web Service Protocol*, *http://hessian.caucho.com*, dostęp 10.04.2012.
4. Denizet C., *Java Interaction Mechanism in Scilab*, *http://atoms.scilab.org/ toolboxes/JIMS*, dostęp 10.04.2012.
5. Duggan D., *Enterprise Software Architecture and Design: Entities, Services, and Resources*, Wiley 2012.
6. Fielding R.T., *Architectural Styles and the Design of Network-based Software Architectures*, University of California, 2000, *http://www.ics.uci.edu/~fielding/pubs /dissertation/fielding_dissertation.pdf*, dostęp 10.04.2012.
7. *GNU Octave Wiki: Java Package*, *http://www.octave.org/wiki/index.php?title =Java_package*, dostęp 10.04.2012.
8. Guinard D., Trifa V., Mattern F., Wilde E., *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices*, w: *Architecting the Internet of Things*, red. D. Uckelmann, M. Harrison, F. Michahelles, Springer 2011.
9. The MathWorks, Inc., *MATLAB Documentation. External Interfaces*, *http://www.mathworks.com/help/techdoc/matlab_external/bp_kqh7.html*, dostęp 10.04.2012.
10. Object Management Group, *Common Object Request Broker Architecture (CORBA) Specification*, *http://www.omg.org/spec/CORBA/Current/*, dostęp 10.04.2012.

11. Pautasso C., Zimmermann O., Leymann F., *Restful web services vs. "big" web services: making the right architectural decision*, Proceedings of the 17th international conference on World Wide Web, s. 805–814.
12. Winer D., *XML-RPC Specification*, *http://xmlrpc.scripting.com/spec.html*, dostęp 10.04.2012.
13. World Wide Web Consortium, *SOAP Messages with Attachments*, W3C Note, *http://www.w3.org/TR/SOAP-attachments/*, dostęp 10.04.2012.
14. World Wide Web Consortium, *SOAP Version 1.2 Part 0: Primer (Second Edition)*, W3C Recommendation, *http://www.w3.org/TR/soap12-part0/*, dostęp 10.04.2012.
15. World Wide Web Consortium, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, W3C Recommendation, *http://www.w3.org/TR/soap12-part1/*, dostęp 10.04.2012.
16. World Wide Web Consortium, *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*, W3C Recommendation, *http://www.w3.org/TR/soap12-part2/*, dostęp 10.04.2012.
17. World Wide Web Consortium, W3C *XML Schema Definition Language (XSD) 1.1 Part 1: Structures*, W3C Recommendation, *http://www.w3.org/TR/xmlschema11–1/*, dostęp 10.04.2012.
18. World Wide Web Consortium, W3C *XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*, W3C Recommendation, *http://www.w3.org/TR/xmlschema11–2/*, dostęp 10.04.2012.
19. World Wide Web Consortium, *Web Application Description Language*, W3C Member Submission, *http://www.w3.org/Submission/wadl/*, dostęp 10.04.2012.
20. World Wide Web Consortium, *Web Services Description Language (WSDL) Version 2.0*, W3C Recommendation, *http://www.w3.org/TR/wsdl20-primer/*, dostęp 10.04.2012.

STANDARDS OF WEB SERVICES AND THE INTEGRATION WITH ENVIRONMENTS FOR SCIENTIFIC AND ENGINEERING COMPUTATIONS

Summary

The paper presents the discussion of the possibility of integrating Web Services with environments for scientific and engineering computations, such as MATLAB, GNU Octave or SciLab. Major web services standards (SOAP, REST, Hessian) were analysed regarding the feasibility of their application in scientific computations. The goals of such an integration were explicated, as well as the exemplary usage scenarios and the technical requirements.

Translated by Piotr Czapiewski