AUTOMATYKA • 2009 • Tom 13 • Zeszyt 2

Lev Belava*

Koncepcja hybrydowej kompozycji usług w środowisku SOA

1. Wprowadzenie

Z obserwacji rosnącej złożoności współczesnych procesów przetwarzania informacji można wysnuć wniosek, iż tematyka kompozycji usług w ramach SOA (*Service Oriented Architecture*) w sposób naturalny pasuje do prób pozyskiwania metod wytwarzania oprogramowania korzystającego nie tylko z wyspecjalizowanych funkcji dostarczanych za pomocą usług sieciowych, ale również będącego dynamicznym z punktu widzenia swojej struktury wewnętrznej. Komponując różne usługi podstawowe w ramach tworzenia bardziej złożonych, system może oferować specjalistom zarówno odpowiedni dla problemu poziom abstrakcji (np. poziom całej kompozycji, jej części, usługi podstawowej, pojedynczego wywołania funkcji, etc.), jak i dużą elastyczność z punktu widzenia procesów tworzenia tych abstrakcji.

1.1. SOA

SOA, czyli architektura zorientowana na usługi, jest paradygmatem opisującym, jakie cechy powinien mieć system informatyczny oparty na usługach (niekoniecznie sieciowych). Przedstawia nowy pogląd na tworzenie systemów informatycznych. Odchodzi od podejść klasycznych takich jak oprogramowanie „w pudełku" czy oprogramowanie w postaci strony WWW. Dotyczy tworzenia systemów składających się z wielu relatywnie niezależnych od siebie komponentów zwanych usługami, które ukrywają w sobie procesy biznesowe przedsiębiorstwa lub ich części. Przynosi to korzyści w postaci łatwego dodawania kolejnych usług do istniejącej infrastruktury, łatwej integracji i wchłonięcia w nowe procesy biznesowe starego oprogramowania oraz duże możliwości bezpiecznej wymiany i ulepszenia poszczególnych słabo związanych między sobą części składowych [1].

Model referencyjny SOA określa usługę, jako byt informatyczny powołany dla organizacji dostępu do jednej lub więcej możliwości (*capabilities*) [2], przy czym dostęp ten musi

* Doktorant, Katedra Informatyki, Akademia Górniczo-Hutnicza w Krakowie

być zorganizowany nie w sposób dowolny, ale za pomocą ustalonego interfejsu, który powinien być używany zgodnie z jego ograniczeniami (*constraints*) i politykami (*policies*) zawierającymi się w opisie usługi (*service description*) [2].

Zatem usługa jest czymś, co oferuje różne funkcjonalności w sposób ustalany jej opisem. Istnienie takiego dostępnego formalnego opisu w sposób zasadniczy różni SOA od podejścia obiektowego do programowania oraz innych technik i technologii, w których byty korzystające z usług muszą posiadać wiedzę o realizacji komponentów programowych zawierających pożądane funkcjonalności. Nie należy również mylić takiego publicznego interfejsu z interfejsami Javy, które nie uwzględniają wspomnianych ograniczeń i polityk [1].

1.2. Usługi sieciowe

Jedną z możliwości realizacji modelu referencyjnego SOA jest podejście związane z użyciem usług sieciowych (*Web Services*). Definicja przyjęta przez grupę roboczą W3C podaje, że usługą sieciową jest komponent programowy niezależny od platformy i implementacji, dostarczający określonej funkcjonalności i osiągalny za pomocą sieci komputerowej poprzez standardowe protokoły komunikacyjne [3]. Interfejs takiej usługi powinien być opisany w języku zrozumiałym dla komputera takim jak WSDL (*Web Services Description Language*). W celach użycia usługi program klienta powinien dogadać się z nią za pomocą komunikatów SOAP (*Simple Object Access Protocol*).

Agentem w technologii usług sieciowych nazywa się oprogramowanie, które wysyła i otrzymuje wiadomości. Usługa jest z kolei charakteryzowana poprzez dostarczane funkcje [3]. Chodzi o to, że agent jest tylko realizacją abstrakcji usługi. Na przykład implementacje usługi mogą być napisane w różnych językach oprogramowania i będą wtedy różnymi agentami.

Interfejs, czyli zestaw mechanizmów wymiany komunikatów, jest dokumentowany w wiadomości WSD (*Web Service Description*), która jest zrozumiałą przez komputer specyfikacją interfejsów zapisaną w języku WSDL. WSD definiuje postać komunikatów używanych przez usługę, typy danych, protokoły transportowe oraz mechanizmy serializacji, które powinny być zastosowane. Specyfikuje również jeden lub więcej adresów, pod którymi znajdują się agenci dostawcy.

1.3. Kompozycja usług

Potrzeba kompozycji usług wynika wprost z możliwości realizacji bardzo złożonych usług przy użyciu dostawców oferujących elementarne funkcje. Istnieje kilka podejść do problemu kompozycji.

Język WS-CDL (*Web Services Choreography Description Language*) jest promowany w standardzie W3C [4] i oferuje powiązania pomiędzy orkiestrowaniem[1] w BPEL (*Busi-*

[1] Orkiestracja (*orchestration*) skupia się na zachowaniu pojedynczych usług i opisuje przepływ sterowania, zmienne, ograniczenia, wyjątki itd.

ness Process Execution Language) a choreografią[2]. W tym podejściu BPEL jest traktowany jako język orkiestracji specyfikujący zachowanie uczestników choreografii, natomiast sama choreografia za pomocą WS-CDL dopełnia BPEL i opisuje wymianę komunikatów pomiędzy uczestnikami podczas złożonych interakcji.

Semantyczna kompozycja skupia się na możliwościach użycia ontologii usług opisanych w językach ontologicznych. Najbardziej rozpowszechniony jest język opisu ontologii OWL (*Web Ontology Language*) [5] oraz ontologia dotycząca usług OWL-S (*OWL – Semantic Markup for Web Services*) [6]. OWL-S modeluje usługi, używając ontologii złożonej z profilu, modelu oraz uziemienia (*grounding*). Profil opisuje, czego usługa potrzebuje na wejściu i co dostarcza na wyjściu. Model specyfikuje jak usługa pracuje. Uziemienie podaje informacje o tym, jak usługę należy używać. Na podstawie danych zawartych w ontologiach silniki wnioskujące mogą automatycznie komponować usługi [7].

Idea algebraicznej kompozycji procesów jest oparta na prostych opisach usług – typach. Porównując wejścia usług z wyjściami można komponować złożone procesy. Głównym problemem jest co i jak szczegółowo uważać za typ. Zbytnie uproszczenie powoduje utratę niektórych możliwości. Z kolei rozbudowanie modelu doprowadza do sytuacji, w której staje się niepraktyczne podejście z powodu rosnącej złożoności [8].

Usługi mogą być również modelowane i komponowane jako sieci Petriego [9].

W podejściu HTN (*hierarchical task networks*) zadania (usługi) o dużej złożoności są rekursywnie dekomponowane aż do chwili otrzymania operacji podstawowych, które mogą być wykonane bezpośrednio. Oczywiście krok dekomponowania wykonywany jest tylko wtedy, gdy są spełnione ograniczenia zadań bardziej i mniej złożonych. Zadania można dekomponować na kilka sposobów, a więc alternatywne ścieżki też można wziąć pod uwagę.

Zaproponowano również translację opisów modeli procesów z OWL-S do postaci domeny SHOP2 (*Simple Hierarchical Ordered Planner*). Pozwala to traktować kompozycję usług jako problem planowania [10].

2. Hybrydowa kompozycja usług

Pomysł hybrydowej kompozycji usług wywodzi się z próby połączenia zalet tworzenia statycznych planów kompozycji z możliwościami automatyzacji tego procesu. Kiedy opłaca się stosować metodę kompozycji A, robimy to. W przypadku kiedy bardziej korzystne jest stosowanie metody B lub istnienie tej potrzeby zachodzi z innych powodów (na przykład w zleceniu strony zamawiającej kompozycję), w miarę możliwości stosujemy ją. Takie podejście pozwoli użytkownikom uzyskać większy stopień kontroli nad procesem tworzenia planu kompozycji, co jest bardzo ważne w przypadkach, gdy kryteria optymalności lub poprawności planu z różnych powodów nie mogą być formalnie opisane i brane pod uwagę przez system informatyczny.

---

[2] Choreografia (*choreography*) koncentruje się na długoterminowych, ogólnych interakcjach pomiędzy wieloma uczestnikami.

Pierwsze wymaganie stawiane takiemu podejściu to konieczność tworzenia kompozycji usług jako planów złożonych z atomowych wywołań funkcji w ramach jednej sekwencji. Wynika to z tego że tylko taki rodzaj planu może być wykonany bezpośrednio po znalezieniu wszystkich usług rzeczywistych oferujących żądane funkcje. Po drugie, powinna istnieć możliwość wymiany między sobą zarówno pojedynczych wywołań, jak i całych serii na alternatywne. Z punktu poprzedniego wynika niezbędność istnienia w planach abstrakcyjnych, takiego zbioru parametrów, za pomocą którego można jednoznacznie zidentyfikować, które elementy można wykonać w pewnej sekwencji (np. usługę prostą lub złożoną), a które nie.

Aby spełnić stawiane wymagania proponowana, koncepcja przyjmuje następujące założenia:

1. Usługi w ramach koncepcji są traktowane jako zbiory możliwych do wywoływania funkcji, które przyjmują i zwracają określone typy danych.
2. Funkcje oraz usługi mogą, lecz nie muszą, określać specjalne warunki, które powinny być spełnione przed rozpoczęciem ich pracy oraz po jej wykonaniu. Mogą one dotyczyć danych, z którymi pracuje usługa, cech funkcji, znaczeń parametrów wywołania.
3. Globalne warunki specjalne dotyczące składanych planów kompozycji, mają za zadanie określanie rodzajów kompozycji, które system musi zastosować w określonych przypadkach a także dostarczać kryteriów co do optymalności planów na wypadek gdy algorytm kompozycji opracował kilka równoważnych.
4. Wszystkie algorytmy pracujące w ramach koncepcji posługują się wspólnymi formatami planów kompozycji oraz opisów usług.

Aby spełnić te założenia, należy: wybrać lub skonstruować język opisu usług, wybrać lub skonstruować język opisu planów abstrakcyjnych i dobrać algorytmy kompozycji warte zastosowania.

Jako język opisu usług można przyjąć pewne rozszerzenie koncepcji WSDL o nazwie WSDL-S (*Web Service Semantics*), które posiada możliwość opisywania warunków wstępnych wywołań funkcji. Niestety w WSDL-S nie operuje się warunkami końcowymi, dlatego docelowo trzeba albo opakowywać wiadomości tego języka dodatkowymi informacjami, albo poszerzyć jego koncepcję o nowe cechy. Istnieje również ontologia WSMO (*Web Service Modeling Ontology*), która wspiera warunki końcowe oraz wstępne, jednak od dłuższego czasu nie jest już rozwijana.

Język BPEL jako najbardziej rozpowszechniony język opisu planów wykonania kompozycji usług nie posiada wszystkich potrzebnych dla proponowanej koncepcji funkcjonalności i nie spełnia założonych wymagań. Nie ma w nim warunków wstępnych czy końcowych jako takich, nie ma również żadnego wsparcia dla określania parametrów procesu przy doborze algorytmów kompozycji. Dlatego należy albo poszerzyć koncepcję BPEL o wymagane elementy lub opracować i użyć własny język zapisu planów abstrakcyjnych, które na potrzeby uruchamiania można będzie mapować na usługi i funkcje rzeczywiste oraz generować w wyniku proces opisany w BPEL.

Biorąc pod uwagę fakt, że każdy plan abstrakcyjny kompozycji usług w ramach SOA jest tak naprawdę opisem serii wywołań funkcji dostarczanych usługami oraz to, że warun-

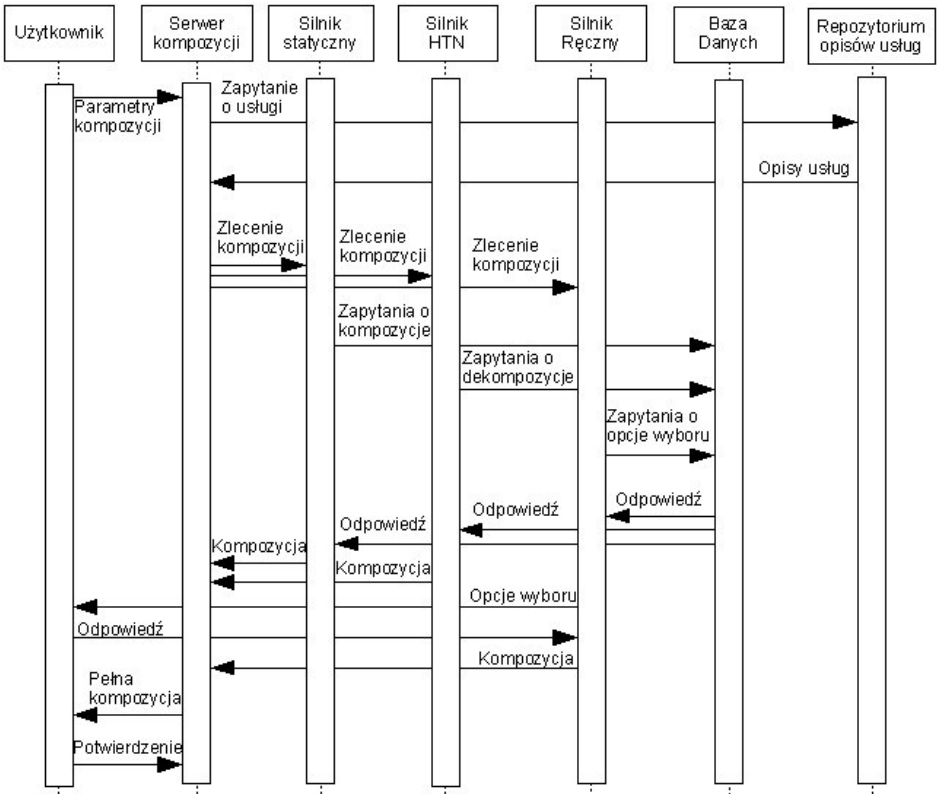
kiem koniecznym poprawnego wywołania omawianych funkcji jest dostarczenie im na wejście prawidłowych typów danych, można przyjąć założenie, że te algorytmy kompozycji, które w swojej istocie działania bazują na dopasowywaniu wejść jednych funkcji z wyjściami innych lub na doborze już zdefiniowanych kompozycji z uwzględnieniem typów danych wejściowych i wyjściowych nadają się do użycia w proponowanej metodzie ponieważ potrafią generować równoznaczne kompozycje.

Każdy kto chciałby zaimplementować proponowaną koncepcję, może sam wybrać interesujące dla niego algorytmy kompozycji opierając się na zdefiniowanych wcześniej warunkach możliwości ich współpracy. Proponowana koncepcja zakłada użycie podejścia HTN, kompozycji statycznej w postaci repozytorium gotowych planów oraz kompozycji ręcznej dostarczanej przez klienta.

3. Koncepcja kompozycji planu abstrakcyjnego

Na rysunku 1 przedstawiono za pomocą diagramu sekwencji propozycję procesu kompozycji planu abstrakcyjnego:

1. Na początku użytkownik systemu określa parametry kompozycji:
   a. Podaje typy parametrów danych dostarczanych systemowi oraz pożądane typy parametrów wyjściowych.
   b. Dostarcza wskazówek co do tego jakich algorytmów kompozycji oraz w których przypadkach należy używać dla tworzenia planu abstrakcyjnego. Niekoniecznie muszą to być reguły jednoznacznie definiujące warunki oraz odpowiadające im algorytmy. Mogą to być na przykład reguły wywołujące w pewnych przypadkach interfejs użytkownika, posługując się którym można wybrać interesujący go algorytm czy już gotowy plan.
   c. Dostarcza kryteria optymalizacji planu.
2. Serwer kompozycji odpytuje repozytorium opisów usług o dostępne usługi i ich funkcje.
3. Serwer kompozycji analizuje dostarczone użytkownikiem dane, dobiera silniki kompozycji oraz wysyła do nich zlecenia kompozycji.
4. Silnik statyczny odpytuje bazę danych o gotowe kompozycje statyczne.
5. Silnik HTN odpytuje o możliwe dekompozycje.
6. Silnik ręczny odpytuje bazę danych o możliwe opcje wyboru użytkownika.
7. Baza danych zwraca wyniki zapytań silnikom.
8. Silnik HTN na ich podstawie buduje plan dekompozycji.
9. Silnik statyczny wybiera interesujące gotowe kompozycje.
10. Silnik ręczny odpytuje użytkownika i podaje mu możliwości do wyboru.
11. Użytkownik odpowiada silnikowi ręcznemu.
12. Silniki zwracają plany oraz ich alternatywy, o ile powstały, serwerowi kompozycji.
13. Jeśli zostały opracowane plany równoważne z punktu widzenia wejść oraz wyjść serwer kompozycji, na podstawie dostarczanych kryteriów, wybiera plan optymalny.
14. Serwer kompozycji scala plany i wysyła wynik do użytkownika.
15. Użytkownik potwierdza kompozycję.



Rys. 1. Diagram sekwencji, ilustrujący proces kompozycji planu abstrakcyjnego

4. Propozycja architektury systemu hybrydowej kompozycji usług

Opierając się na powyższych wnioskach opracowano architekturę systemu hybrydowej kompozycji usług.

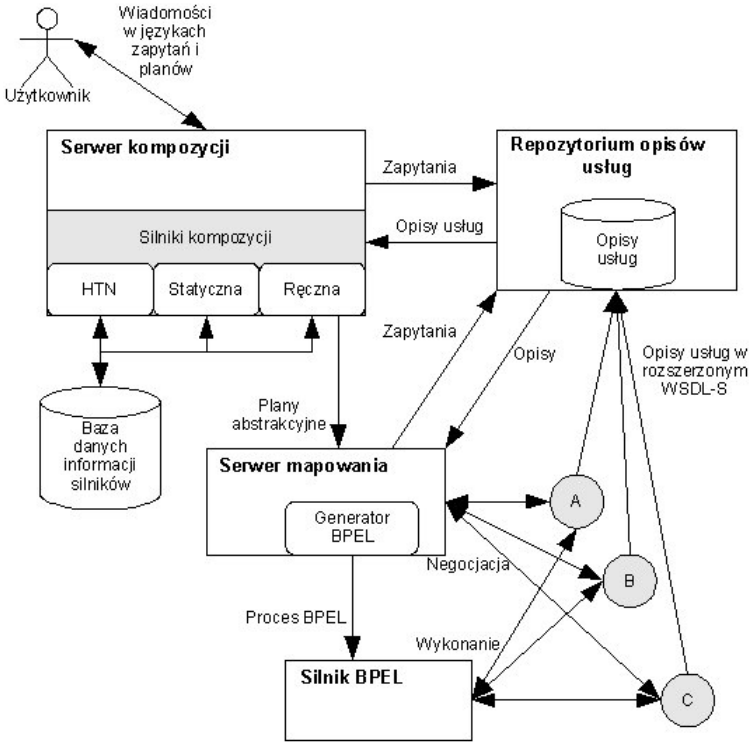
Na rysunku 2 przedstawiono:

- Serwer kompozycji – główny element koncepcji hybrydowej kompozycji usług, pracuje opierając się na informacji zawartej w wiadomości opisującej wymagania i warunki dotyczące pożądanej kompozycji odpowiada za poprawny wybór silników.
- Silnik kompozycji HTN – silnik tworzący abstrakcyjne plany kompozycji usług zbudowany przy użyciu podejścia hierarchicznych sieci zadań
- Silnik kompozycji statycznej – dostarcza ręcznego lub automatycznego wyboru gotowych rozwiązań z biblioteki gotowych planów abstrakcyjnych.
- Silnik kompozycji ręcznej – pozwala użytkownikowi tworzyć ręczne plany kompozycji oraz zmieniać już istniejące.

- Baza danych informacji silników kompozycji – baza danych służąca dla przechowywania informacji potrzebnych dla silników kompozycji (gotowe plany, biblioteka dekompozycji HTN, możliwe opcje wyboru dla kompozycji własnej).
- Serwer mapujący plany abstrakcyjne na rzeczywiste wykonywalne w silniku BPEL.
- Generator BPEL – moduł serwera mapującego dla generacji planów w języku BPEL.
- Silnik BPEL służy do orkiestracji wykonywania kompozycji.
- Przykładowe usługi sieciowe świadczące funkcje A, B oraz C.
- Repozytorium opisów usług – mechanizm przechowywania i dostarczania informacji o dostępnych usługach i ich funkcjach do serwera kompozycji oraz serwera mapowania.

Dodatkowo koncepcja zakłada użycie następujących języków formalnych:

- Język WSDL-S z rozszerzeniem dla opisu warunków końcowych pracy funkcji i usług.
- Język zapytań kompozycji – prosty język formalny dla opisu parametrów wejściowych, żądanych parametrów wyjściowych oraz warunków wyboru silników kompozycji.
- Język planów abstrakcyjnych – prosty język formalny dla opisu planów abstrakcyjnych generowanych silnikami kompozycji.



Rys. 2. Propozycja architektury systemu hybrydowej kompozycji usług

Dla realizacji proponowanej architektury zostały wybrane następujące narzędzia oraz technologie:

1) XML (*Extensible Markup Language*) dla zapisu języków zapytań kompozycji oraz planów abstrakcyjnych.
2) WSDL-S z własnym rozszerzeniem dla warunków końcowych.
3) OpenUDDI – repozytorium opisów usług.
4) Java SE 6 – język programowania do napisania komponentów systemu.
5) SUN JRE 6 – maszyna wirtualna języka Java do uruchamiania komponentów systemu.
6) Postgresql 8 – baza danych dla przechowywania danych silników kompozycji.
7) Apache ODE – silnik BPEL.

5. Podsumowanie

Problematyka doboru metod kompozycji usług nabiera ważności w dzisiejszych czasach, ponieważ proponowane metody w większości przypadków nie mogą w pełni zadowolić wymagających użytkowników. Dzieje się to dlatego, że użytkownicy najczęściej chcą posiadać większy stopień kontroli nad procesem kompozycji, niż to jest możliwe przy użyciu tylko jednego rodzaju algorytmu.

Proponowana metoda pozwala znacznie poszerzyć możliwości optymalizacji tworzenia planów kompozycji poprzez stosowanie różnych algorytmów, tam gdzie zachodzi taka potrzeba.

Literatura

[1] Cetnarowicz K., Belava L., Dyduch T., Koźlak J., Wąchocki G., Żabińska M., *Przegląd metod komponowania usług webowych ze szczególnym uwzględnieniem podejścia agentowego.* Raport dla Instytutu Podstaw Informatyki PAN 2009.

[2] MacKenzie C.M., Laskey K., McCabe F., Brown P., Metz R., *Reference Model for Service Oriented Architecture 1.0.* http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf 2006.

[3] Booth D., Haas H., McCabe F., Newcomer E., Champion I. M., Ferris C., Orchard D., *Web Services Architecture.* http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ 2004.

[4] Kavantzas N., Burdett D., Barreto C., Ritzinger G., Fletcher T., *Web Services Choreography Description Language Version 1.0.* http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/ 2005.

[5] Dean M., Connolly D., Harmelen F., Hendler J., Horrocks I., McGuinness D.L., *OWL Web Ontology Language 1.0.* Reference http://www.w3.org/TR/2002/WD-owl-ref-20020729/ 2004.

[6] Martin D., Burstein M., Hobbs J., Lassila O., McDermott D., McIlraith S., Narayanan S., *OWL-S: Semantic Markup for Web Services*. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ 2004.

[7] Ankolekar A., Burstein M., Hobbs J., Lassila O., Martin D., *DAML-S:Web Service Description for the Semantic Web.* Proceedings of the International Semantic Web Conference (ISWC) 2002, 348–363.

[8] Milanovic N., *Contract-based Web Service Composition.* Rozprawa doktorska w Humboldt-Universität 2006, 23–24.
[9] Hamadi R., Benatallah B., *Petri Net-based model for Web Service Composition.* Proceedings of the 14th Australasian database conference on database technologies 2003, 191–200.
[10] Sirin E., Parsia B., Wu D., Hendler J., Nau D., *HTN planning for Web Service composition using SHOP2.* Web Semantics: Science, Services and Agents on the World Wide Web 2004, 4, 377–396.