AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE   
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI

KATEDRA INFORMATYKI



**Serwisy w architekturze chmury obliczeniowej**

**PRACA MAGISTERSKA**

**Michał Soboń**

mikesobon@gmail.com

**Promotor: dr inż. Piotr Nawrocki**

Kraków 2012

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie (w zakresie wyszczególnionym we wstępie) i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Spis treści

[Wstęp 5](#_Toc339796798)

[Struktura pracy 5](#_Toc339796799)

[1. Koncepcja usługi 6](#_Toc339796800)

[1.1 Charakterystyka nowoczesnej architektury systemu informatycznego 6](#_Toc339796801)

[1.2 Serwisy w nowoczesnych systemach informatycznych 7](#_Toc339796802)

[1.3 Definicja Serwisu 8](#_Toc339796803)

[1.3.1 Typy serwisów 11](#_Toc339796804)

[1.4 Architektura zorientowana na usługi 12](#_Toc339796805)

[1.4.1 Składniki architektury 13](#_Toc339796806)

[1.4.2 Cykl życia systemu opartego o paradygmat SOA 15](#_Toc339796807)

[1.4.3 Zalety podejścia SOA 16](#_Toc339796808)

[1.4.4 Ograniczenia podejścia SOA 17](#_Toc339796809)

[1.5 Podsumowanie 18](#_Toc339796810)

[2. Architektura chmury obliczeniowej 18](#_Toc339796811)

[2.1 Koncepcja chmury obliczeniowej 18](#_Toc339796812)

[2.2 Modele chmury obliczeniowej 18](#_Toc339796813)

[Kolokacja 18](#_Toc339796814)

[Oprogramowanie jako usługa 18](#_Toc339796815)

[Platforma jako usługa 19](#_Toc339796816)

[Infrastruktura jako usługa 19](#_Toc339796817)

[Komunikacja jako usługa 19](#_Toc339796818)

[2.3 Charakterystyka wybranych implementacji chmur obliczeniowych 19](#_Toc339796819)

[Amazon EC2 19](#_Toc339796820)

[Google AppEngine 19](#_Toc339796821)

[Microsoft Azure 19](#_Toc339796822)

[Heroku 19](#_Toc339796823)

[2.4 Aplikacja jako serwis 19](#_Toc339796824)

[3. Architektura systemu udostępniającego oprogramowanie w chmurze obliczeniowej 20](#_Toc339796825)

[3.1 Analiza wymagań 20](#_Toc339796826)

[3.1.1 Wymagania funkcjonalne 20](#_Toc339796827)

[Portal udostępniający aplikacje 20](#_Toc339796828)

[Przykładowa aplikacja kliencka 1 – Staff Manager 21](#_Toc339796829)

[Przykładowa aplikacja kliencka 2 – Warehouse Manager 21](#_Toc339796830)

[3.1.2 Wymagania niefunkcjonalne 21](#_Toc339796831)

[Portal udostępniający aplikacje 21](#_Toc339796832)

[Aplikacje klienckie (wersja ogólna – podzielić na Staf i Warehouse Manager) 21](#_Toc339796833)

[3.2 Opis architektury systemu 22](#_Toc339796834)

[3.2.1 Architektura portalu udostępniającego aplikacje jako usługi 22](#_Toc339796835)

[Single Sign – On 22](#_Toc339796836)

[3.2.2 Architektura aplikacji - usług 22](#_Toc339796837)

[4. Implementacja systemu 22](#_Toc339796838)

[4.1 Wykorzystane technologie 22](#_Toc339796839)

[4.1.1 Google AppEngine 22](#_Toc339796840)

[4.1.2 Microsoft Azure 22](#_Toc339796841)

[4.1.3 Amazon EC2 22](#_Toc339796842)

[4.1.4 Heroku 22](#_Toc339796843)

[4.1.5 Play! 22](#_Toc339796844)

[4.1.6 Spring 22](#_Toc339796845)

[4.2 Model architektury rozproszonej 22](#_Toc339796846)

[4.3 Implementacja portalu udostępniającego aplikacje 22](#_Toc339796847)

[4.4 Implementacja przykładowych aplikacji - usług 23](#_Toc339796848)

[5. Testy systemu 23](#_Toc339796849)

[Podsumowanie 23](#_Toc339796850)

[Wykaz skrótów 24](#_Toc339796851)

[Spis ilustracji 24](#_Toc339796852)

[Spis tabel 25](#_Toc339796853)

[Spis listingów 25](#_Toc339796854)

[Bibliografia 26](#_Toc339796855)

[Spis ilustracji i tabel 32](#_Toc339796856)

[Aneksy 32](#_Toc339796857)

# Wstęp

W obecnych czasach dostęp do zasobów sieci komputerowej czy Internetu staje się nieodłącznym atrybutem komputera. Wraz z jego rozwojem ludzie zyskali dostęp do coraz większej wiedzy o technologii przez znacząco wzrosły ich wymagania dotyczące kompleksowości posiadanych przez nich rozwiązań informatycznych. Główną przeszkodą dla użytkowników, w osiągnięciu tego celu, były wysokie ceny złożonego oprogramowania oraz sprzętu niezbędnego do jego poprawnego funkcjonowania – istniała spora grupa użytkowników dla których zakup zaawansowanego, kompleksowego systemu informatycznego był nieopłacalny.

Jednym z rozwiązań tego problemu jest dostarczenie usługi polegającej na dostępie do aplikacji (bądź ich zestawu) wraz z możliwością bezpiecznego przechowywania niezbędnych danych. Jej koszt jest zależny od intensywności użytkowania, co znacząco poszerza rynek potencjalnych nabywców. Takie podejście jest korzystne zarówno dla odbiorców, jak i twórców aplikacji. Klient nie musi martwić się o zapewnienie stosownego sprzętu wraz z jego obsługą przez co może całkowicie skupić się na swoim biznesie. Dostawca oprogramowania zyskuje unifikacje środowiska, w którym działa aplikacja, powoduje to zmniejszenie ilości wersji oprogramowania wykorzystywanych przez klientów. Dodatkowo istnieje możliwość zmiany sposobu finansowania twórców oprogramowania - zamiast jednorazowej opłaty licencyjnej mogą pobierać abonament, którego wysokość zależy od stopnia wykorzystania usług. Zwiększa to stabilność dochodów, co skutkuje zwiększeniem możliwości rozwoju.

Niniejsza praca ma za zadanie przedstawić możliwości wykorzystania architektury opartej o serwisy działające w chmurach obliczeniowych realizującej model Software as a Service. Opisuje także architekturę, implementację oraz testy przykładowej platformy. Przykładowa aplikacja składa się z kilku podstawowych komponentów:

* Portalu zapewniającego użytkownikom dostęp do aplikacji jako usług, jest on odpowiedzialny za przechowywanie danych o użytkowniku, rozliczenia finansowe oraz zapewnie Single Sign-On
* zestawu aplikacji klienckich udostępnianych przez Portal jako usługi

Portal jest odpowiedzialny za zapewnienie użytkownikom dostępu do aplikacji klienckich w nim zarejestrowanych. Aplikacje dzięki implementacji odpowiedniego interfejsu rejestrują się w Portalu i mogą być udostępniane klientom.

## Struktura pracy

Celem niniejszej pracy jest prezentacja możliwości wykorzystania chmur obliczeniowych jako środowiska działania usług oraz stworzenie przykładowej implementacji.

Pierwszy rozdział pracy prezentuje koncepcję usług (serwisów) oraz architektury zorientowanej na usługi. Zawiera także analizę możliwości ich zastosowań w systemach informatycznych.

Kolejny rozdział poświęcony jest chmurom obliczeniowych, zawiera on także, oprócz wyjaśnienia pojęcia chmury, omówienie różnych jej modeli wraz z prezentacją istniejących implementacji. W tym rozdziale dokonany jest także krotki przegląd ich komercyjnego wykorzystania.

Trzeci rozdział poświęcony jest przykładowej aplikacji stworzonej na potrzeby tej pracy. Zawiera opis wymagań funkcjonalnych i niefunkcjonalnych, prezentuje zasadę działania wraz z opisem najważniejszych komponentów. Omówione zostały także technologie wykorzystane w czasie implementacji

W podsumowanie zawarte są dalsze możliwości rozwoju projektu oraz analiza potencjalnych problemów, które mogą pojawić się w dalszych fazach rozwoju. W ostatniej części pracy przedstawione są wnioski powstałe podczas pracy nad systemem wraz z krótkim komentarzem.

1. Koncepcja usługi

Usługi (serwisy) są głównym elementem architektury nowoczesnych systemów informatycznych. Są odpowiedzialne za realizacje poszczególnych funkcjonalności biznesowych, znajdują szerokie zastosowanie gdyż upraszczają budowę systemu oraz zwiększają jego elastyczność. Dobrze zaprojektowane serwisy mogą być wielokrotnie wykorzystywane dzięki wzajemnym słabym powiązaniom oraz dobrze zdefiniowanym interfejsom, obniża to koszty produkcji i utrzymania oprogramowania.

* 1. Charakterystyka nowoczesnej architektury systemu informatycznego

Współczesne systemy informatyczne to w większości systemy rozproszone o heterogenicznej charakterystyce. Możemy wyróżnić szereg cech, które powinien posiadać nowoczesny system:

* zorientowanie na usługi – system składa się z wielu niezależnych serwisów połączonych protokołem komunikacyjnym, które realizują poszczególne funkcjonalności lub ich części
* zorientowanie na zdarzenia – paradygmat programowania według którego do systemu są nieustannie dostarczane zdarzenia, które muszą zostać przetworzone
* słabe powiązanie komponentów (Loosely coupled) – cecha systemu polegająca na ograniczeniu do minimum konieczności by poszczególne komponenty systemu posiadały wiedzę o pozostałych. Jest bardzo to istotne z punktu widzenia architektury systemu ponieważ nadmierne powiązania utrudniają rozwój aplikacji oraz wpływają negatywnie na jej jakość – poprzez utrudnienie testowania
* zapewnienie równoległego procesu wsparcia – użytkownicy systemu (zarówno programiści rozwijający inne komponenty jak i systemy zależne oraz klienci biznesowi) mogą liczyć na dostęp do dokumentacji bądź linii wsparcia
* możliwość łatwej integracji z innymi systemami – można to osiągnąć dzięki dobremu zdefiniowaniu interfejsów udostępnianych przez system jasno określających dostępne funkcjonalności
* możliwość pracy z istniejącymi systemami i sprzętem – podczas projektowania systemu należy wziąć pod uwagę systemy już wykorzystywane przez klienta i uwzględnić je przy tworzeniu nowego systemu. Przykładowo, jeśli Klient posiada 2 000 serwerów działających pod kontrola systemu Solaris należy rozważyć wykorzystanie tej platformy – wprowadzanie nowego systemu operacyjnego bez ważnych powodów podnosi koszty(zatrudnienie lub wyszkolenie administratorów) oraz utrudnia utrzymanie
  1. Serwisy w nowoczesnych systemach informatycznych

Fundamentem nowoczesnych systemów informatycznych jest konglomerat serwisów (usług) realizujących poszczególne zadania biznesowe. W większości przypadków budowany system jest rozproszony – poszczególne usługi działają w rożnych środowiskach i lokalizacjach. Usługi są wykorzystywane w miarę potrzeb – w chwili gdy nastąpi zdarzenie wykonywana jest stosowna akcja przez zadany serwis. Dodatkowo, dzięki istnieniu w systemie rejestru usług (wzorzec projektowy *service locator*) poszczególne komponenty nie muszą posiadać informacji gdzie znajdują się poszukiwane przez nie serwisy – wystarczy, że będą znały adres rejestru, który dostarczy im lokalizację oraz metodę komunikacji z żądaną usługą.

Zastosowanie service lokatora wprowadza abstrakcję obejmującą fizyczną lokalizację usługi. Umożliwia także możliwość dynamicznego dodawania nowych usług bez konieczności rekompilacji całości systemu. Pozwala to na optymalizację pracy aplikacji w czasie jej działania np. w przypadku stwierdzenia przez system, że w rejestrze znajduje się lepsza biblioteka realizująca zadaną funkcjonalność obliczeniową istnieje możliwość jej dynamicznej podmiany.

Pomimo tych zalet istnienie *service locatora* niesie za sobą pewne niebezpieczeństwa. Głównym z nich jest możliwość nieautoryzowanej podmiany kodu aplikacji, co może doprowadzić do przejęcia kontroli nad systemem przez nieuprawnione osoby. Dodatkowo istnienie rejestru usług wprowadza centralny punkt systemu – jego awaria poważnie wpływa na całość systemu. Wymagana jest odpowiednia implementacja tego komponentu obsługująca wykorzystywanie go przez wiele wątków równocześnie oraz stosowna architektura fizyczna zapewniająca wysoką dostępność. Dodatkowo należy zadbać o stosowne procedury bezpieczeństwa ograniczające możliwość niepożądanego wykorzystania tego komponentu, dotyczy to w szczególności procedur walidacji dołączanych komponentów do systemu celem uniknięcia wprowadzenia wadliwego czy też w skrajnym przypadku złośliwego kodu.

* 1. Definicja Serwisu

Eben Hewitt w książce *Java SOA cookbook* definiuje serwis jako komponent oprogramowania posiadający następujące właściwości:

* Serwis jest zdefiniowany przez interfejs, który może udostępniać jego funkcjonalność komponentom stworzonym w oparciu o inne platformy
* Interfejs serwisu określa jakie operacje może on wykonywać na obiektach biznesowych
* Interfejs serwisu jak i jego implementacja może być może być dekorowana (wzorzec projektowy dekorator) dodatkowymi funkcjonalnościami.

Powyższa definicja umożliwia osobie nieznającej tematu na łatwe pozyskanie wyobrażenia o serwisie rozumianym w kontekście architektury oprogramowania. Nie wyczerpuje ona jednak tematu i w wielu przypadkach może okazać się ona niewystarczająca – trzeba spojrzeć na serwis w szerszym kontekście obejmującym także środowisko jego działania.

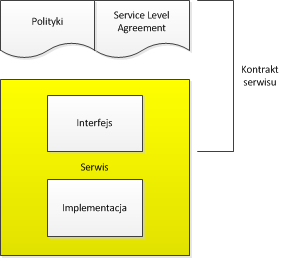
Bardziej uniwersalną definicję serwisu możemy znaleźć w książce *Applied SOA*, gdzie serwis (usługa) jest zdefiniowany jako dyskretna część biznesowej funkcjonalności dostępna dzięki kontraktowi serwisu.

Kontrakt usługi określa interakcje pomiędzy dostawcą usługi a jej klientem. W jego skład wchodzą:

* Interfejs serwisu
* Dokumentacja interfejsu
* Zbiór reguł (polityk) określających działanie serwisu
* Zasady zapewnienia jakości usług (QoS - *Quality of service*)
* Specyfikacja wydajności serwisu

Zasadniczą różnicą pomiędzy usługami a pozostałymi elementami oprogramowania jest sposób zarządzania – serwis jest bezpośrednio zarządzany, dotyczy to także cyklu życia serwisu (od tworzenia, poprzez wdrożenie, utrzymanie oraz rozbudowę). Jego wydajność czy QoS jest definiowane przy użyciu *Service Level Agreement (SLA)* .

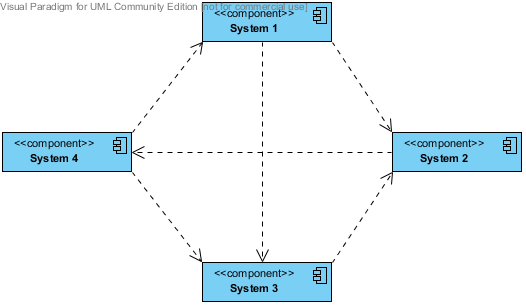
Relację pomiędzy serwisem a kontraktem przedstawia Rysunek 1 . Kontrakt serwisu jest realizowany przez interfejs w oparciu o specyfikację dostarczoną w zbiorze polityk oraz SLA. Należy zauważyć, że do kontraktu nie należy implementacja usługi. Umożliwia to jej modyfikacje (w ramach określonych przez kontrakt) bez obawy, że wpłynie to na zachowanie innych komponentów systemu.



Rysunek . Relacja serwisu z jego kontraktem

Funkcjonalności udostępniane przez serwis są reprezentowane poprzez jego interfejs. Jest to punkt spotkania konsumentów (oraz ich potrzeb) z dostawcami usług (wraz z ich ofertą). Określa on możliwe operacje wraz z parametrami niezbędnymi do ich przeprowadzenia. Kolejnym elementem interfejsu usługi są protokoły komunikacyjne umożliwiające dostęp do dostarczanych operacji. Oprócz definiowania udostępnianych funkcjonalności, Odpowiednią ziarnistością rozumianą jako wielkość usługi. Ziarnistość określa jaka ilość funkcjonalności biznesowych jest realizowana przez dany serwis. Nie należy łączyć tego pojęcia z fizyczną wielkością usługi wyrażoną przez ilość linii kodu czy mocą maszyny niezbędnej do jej wykonania.

* Enkapsulacja – jej zadaniem jest ukrycie implementacji (sposobu ich realizacji) oraz dostarczenie użytkownikowi tylko tych operacji, które są wymagane przez zdefiniowany interfejs.
* Luźne powiązania – opisują wzajemne zależności pomiędzy konsumentem a dostawcą. W celu zapewnienia lepszego zarządzania systemem oraz jego łatwiejszego rozwoju istotne jest zredukowanie wzajemnych powiązań do minimum oraz ich dobre udokumentowanie opisujące ich cel oraz zakres wpływu na pozostałe elementy relacji. Przykład wzajemnych powiązań komponentów systemu przedstawia , w prezentowanym systemie możemy zaobserwować ścisłe powiązanie komponentów co stanowi zły wzorzec projektowy (potocznie zwany *spaghetti*). Charakteryzuje się on trudnościami w rozbudowie i modyfikacji systemu oraz podatnością na błędy.



Rysunek – Przykład wzajemnych powiązań komponentów systemu

* Ograniczona odpowiedzialność – poszczególne serwisy są odpowiedzialne za realizację dyskretnej funkcjonalności lub zarządzanie danym zasobem. W celu zapewnienia spójności oraz uniknięciu redundancji zadana funkcjonalność nie powinna być równolegle dostarczana przez więcej niż jeden serwis.
* Autonomia – umożliwia wdrożenie, utrzymanie oraz modyfikację serwisu niezależnie od pozostałych. Cykl życia niezależnego (autonomicznego) serwisu nie jest zależny od cykli życia innych usług.
* Reużywalność – dany serwis jest wykorzystywany w orkiestracji celem dostarczenia rożnych funkcjonalności biznesowych. Jest to możliwe dzięki jego charakterystyce wyróżniającej się luźnymi powiązaniami, modularnością, inkapsulacją, ograniczoną odpowiedzialnością oraz autonomią.
* Dynamiczną lokalizacją oraz łączeniem – dzięki centralnemu repozytorium serwisów możemy znajdować lokalizacje poszczególnych serwisów oraz uzyskiwać dane niezbędne do ich wykorzystania. Zapewnia to łatwą możliwość podmiany poszczególnego serwisu bez wpływania na całość systemu oraz umożliwia szybki dostęp do informacji o poszczególnych usługach
* Bezstanowość – odpowiedź serwisu nie jest zależna od kontekstu operacji, innymi słowy zwracany wynik nie jest zależny od poprzednich działań oraz nie wpływa na te, które po nim nastąpią. Bezstanowość zapewnia lepszą elastyczność, skalowalność oraz zwiększa stabilność systemu. Jest to jedno z założeń architektury, jednakże nie zawsze jest ono praktyczne, zwłaszcza przy czasochłonnych interakcjach zachodzących między serwisami.
* Udostępnianie informacji o sobie – Kontrakt serwisu zapewnia informację o interfejsie usługi, możliwych operacjach oraz ich parametrach. Opisuje także powiązania wykorzystywane w celu dostarczenia wyniku.
* Zdolność do kompozycji – możliwość budowania złożonych zorkiestrowanych serwisów na podstawie istniejących usług oraz umożliwienie ich późniejszego wykorzystania celem dalszej kompozycji.
* Niezależność od lokalizacji, języka programowania, wykorzystywanych protokołów – usługi są projektowane by były przeźroczyste dla użytkownika – są dostępne dla każdego uprawnionego użytkownika niezależnie od wykorzystywanych przez niego platform sprzętowych czy programowych oraz w każdej dozwolonej lokalizacji (ograniczenie do intranetu lub dostępność w całym Internecie)
* Relacje pomiędzy konsumentami a usługami są opisane w polityce usługi bądź SLA, innymi słowy jest ściśle określone w jaki sposób konsument może wykorzystać serwis.
  + 1. Typy serwisów

Systemy informatyczne znajdują zastosowanie we wszystkich dziedzinach życia, pomimo różnorodności ich przeznaczenia możemy wyróżnić 3 podstawowe grupy serwisów, które możemy spotkać w zdecydowanej większości istniejących rozwiązań. Są to:

* Serwisy zadaniowe – usługi realizujące różnorakie funkcjonalności biznesowe, przykładowo obliczające ratę kredytu lub sprawdzające poprawność formatu danych osobowych. Rozmiar serwisów tego typu jest rożny – od małych realizujących ogólne zadania po duże wyspecjalizowane usługi implementujące zaawansowane procesy biznesowe. Należy podkreślić, że wraz ze wzrostem zaawansowania serwisu zadaniowego maleje jego reużywalność, dlatego w idealnym przypadku skomplikowane usługi powinny być skomponowane z mniejszych, prostszych i reużywalnych elementów.
* Serwisy encyjne – ich zadaniem jest udostępnianie encji danych, na przykład mogą udostępniać dane klientów, szczegóły rachunków bankowych czy kart kredytowych. Serwisy encyjne są zazwyczaj średniego bądź dużego rozmiaru oraz są wielokrotnie wykorzystywane przez inne komponenty systemu dzięki swojej niezależności od procesów biznesowych. Uogólniając, serwisy encyjne dostarczają i odbierają dane z komponentów przetwarzających informacje.
* Serwisy decyzyjne – wykorzystują dostarczone reguły biznesowe w celu podjęcia decyzji (najczęściej tak/nie). Zazwyczaj ich rozmiar wacha się od małych do średnich. Są często wykorzystywane w komponowaniu większych drzew decyzyjnych.

Wyżej wymieniony podział na typy należy traktować jako pewną generalizację – nie jest to ścisła klasyfikacja. W niektórych przypadkach możemy spotkać specyficzne implementacje serwisów niemieszczące się w przedstawionych granicach.

* 1. Architektura zorientowana na usługi

Architektura zorientowana na usługi jest to paradygmat tworzenia oprogramowania jako grupy zorkiestrowanych, współpracujących ze sobą usług, które działając ze sobą spełniają wymagania użytkownika. Istotne jest słabe powiązanie pomiędzy poszczególnymi usługami – umożliwia to wielokrotne użycie serwisów oraz ułatwia ich testowanie. Wzajemna komunikacja pomiędzy współpracującymi serwisami jest prowadzona przy użyciu zdefiniowanego protokołu komunikacyjnego. Komunikaty przesyłane są w formacie sprecyzowanym w protokole (najczęściej oparty o XML, JSON).

Trzeba rozgraniczyć dwa rodzaje systemów informatycznych opartych o serwisy:

* Systemy wykorzystujące pojedyncze serwisy
* Systemy zbudowane w oparciu o paradygmat SOA

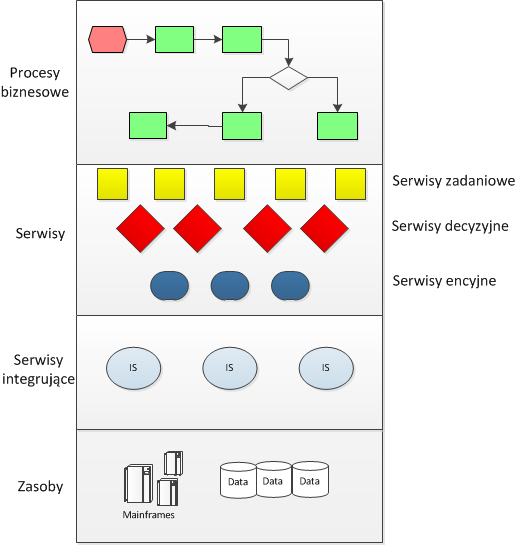
Źródłem prawdziwej wartości aplikacji zbudowanej w oparciu o paradygmat SOA jest implementacja złożonych procesów biznesowych przy wielokrotnym wykorzystaniu serwisów, dzięki możliwości ich elastycznej kompozycji. Wymaga to od architektury zapewnienia środowiska, które umożliwi tworzenie i kompozycję usług. Zadanie to jest raczej proste w małych, jednolitych organizacjach. Jednakże w bardziej złożonych środowiskach biznesowych, których poszczególne organizacje posiadają własne implementacje danych serwisów, pojawia się potrzeba nałożenia pewnych dodatkowych wymagań do specyfikacji serwisów:

* Zbliżona charakterystyka usług (kształt, forma, rozmiar)
* Zgodność ze standardami przyjętymi w organizacji
* Zapewnienie kompatybilności komunikacji na poziomie technicznym
* Stosowanie komunikatów o zgodnej semantyce
* Zapewnienie rozłącznej odpowiedzialności poszczególnych serwisów
  + 1. Składniki architektury

Celem architektury systemu jest odpowiedź na trzy podstawowe pytania: jakie są istotne jej elementy? Jakie relacje je łączą? W jaki sposób należy je połączyć by dostarczyć zadaną wartość?

W SOA główne elementy składowe architektury to:

* Procesy – wysokopoziomowe funkcje biznesowe występujące w organizacji
* Serwisy - modułowe elementy składowe funkcjonalności biznesowych
* Integracja – połączenie z istniejącymi systemami oraz dostawcami danych
* Istniejące systemy – systemy odziedziczone, nierzadko kluczowe z punktu widzenia organizacji
* Dokumenty – wysokopoziomowe jednostki zawierające informacje biznesowe, przykładowo zamówienia, faktury
* Semantyka – znaczenie informacji przetwarzanych w procesach biznesowych
* Transformacja – konwersja formatu bądź semantyki informacji
* Komunikacja – możliwość wymieniania informacji pomiędzy serwisami



Rysunek . Warstwowy model architektury SOA

Rysunek 3 przedstawia warstwowy model architektury SOA. Poszczególne warstwy posiadają różne odpowiedzialności, które są przedstawione w poniższym zestawieniu:

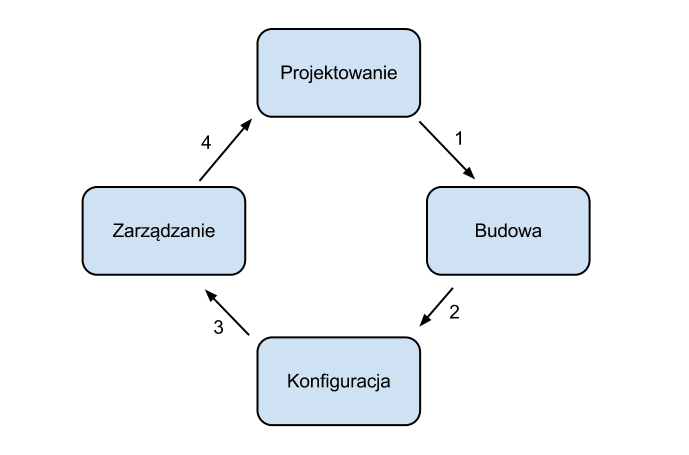
* **Warstwa procesów biznesowych** znajdują się w niej procesy biznesowe czyli sekwencje operacji, których wykonanie w zdefiniowanej kolejności realizują zadany cel. Modelowanie procesów odbywa się przy wykorzystaniu systemów zarządzania procesami biznesowymi (Business Process Management System - BPMS) z wykorzystaniem notacji BPMN (Business Process Modeling Notation). Umożliwia to orkiestrację dostępnych serwisów celem realizacji pożądanych operacji dokonywanych na dokumentach. Przykładem takiego działania jest wystawienie przez system nowej faktury, czy realizacja złożonego zamówienia.
* **Warstwa serwisów biznesowych** zawiera usługi realizujące poszczególne funkcjonalności biznesowe. Udostępnia interfejs wykorzystywany w realizacji procesów biznesowych oraz wykorzystuję warstwę integracji. Serwisy są konfigurowanie i zarządzane w sposób zapewniający realizację SLA. W swoim działaniu wykorzystują semantykę dostarczonych informacji. Serwisy tej warstwy możemy pogrupować na serwisy zadaniowe implementujące funkcjonalności biznesowe, serwisy decyzyjne oparte o silniki regałowe oraz serwisy encyjne zajmujące się przechowywaniem wysokopoziomowych danych
* **Warstwa integracji** składa się z serwisów integrujących warstwę serwisów biznesowych z systemami odziedziczonymi oraz zewnętrznymi systemami. Istnienie tej warstwy jest kluczowe w zapewnieniu elastyczności środowiska – ogranicza ona wpływ zmian w zewnętrznych aplikacjach na wyższe warstwy aplikacji. Serwisy integrujące dokonują zazwyczaj konwersji danych pochodzących z warstwy zasobów do formatów zgodnych ze standardami wykorzystywanymi w budowanym systemie.
* **Warstwa zasobów sprzętowych wraz z systemami odziedziczonymi** dostarcza transakcyjne operacje reprezentowane przez pojedyncze jednostki logiczne. W większości przypadków wykonanie takiej operacji skutkuje odczytaniem bądź zapisaniem rekordu do bazy danych aplikacji. Komunikacja z taką aplikacją odbywa się przy pomocy udostępnionego interfejsu umożliwiającego wykonywanie zadań oraz odczytywanie zwróconych wyników. Przykładem aplikacji z tej warstwy może być aplikacja do zarządzania relacjami z klientem (CRM).

* + 1. Cykl życia systemu opartego o paradygmat SOA

Jako cykl życia aplikacji rozumiemy kroki jakie należy podjąć by zastosować paradygmat SOA w tworzonej aplikacji. Możemy wyróżnić najważniejsze fazy cyklu życia:

* Projektowanie
* Budowanie
* Konfiguracja
* Zarządzanie/rozwój

Przejścia pomiędzy poszczególnymi fazami życia systemu przedstawia . Jak widać przedstawiony cykl życia nie rożni się w znaczący sposób od cyklów życia systemów nie tworzonych w oparciu o SOA.



Rysunek - cykl życia systemu

Najważniejszym etapem z punktu widzenia SOA jest projektowanie, błędy popełnione na tym etapie są bardzo trudne w usunięciu – wymagają zmian podczas pozostałych 3 etapów tworzenia oprogramowania. W tym etapie kluczowe jest zebranie, zrozumienie i zamodelowanie wszystkich procesów biznesowych, w których system ma uczestniczyć. Projektując interfejsy należy pamiętać o zapewnieniu stosownej ziarnistości, słabym powiązaniu pomiędzy usługami oraz zapewnieniu maksymalnej uniwersalności serwisów. Z uwagi na fakt, że serwisy z założenia mają być reużywalne musimy zapewnić ich bezstanowość – informacje o poprzednich akcjach nie mogą być przechowywane w serwisach. Trzeba mięć także na uwadze hermetyzację, czyli ukrycie szczegółów implementacyjnych oraz konfiguracyjnych za interfejsem.

* + 1. Zalety podejścia SOA

Zbudowanie aplikacji w oparciu o paradygmat SOA przynosi wiele korzyści zarówno dla twórców oprogramowania jak i dla klientów zlecających tworzenie systemów. Najważniejszą z korzyści jest zwiększenie elastyczności i możliwości rekonfiguracji tworzonego systemu, co umożliwia szybsze i tańsze przystosowywanie systemu do zmian w procesach biznesowych klienta.

Kolejną zaletą jest możliwość tworzenia poszczególnych serwisów z wykorzystaniem różnych technologii oraz platform jest to możliwe dzięki zunifikowanej komunikacji opartej o protokół http. Jest to szczególnie ważne w przypadku integracji z systemami odziedziczonymi, których rozwój jest bardzo utrudniony z uwagi na często przestarzałe technologie lub niemożliwy ze względu na brak dostępu do kodu źródłowego istniejącego systemu.

Wszystkie wyżej wymienieni one cechy pozytywnie wpływają na jakość rozwiązania, która polega na uproszczeniu utrzymania istniejącego systemu oraz umożliwieniu prostej i szybkiej rekonfiguracji, tak by nadążyć za zmieniającymi się procesami biznesowymi. Dzięki realizacji poszczególnych funkcjonalności poprzez wyspecjalizowane serwisy (zasada jednej odpowiedzialności) dodanie kolejnych sprowadza się do implementacji nowego serwisu, który w wielu przypadkach może zostać po prostu skomponowany z już istniejących. Powoduje to znaczne obniżenie kosztów związanych z systemem zarówno tych powstałych przy jego implementacji jak i tych generowanych przez system w czasie jego działania.

* + 1. Ograniczenia podejścia SOA

Stworzenie aplikacji zgodnie z paradygmatem SOA wymaga dodatkowych nakładów pracy podczas projektowania i implementacji, co nie zawsze jest opłacalne z biznesowego punktu widzenia. Podejście zorientowane na usługi nie jest odpowiednie w przypadku aplikacji:

* nie działających w środowisku rozproszonym, w tym aplikacji, których zadaniem jest obsługa zdarzeń pochodzących z interfejsu graficznego (GUI)
* nie wymagających integracji z innymi komponentami systemu
* których czas życia jest ograniczony, a aplikacja zostanie w późniejszym czasie zastąpiona innym rozwiązaniem
* których zadaniem jest obsługa zdarzeń pochodzących z interfejsu graficznego (GUI)
* w systemach homogenicznych, w których wprowadzenie komunikacji z wykorzystaniem tekstowego protokołu komunikacyjnego przyniesie straty wydajności

Tworząc aplikację opartą o paradygmat SOA musimy cały czas pamiętać o fakcie, że nie rozwijamy systemu w próżni – musimy mieć na uwadze kontekst w którym mają działać zarówno poszczególne serwisy jak i całość systemu. Kolejnym wyzwaniami jest zapewnienie luźnych powiązań między komponentami systemu, zdefiniowanie adekwatnych interfejsów, czy zapewnienie spójnej komunikacji. Te wszystkie wyzwania nie mogą mieć przypadkowych rozwiązań – dobrze zdefiniowana architektura powinna wskazywać rozwiązania najważniejszych składników systemu jak takich jak mechanizm komunikacji czy sposób wersjonowania serwisów. Projektując architekturę musimy dobrze rozumieć implementowane procesy biznesowe, umożliwi to nam zapewnienie spójności systemu wraz ze stosowną ilością wzajemnych powiązań.

* 1. Podsumowanie

W tym rozdziale została zaprezentowana koncepcja serwisów wraz z wyszczególnieniem ich najważniejszych typów. Dokonano tego na podstawie rożnych definicji spotykanych w literaturze. Usługi zostały określone jako podstawowa jednostka budowy systemu, która może być wielokrotnie wykorzystywaną oraz komponowane celem implementacji zaawansowanych funkcjonalności biznesowych. Zdefiniowano także niezbędne wymagania, stawiane przed serwisem, umożliwiające jego integrację z system opartym o paradygmat SOA.

Kolejnym poruszonym tematem jest architektura zorientowana na usługi. W rozdziale omówiono poszczególne zadania stawiane przed architekturą oraz sposób ich realizacji przez SOA. Przedstawiono także cykl życia systemu, co stanowi swoistą skróconą instrukcję mówiącą co należy zrobić by stworzyć system oparty o ten paradygmat. Na końcu wypunktowano oraz krótko omówiono zalety oraz ograniczenia podejścia zorientowanego na serwisy.

1. Architektura chmury obliczeniowej

Chmura obliczeniowa to system komputerowy dostarczający obliczenia, składowanie danych czy infrastrukturę jako usługi. Historycznie wywodzi się z systemów gridowych. Jest zbudowany w oparciu o współdzielone, reużywalne zasoby z wykorzystaniem technik wirtualizacji umożliwiających w izolowanie danych poszczególnych użytkowników. Dostęp do chmury obliczeniowej odbywa się poprzez sieć komputerową.

* 1. Koncepcja chmury obliczeniowej
  2. Modele chmury obliczeniowej

Możemy wyróżnić kilka podstawowych modeli chmur obliczeniowych

### Kolokacja

*Kolokacja – usługa polegająca na udostępnieniu miejsca w data center przeznaczonego do umieszczenia serwera stanowiącego własność klienta*

### Oprogramowanie jako usługa

*Oprogramowanie jako usługa (software as a service – SaaS) – najstarszy i najbardziej dojrzały model chmury obliczeniowej. Polega na udostępnieniu użytkownikowi dostępu do aplikacji znajdującej się gdzieś w sieci komputerowej. Pozwala mu to na korzystanie z produktu kiedy chce i gdzie chce, dodatkowo odbiorca nie musi troszczyć się o sprzęt i oprogramowanie niezbędne do działania aplikacji - należy to do odpowiedzialności dostawcy. Ewentualna opłata za użytkowanie może być naliczana w zależności od dokonanej ilości operacji, bądź abonamentowo za określony czas. Model SaaS posiada pewne niedogodności jak obawy o bezpieczeństwo danych czy możliwość nadmiernej ingerencji dostawcy. Przykładową komercyjną platformą SaaS jest Salesforce.com udostępniająca aplikacje do zarządzania.*

### Platforma jako usługa

*Platforma jako usługa (platform as a service - PaaS) – jest to kombinacja platformy deweloperskiej oraz istniejących rozwiązań z zakresu chmur obliczeniowych. Zapewnia infrastrukturę niezbędną do wdrożenia aplikacji stworzonych w oparciu o dostarczone API, jak również umożliwia rozwijanie istniejących aplikacji. Istotną cechą jest dynamiczna skalowalność – pozwala to skupić się twórcom aplikacji na funkcjonalności produktu, za dostarczenie niezbędnych zasobów odpowiedzialny jest dostawca. Przykładowe implementacje to Google AppEngine czy Microsoft Azure.*

### Infrastruktura jako usługa

*Infrastruktura jako usługa (infrastructure as a service - IaaS) – dostarcza infrastrukturę, jest platformą opartą o wirtualizację. Przeznaczona jest dla klientów posiadających w swoich szeregach specjalistów odpowiedzialnych za tworzenie i zarządzanie systemami informatycznymi, jednakże nieposiadających (bądź nie chcących posiadać) infrastruktury. Stworzenie własnego data center jest drogą inwestycją, obniżającą zdolność firmy do dynamicznego zarządzania kosztami. IaaS pozwala elastycznie dostosowywać infrastrukturę IT do bieżących potrzeb przedsiębiorstwa co powoduje zmniejszenie strat wynikających z istnienia niewykorzystywanych zasobów. Za przykład może posłużyć Amazon EC2 czy GoGrid.*

### Komunikacja jako usługa

* *Komunikacja jako usługa (Communication as a service – CaaS) – platforma zapewniająca środowisko telekomunikacyjne niezbędne dla funkcjonowania organizacji bez konieczności inwestowania w drogi sprzęt.*
  1. Charakterystyka wybranych implementacji chmur obliczeniowych

Charakterystyka najpopularniejszych rozwiązań

### Amazon EC2

### Google AppEngine

### Microsoft Azure

### Heroku

* 1. Aplikacja jako serwis

Integracja z platforma udostepniajaca,

protokól komunikacyjny

Zasady uzyskiwania dostępu do aplikacji,

koszty uzytkowania,

1. Architektura systemu udostępniającego oprogramowanie w chmurze obliczeniowej
   1. Analiza wymagań

Głównym elementem systemu jest Portal udostępniający aplikacje jako usługi. Jego celem jest przechowywanie danych o użytkownikach, zbieranie informacji o wykorzystaniu usług. Udostępnia on interfejs umożliwiający aplikacjom klienckim dostęp do posiadanych danych oraz stanowi punkt integracji dla usługi single sign -on . Do Portalu (dzięki udostępnionemu interfejsowi) podłączają się aplikacje klienckie, które stanowią udostępniane na żądanie użytkownika oprogramowanie biznesowe.

* + 1. Wymagania funkcjonalne

### Portal udostępniający aplikacje

Podstawowe wymagania funkcjonalne:

* Tworzenie, edycja i usuwanie użytkowników i administratorów
* Dodawanie i usuwanie aplikacji udostępnianych w Portalu
* Wyświetlanie listy dostępnych aplikacji wraz z opisem
* Zgłaszanie żądań o dostęp do aplikacji
* Monitorowanie wykorzystania aplikacji przez użytkowników w systemie punktowym
* Wyświetlanie listy aplikacji użytkownika
* Udostępnienie interfejsu umożliwiającego rejestrację aplikacji klienckich w Portalu
* Udostępnienie interfejsu umożliwiającego aplikacjom naliczanie kosztów użytkowania aplikacji,

### Przykładowa aplikacja kliencka 1 – Staff Manager

Aplikacja stanowi wirtualną kartotekę pracowników firmy, która przechowuje podstawowe dane pracownika.

Podstawowe wymagania funkcjonalne:

* Mianowanie użytkownika administratorem aplikacji
* Single sign-on z Portalem
* Dodawanie, edycja oraz usuwanie informacji personalnych o pracownikach – imię, nazwisko, adres zamieszkania, PESEL, NIP, data zatrudnienia, uwagi

### Przykładowa aplikacja kliencka 2 – Warehouse Manager

Aplikacja ma za zadanie uproszczenie zarządzania magazynem – przechowuje informacje o produktach znajdujących się na stanie. Dodatkowo przechowywana jest informacja o ilości i lokalizacji towarów.

Podstawowe wymagania funkcjonalne:

* Mianowanie użytkownika administratorem aplikacji
* Single-sin-on z Portalem
* Dodawanie, edycja oraz usuwanie informacji o produkcie. Są to nazwa, waga, rozmiar opakowania, uwagi oraz jego ilość i aktualne położenie w magazynie
  + 1. Wymagania niefunkcjonalne

### Portal udostępniający aplikacje

Podstawowe wymagania niefunkcjonalne:

* Implementacja z wykorzystaniem Play! Framework
* Wdrożenie na platformę Heroku
* Wykorzystanie bazy danych oferowanej przez platformę

### Aplikacje klienckie (wersja ogólna – podzielić na Staf i Warehouse Manager)

* Implementacja z wykorzystaniem Django/AppEngine
* Implementacja z wykorzystaniem innej chmury obliczeniowej
  1. Opis architektury systemu
     1. Architektura portalu udostępniającego aplikacje jako usługi

### Single Sign – On

* + 1. Architektura aplikacji - usług

1. Implementacja systemu
   1. Wykorzystane technologie
      1. Google AppEngine
      2. Microsoft Azure
      3. Amazon EC2
      4. Heroku
      5. Play!
      6. Spring
   2. Model architektury rozproszonej

Diagram przedsawiajacy portal oraz aplikacje hostowane w rożnych cloudach

* 1. Implementacja portalu udostępniającego aplikacje
  2. Implementacja przykładowych aplikacji - usług

1. Testy systemu

Podsumowanie

# Wykaz skrótów

BDD - Behavior Driven Development

BPMN - Business Process Modeling Notation

BPMS - Bussiness Process Management System

CRM – Customer Relationship Management

ESB – Enterprise Service Bus

GAE – Google App Engine

IaaS – Infrastructure as a Service

PaaS – Platform as a Service

SaaS - Software as a Service

SOA – Service Oriented Architecture

SSO – Single Sign-On

TDD – Test Driven Development

# Spis ilustracji

Rysunek 1. Trudności w integracji systemów

Rysunek 2. Zależności poszczególnych komponentów systemu

Rysunek 3. „Magic Bus”

Rysunek 4. Fazy życia systemu

# Spis tabel

# Spis listingów

Listing 1. Interfejs czytelny z biznesowego punktu widzenia

Listing 2. Interfejs niezrozumiały z biznesowego punktu widzenia

# Bibliografia

|  |  |
| --- | --- |
| [1] | E. Hewitt, Java SOA cookbook, Beijing, Cambridge, Farnham, Köln, Sebastopol, Taipei, Tokyo: O'Reilly, 2009. |
| [2] | M. Rosen, B. Lublinsky, K. T. Smith i M. J. Balcer, Applied SOA, Indianapollis: Wiley Publishing Inc., 2008. |
| [3] | A. Lopez, „Statistical machine translation,” 2008. |
| [4] | a. R. H. G. E. M. Joshua S. Albrech, „Correcting Automatic Translations through Collaborations between MT and Monolingual Target-Language Users,” 2009 . |
| [5] | S. V. Narjes Sharif Razavian, „The Web as a Platform to Build Machine Translation,” 2009. [Online]. |
| [6] | „Workshop on statistical machine translation,” [Online]. Available: http://www.statmt.org/wmt11/. |
| [7] | „Direct Web Remoting,” [Online]. Available: http://directwebremoting.org. |
| [8] | J. D. S. G. W. C. H. D. A. W. Fay Chang, „Bigtable: A Distributed Storage System for Structured Data”.*OSDI 2006.* |
| [9] | „Translation bots for GTalk,” [Online]. Available: http://www.google.com/support/talk/bin/answer.py?answer=89921. |
| [10] | „Tbot for Windows Live Messenger,” [Online]. Available: http://www.microsofttranslator.com/user/bot/. |
| [11] | I. G.-V. F. C. Daniel Ortiz-Martinez, „Online Learning for Interactive Statistical Machine Translation,” [Online]. |
| [12] | P. Koehn, „Enabling Monolingual Translators: Post-Editing vs. Options,” [Online]. |
| [13] | Apache, „HBase,” [Online]. Available: http://wiki.apache.org/hadoop/Hbase. |
| [14] | „The 3rd Asian Conference on Intelligent Information and Database Systems,” [Online]. Available: http://ke.yu.ac.kr/aciids2011/. |
| [15] | International Organization for Standardization, ISO TC37/SC4. |
| [16] | The Gallup Organization, „User language preferences online - Analytical report,” 2011. |
| [17] | M. Kay, „The Proper Place of Men and Machines in Language,” 1980. |
| [18] | J. D. S. G. W. C. H. D. A. W. Fay Chang, „Bigtable: A Distributed Storage System for Structured Data,” 2006. |
| [19] | C. Henderson, „Building Scalable Web Sites,” 2006. |
| [20] | J. Hutchins, „Machine translation: A concise history.,” 2007. |
| [21] | S. R. T. W. W. Z. K. Papineni., „Bleu: a Method for Automatic Evaluation of Machine Translation,” *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, ACL,* 2002. |
| [22] | T. Nomoto, „Multi-Engine Machine Translation with Voted Language Model”. |
| [23] | D. D. Palmer, „Tokenisation and sentence segmentation”. |
| [24] | M. Pytel, „Rozproszone składowanie danych dla potrzeb systemu tłumaczenia iteracyjnego,” 2011. |
| [25] | O. B. F. C. Sergio Barrachina, „Statistical Approaches to Computer-Assisted Translation,” 2008. |
| [26] | Google, „Google Wave,” [Online]. Available: http://wave.google.com/. |
| [27] | Yahoo, „Yahoo Babelfish,” [Online]. Available: http://babelfish.yahoo.com/. |
| [28] | C. Callison-Burch, C. Fordyce, P. Koehn, C. Monz i J. Schroeder, „(Meta-) Evaluation of Machine Translation,” *Proceedings of the Second Workshop on Statistical Machine Translation,* 2007. |
| [29] | C. Callison-Burch, C. Fordyce, P. Koehn, C. Monz i J. Schroeder, „Further Meta-Evaluation of Machine Translation,” *Proceedings of the Third Workshop on Statistical Machine Translation,* 2008. |
| [30] | C. Callison-Burch, P. Koehn, C. Monz i J. Schroeder, Findings of the 2009Workshop on Statistical Machine Translation. |
| [31] | C. Callison-Burch, P. Koehn, C. Monz, K. Peterson, M. Przybocki i O. F. Zaidan, „Findings of the 2010 JointWorkshop on Statistical Machine Translation and Metrics for Machine Translation”.*Proceedings of the Joint 5th Workshop on Statistical Machine Translation and MetricsMATR.* |
| [32] | C. Callison-Burch, P. Koehn i M. Osborne, „Improved Statistical Machine Translation Using Paraphrases in Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL,” 2006. |
| [33] | F. Casacuberta, J. Civera, E. Cubel, A. L. Lagarda, G. Lapalme, E. Macklovitch i E. Vidal, „Human Interaction For High-Quality Machine Translation,” *Communications of the ACM,* 2009. |
| [34] | N. Madnani, N. F. Ayan, P. Resnik i B. J. Dorr, „Using Paraphrases for Parameter Tuning in Statistical Machine Translation,” *Proceedings of the Second Workshop on Statistical Machine Translation,* 2007. |
| [35] | M. Milkowski i J. Lipski, Using SRX standard for sentence segmentation in LanguageTool, 2009. |
| [36] | Y. Chen, M. Jellinghaus, A. Eisele, Y. Zhang, S. Hunsicker, S. Theison, C. Federmann i H. Uszkoreit, „Combining Multi-Engine Translations with Moses,” 2009. |
| [37] | J. Savoy i L. Dolamic, How Effectiveis Google's Translation System in Search. |
| [38] | T. Nguyen, S. Vogel i N. A. Smith, „Nonparametric Word Segmentation for Machine Translation,” 2010. |
| [39] | J. S. Albrech, R. Hwa i G. E. Marai, „Correcting Automatic Translations through Collaborations between MT and Monolingual Target-Language Users,” 2009. |
| [40] | S. Barrachina, F. Casacuberta, E. Cubel, A. Lagarda, J. Tomas, J.-M. Vilar, O. Bender, J. Civera, S. Khadivi, H. Ney i E. Vidal, „Statistical Approaches to Computer-Assisted Translation,” 2008. |
| [41] | E. Gamma, R. Helm, R. Johnson i J. Vlissides, Design Patterns - Elements of reusable Object Oriented Software, Warszawa: Wydawnictwo Nukowo Techniczne, 2005. |
| [42] | R. Gangadharaia, R. Brown i J. Carbonell, „Monolingual Distributional Profiles forWord Substitution in Machine Translation,” *Coling 2010: Poster Volume,* 2010. |
| [43] | P. Resnik, O. Buzek, C. Hu, Y. Kronrod, A. Quinn i B. B. Bederson, „Improving Translation via Targeted Paraphrasing”.*Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing.* |
| [44] | Y. Shinyama, S. Sekine i K. Sudo, „Automatic Paraphrase Acquisition from News Articles,” *Proceedings of HLT 2002, Second International Conference on Human Language Technology Research,* 2002. |
| [45] | P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin i E. Herbst, „Moses: Open Source Toolkit for Statistical Machine Translation,” 2007. |
| [46] | Localization Industry Standards Association, „http://www.lisa.org,” [Online]. |
| [47] | Apache, „Apache CXF,” [Online]. Available: http://cxf.apache.org/. |
| [48] | Apache, „Apache HttpClient,” [Online]. Available: http://hc.apache.org/httpcomponents-client-ga/. |
| [49] | Yahoo, „Yahoo Babelfish,” [Online]. Available: http://babelfish.yahoo.com/. |
| [50] | Google, „Google Gadget,” [Online]. Available: http://code.google.com/apis/wave/extensions/gadgets/guide.html. |
| [51] | Google, „Google Docs,” [Online]. Available: https://docs.google.com/. |
| [52] | Google, „Google Translate,” [Online]. Available: http://translate.google.com. |
| [53] | Google, „Google Translate API for Java,” [Online]. Available: http://code.google.com/p/google-api-translate-java/. |
| [54] | Google, „Google translate REST Api,” [Online]. Available: http://code.google.com/intl/pl/apis/language/translate/v1/getting\_started.html. |
| [55] | iTranslate. [Online]. Available: http://itranslate4.eu/. |
| [56] | R. Matt, „Appfuse 2,” [Online]. Available: http://appfuse.org/display/APF/Home. |
| [57] | Apache, „Apache POI,” [Online]. Available: http://poi.apache.org/. |
| [58] | Apache, „JMeter,” [Online]. Available: http://jakarta.apache.org/jmeter/. |
| [59] | Microsoft, „Microsoft Translator,” [Online]. Available: http://www.microsofttranslator.com/. |
| [60] | Google , „Narzędzia Google dla Tłumaczy,” [Online]. Available: http://translate.google.com/toolkit/. |
| [61] | Localization Industry Standards Association, „Segmentation Rules Exchange,” [Online]. Available: http://www.lisa.org/fileadmin/standards/srx20.html. |
| [62] | Wordpress, [Online]. Available: http://wordpress.com/. |
| [63] | Translators without borders, [Online]. Available: http://tsf.eurotexte.fr/?lang=en. |
| [64] | Springsource, „Spring,” [Online]. Available: http://www.springsource.org/. |
| [65] | Ignite Realtime, „Spark,” [Online]. Available: http://www.igniterealtime.org/projects/spark/index.jsp. |
| [66] | World Wide Web Consortium, „SOAP,” [Online]. Available: http://www.w3.org/TR/soap/. |
| [67] | World Wide Web Consortium, „WSDL,” [Online]. Available: http://www.w3.org/TR/wsdl. |
| [68] | Google , „University Research Program for Google Translate,” [Online]. Available: http://research.google.com/university/translate/. |
| [69] | World Wide Web Consortium, „XPath,” [Online]. Available: http://www.w3schools.com/xpath/. |
| [70] | Ignite Realtime, „Sparkplug Kit,” [Online]. Available: http://www.igniterealtime.org/projects/spark/sparkplug-kit.jsp. |
| [71] | Okapi Framework, [Online]. Available: http://okapi.sourceforge.net/. |
| [72] | XMPP Standards Foundation, „XMPP: Core,” [Online]. Available: http://xmpp.org/rfcs/rfc6120.html. |
| [73] | XMPP Standards Foundation, „XMPP: Instant Messaging and Presence,” [Online]. Available: http://xmpp.org/rfcs/rfc6121.html. |
| [74] | Gazeta Wyborcza, „StartupFest,” [Online]. Available: http://startupfest.pl. |
| [75] | Ignite Realtime, „Smack API,” [Online]. Available: www.igniterealtime.org/projects/smack/. |
| [76] | Java Content Repository, [Online]. Available: http://www.day.com/specs/jcr/1.0/. |
| [77] | World Wide Web Consortium, „SOAP,” [Online]. Available: http://www.w3.org/TR/soap/. |
| [78] | Google, „The University Research Program for Google Translate,” [Online]. Available: http://research.google.com/university/translate/docs.html. |
| [79] | Google, „Google Wave Robot,” [Online]. Available: http://code.google.com/apis/wave/extensions/robots/. |
| [80] | XTRF, [Online]. Available: http://pl.xtrf.eu/. |
| [81] | iTranslate4.eu, [Online]. Available: http://iTranslate4.eu. |
| [82] | Organization for the Advancement of Structured Information Standards, „XLIFF specification,” [Online]. Available: http://www.oasis-open.org/committees/xliff/documents/cs-xliff-core-1.1-20031031.htm. |
| [83] | A. Potępa, P. Płonka, M. Pytel i D. Radziszowski, „Iterative Translation by Monolinguists - tests of the new approach,” *Lecture Notes in Computer Science,* 2011. |
| [84] | J.-C. Wu, T. Chuang, W.-C. Shei i J. Chang, „Subsentential Translation Memory for Computer Assisted Writing and Translation,” 2004. |

# Spis ilustracji i tabel

# Aneksy