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](#_Toc327395321)

[Struktura pracy 5](#_Toc327395322)

[1. Koncepcja usługi 6](#_Toc327395323)

[1.1 Charakterystyka nowoczesnej architektury systemu informatycznego 6](#_Toc327395324)

[1.2 Serwisy w nowoczesnych systemach informatycznych 7](#_Toc327395325)

[1.3 Definicja Serwisu 7](#_Toc327395326)

[1.4 Architektura zorientowana na usługi (service oriented architecture – SOA) 8](#_Toc327395327)

[SOA - koncepcja 8](#_Toc327395328)

[Serwisy 8](#_Toc327395329)

[Interoperacyjność 9](#_Toc327395330)

[Luźne powiązania 9](#_Toc327395331)

[Enterprise Service Bus 10](#_Toc327395332)

[Zalety podejścia SOA 10](#_Toc327395333)

[Ograniczenia podejścia SOA 10](#_Toc327395334)

[2. Architektura chmury obliczeniowej 11](#_Toc327395335)

[2.1 Koncepcja chmury obliczeniowej 11](#_Toc327395336)

[2.2 Modele chmury obliczeniowej 11](#_Toc327395337)

[Kolokacja 11](#_Toc327395338)

[Oprogramowanie jako usługa 12](#_Toc327395339)

[Platforma jako usługa 12](#_Toc327395340)

[Infrastruktura jako usługa 12](#_Toc327395341)

[Komunikacja jako usługa 12](#_Toc327395342)

[2.3 Charakterystyka wybranych implementacji chmur obliczeniowych 13](#_Toc327395343)

[Amazon EC2 13](#_Toc327395344)

[Google AppEngine 13](#_Toc327395345)

[Microsoft Azure 13](#_Toc327395346)

[Heroku 13](#_Toc327395347)

[2.4 Aplikacja jako serwis 13](#_Toc327395348)

[3. Architektura systemu udostępniającego oprogramowanie w chmurze obliczeniowej 13](#_Toc327395349)

[3.1 Analiza wymagań 13](#_Toc327395350)

[3.1.1 Wymagania funkcjonalne 13](#_Toc327395351)

[Portal udostępniający aplikacje 13](#_Toc327395352)

[Przykładowa aplikacja kliencka 1 – Staff Manager 14](#_Toc327395353)

[Przykładowa aplikacja kliencka 2 – Warehouse Manager 14](#_Toc327395354)

[3.1.2 Wymagania niefunkcjonalne 15](#_Toc327395355)

[Portal udostępniający aplikacje 15](#_Toc327395356)

[Aplikacje klienckie (wersja ogólna – podzielić na Staf i Warehouse Manager) 15](#_Toc327395357)

[3.2 Opis architektury systemu 16](#_Toc327395358)

[3.2.1 Architektura portalu udostępniającego aplikacje jako usługi 16](#_Toc327395359)

[Single Sign – On 16](#_Toc327395360)

[3.2.2 Architektura aplikacji - usług 16](#_Toc327395361)

[4. Implementacja systemu 16](#_Toc327395362)

[4.1 Wykorzystane technologie 16](#_Toc327395363)

[4.1.1 Google AppEngine 16](#_Toc327395364)

[4.1.2 Microsoft Azure 16](#_Toc327395365)

[4.1.3 Amazon EC2 16](#_Toc327395366)

[4.1.4 Heroku 16](#_Toc327395367)

[4.1.5 Play! 16](#_Toc327395368)

[4.1.6 Spring 16](#_Toc327395369)

[4.2 Model architektury rozproszonej 16](#_Toc327395370)

[4.3 Implementacja portalu udostępniającego aplikacje 16](#_Toc327395371)

[4.4 Implementacja przykładowych aplikacji - usług 17](#_Toc327395372)

[5. Testy systemu 17](#_Toc327395373)

[Podsumowanie 17](#_Toc327395374)

[Wykaz skrótów 18](#_Toc327395375)

[Spis ilustracji 18](#_Toc327395376)

[Spis tabel 18](#_Toc327395377)

[Spis listingów 18](#_Toc327395378)

[Bibliografia 19](#_Toc327395379)

[Spis ilustracji i tabel 21](#_Toc327395380)

[Aneksy 21](#_Toc327395381)

# 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 możliwości ich zastosowania we współczesnych systemach informatycznych.

Kolejny rozdział poświęcony jest chmurom obliczeniowych, zawiera on także oprócz wyjaśnienia pojęcia chmury omówienie różnych jej modeli wraz z prezentacją istniejących implementacji wraz z przykładami ich wykorzystania.

Trzeci rozdział poświęcony jest architekturze przykładowej aplikacji. Zawiera opis wymagań funkcjonalnych i niefunkcjonalnych. Przedstawia schemat najważniejszych jej elementów oraz prezentuje zastosowane technologie.

Podsumowanie opisuje dalsze możliwości rozwoju projektu oraz potencjalne problemy. Na końcu przedstawione są także wnioski powstałem podczas pracy nad systemem.

1. Koncepcja usługi

* 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 mogą działać w rożnych środowiskach czy lokalizacjach. Usługi są wykorzystywane w miarę potrzeb – w chwili gdy nastąpi zdarzenie wykonywana jest stosowna akcja przez zadany serwis. Zastosowanie takiej architektury umożliwia łatwiejsze utrzymanie systemu oraz upraszcza jego rozwój czy rekonfigurację.

* 1. Definicja Serwisu

Serwis(usługa) jest to element oprogramowania mogący działać niezależnie, posiadający zdefiniowany interfejs, który określa i udostępnia realizowane funkcjonalności. Szczegóły implementacyjne są nieistotne i niewidoczne dla Klienta wykorzystującego dany serwis. Zastosowanie interfejsów umożliwia wymianę poszczególnych serwisów bez wpływania na całość systemu.

Ważnym elementem każdego serwisu jest implementacja protokołu komunikacyjnego, przy użyciu którego możliwa jest wzajemna komunikacja serwisów. Bardzo często wykorzystuje się lekkie formaty wymiany danych oparte o protokół HTTP, są to SAOP oraz JSON. Ich podstawową zaletą jest niezależność od wykorzystywanych języków programowania czy platform sprzętowych. Dane są przesyłane w formacie tekstowym przez co są czytelne dla człowieka (ułatwienie w debugowaniu), jednakże powoduje to pewne narzuty na komunikację, co należy wziąć pod uwagę zapewniając odpowiednio dużą ziarnistość serwisów. Wymienione formaty danych nie nadają się do przesyłania treści multimedialnych ze względu na ich tekstowych charakter powodujący nadmierne narzuty przy wielkich ilościach danych. W tych przypadkach należy wprowadzić binarny protokół przystosowany do przesyłania danych multimedialnych. Kontrola nad przesyłaniem tych danych może być efektywnie prowadzona przez tekstowy protokół.

* 1. Architektura zorientowana na usługi (service oriented architecture – SOA)

Architektura zorientowana na usługi jest to paradygmat tworzenia oprogramowania jako grupy zorkiestrowanych, współpracujących ze sobą usług, które współpracując ze sobą spełniają wymagania użytkownika. Istotne jest słabe powiązanie pomiędzy poszczególnymi usługami – umożliwia to wielokrotne użycie serwisów. 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).

Tworzenie aplikacji zgodnych z paradygmatem SOA umożliwia integrację i rozbudowę systemów w każdym obszarze działania instytucji Klienta, od aplikacji tworzonych na potrzeby poszczególnych działów do tych tworzonych dla potrzeb całej korporacji.

## SOA - koncepcja

Paradygmat SOA możemy scharakteryzować przy pomocy następujących słów kluczy:

* Serwisy
* Interoperacyjność
* Luźne powiązania

## Serwisy

Przez serwis w SOA rozumiemy reprezentację funkcjonalności biznesowej, która jest określona przez interfejs. Serwisy powinny być tak stworzone by były rozumiane z punktu widzenia biznesowego. Listing 1 prezentuje interfejs serwisu zaprojektowany w sposób czytelny dla biznesu, z kolei listing 2 prezentuje jego przeciwieństwo.

CustomerService{

readCustomer(id);

deleteCustomer(id);

createCustomer(name, address);

changeCustomerAddress(id, newAdddress);

}

Listing 1. Interfejs czytelny z biznesowego punktu widzenia

CustomerService{

doOperation(

id,

operation, //create, delete, update

address

)

}

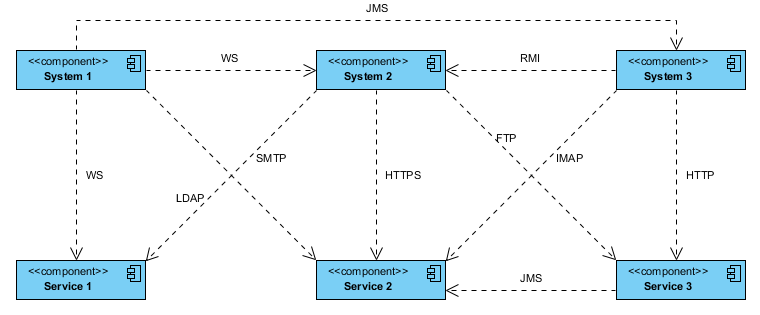
Listing 2. Interfejs niezrozumiały z biznesowego punktu widzenia

## Interoperacyjność

Złożoność organizacji biznesowych wymaga by wykorzystywane przez nie różne systemy komputerowe miały możliwość współpracy ze sobą. Problem współpracy między systemami narasta coraz bardziej wraz z pojawianiem się nowych modułów w poszczególnych systemach (systemy z czasem stają się coraz bardziej wrażliwe na integrację). Jest to spowodowane coraz większo ilością wykorzystywanych protokołów, które są często niekompatybilne ze sobą. Idealnym rozwiązaniem byłoby zastosowanie tylko takich komponentów, które są stworzone w oparciu o standardowe technologie (np. WebServices), jednakże w przypadku dużych organizacji jest to praktycznie niewykonalne. Główne powody takiego stanu rzeczy to:

* Zależność od systemów zewnętrznych, na które nie mamy wpływu
* Brak uwzględnienia potrzeby interoperacyjności w już istniejącym systemie
* Zaszłości technologiczne – konieczność współpracy z systemami odziedziczonymi

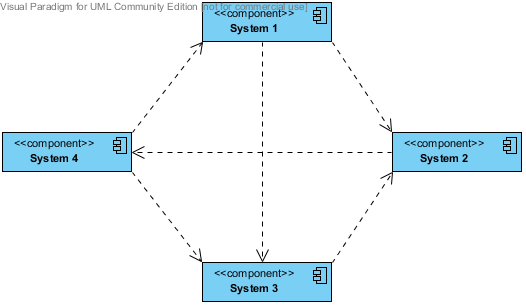
Jednym z głównych wymagań SOA jest uwzględnienie interoperacyjności już od etapu projektowania systemu. Główną trudnością w integracji jest mnogość wykorzystywanych rozwiązań komunikacyjnych co możemy zobaczyć na rysunku 1. Dodanie do takiego konglomeratu systemów kolejnego elementu wymaga skorzystania z wielu rozwiązań co stwarza problemy w czasie implementacji jak i utrzymania systemu.



Rysunek 1. Trudności w integracji systemów

## Luźne powiązania

Termin *loose coupling* bardzo często pojawia się w kontekście większych systemów, w których bardzo często występuje zbyt duży stopień zależności (rysunek 2) co utrudnia ich rozwój oraz integrację. Najczęściej przyczyną takiego stanu rzeczy jest niedostateczne uwzględnienie (na etapie projektowania) kwestii skalowalności czy uniwersalności systemu bądź jego komponentów. Jak wiadomo nie da się zupełnie wyeliminować powiązań pomiędzy poszczególnymi składnikami oprogramowania, jednakże poniesienie kosztów w celu minimalizacji wzajemnych zależności przynosi zyski podczas dalszego rozwoju czy utrzymania systemu.

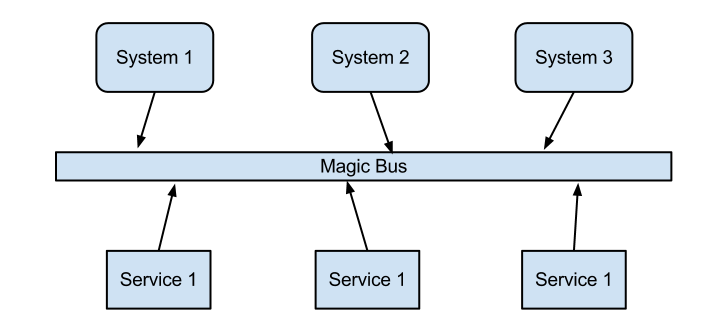


Rysunek 2. Zależności poszczególnych komponentów systemu

## Enterprise Service Bus

Enterprise Service Bus jest to dodatkowa warstwa abstrakcji stosowana w podejściu SOA. Jej nazwa (bus) została zainspirowana sprzętowymi magistralami będącymi elementami każdego komputera. Pełni rolę „magic bus” ułatwiającej komunikacje pomiędzy poszczególnymi komponentami. Celem ESB jest zarządzanie serwisami w trakcie pracy systemu, umożliwia dynamicznie odłączanie i przyłączanie serwisów w trakcie jego działania oraz zajmuje się kontrolą wersji poszczególnych komponentów. Dodatkowo bierze udział w routowaniu oraz kontroli przepływu wiadomości pomiędzy serwisami.

<rozwinąć> BPML



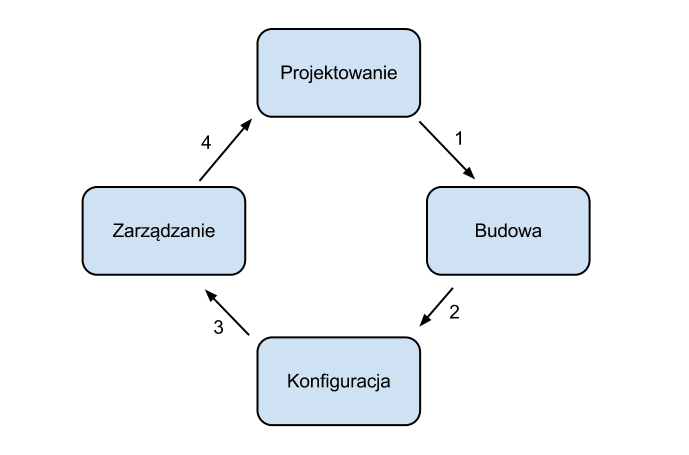
Rysunek 3. „Magic Bus”

## Cykl życia systemu opartego o paradygmat SOA

Jako cykl życia aplikacji rozumiem 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 rysunek 3. Jak widać przedstawiony cykl życia nie rożni się on w znaczący sposób od cyklów życia systemów nie tworzonych w oparciu o SOA.



Rysunek 4. Fazy życia systemu

## 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 rekonfigurowalności 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.

## 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

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

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

# 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

* Service-Oriented Cloud Computing Architecture <http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F5498003%2F5501429%2F05501650.pdf%3Farnumber%3D5501650&authDecision=-203>
* Cloud computing — Issues, research and implementations <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4588381>
* Automatyczna skalowalność aplikacji na platformach Amazon WS i Google App Engine <http://www.ki.agh.edu.pl/sites/default/files/publikacje/164/amazon.pdf>
* SaaAS - The mobile agent based service for cloud computing in internet environment <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5582438>
* A Model of Scheduling Optimizing for Cloud Computing Resource Sevices Based on Buffer-pool Agent <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5576181>
* Service Replication Strategies with MapReduce in Clouds <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5741385>
* Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing <http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F71%2F5756300%2F05719609.pdf%3Farnumber%3D5719609&authDecision=-203>
* Ubiquitous cloud: Managing service resources for adaptive ubiquitous computing <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5766853>
* From outsourcing to Cloud computing: Evolution of IT services <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5996009>
* Service-Oriented Computing and Cloud Computing: Challenges and Opportunities <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5617062>
* <http://www.cloudtweaks.com/2011/02/cloud-computing-for-dummies-saas-paas-iaas-and-all-that-was/>
* Google App Engine Campfire One Transcript <http://code.google.com/intl/pl-PL/appengine/articles/cf1-text.html>
* Amazon EC2 Overview <http://aws.amazon.com/ec2/>
* Windows Azure <http://www.microsoft.com/windowsazure/features/>
* iCloud <http://en.wikipedia.org/wiki/ICloud>
* Nimbus <http://www.nimbusproject.org/>
* Eucalyptus <http://www.eucalyptus.com/>
* Eucalyptus System for Clouds <http://fatcat.ftj.agh.edu.pl/~i7szczep/koala.pdf>
* OpenNebula <http://www.opennebula.org/>
* OpenStack [http://openstack.org](http://openstack.org/)
* VM WebSpere<http://www.ki.agh.edu.pl/technologie/vmware-vsphere>
* SOA czyli service oriented architecture, Software Developer’s Journal 10/2009

# Spis ilustracji i tabel

# Aneksy