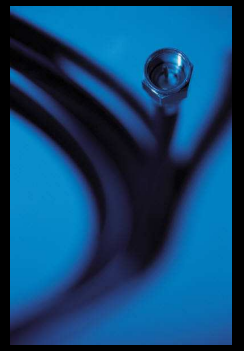
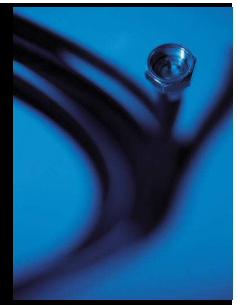


Inżynieria Systemów Rozproszonych

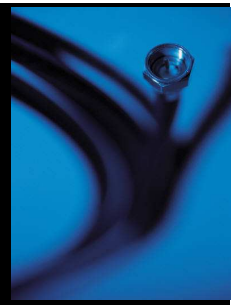
Sommerville, Ian: Software Engineering, edycja 9, rozdział 18

Zagadnienia



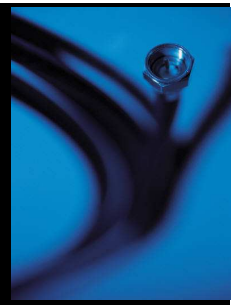
- Problematyka systemów rozproszonych
  - Co jest istotne w kontekście projektowania.
- Przetwarzanie typu klient – serwer
  - Jako podstawowy model systemu rozproszonego
- Wzorce architektury systemów rozproszonych
  - Wzorzec - zastosowanie
- Oprogramowanie jako usługa
  - Internetowy dostęp do systemu

Systemy rozproszone



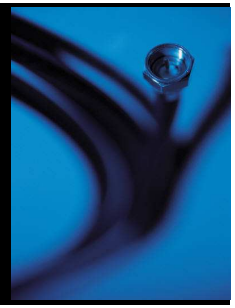
- Zbiór niezależnych komputerów, które z punktu widzenia użytkownika postrzegane są jako pojedynczy, spójny system.
- Przetwarzanie informacji jest rozproszone pomiędzy zespół komputerów – nie ogranicza się do pojedynczej jednostki.
- W praktyce wszystkie duże systemy komputerowe są systemami rozproszonymi
  - Dlatego inżynieria systemów rozproszonych jest bardzo ważna dla systemów przemysłowych.

Charakterystyka systemów rozproszonych



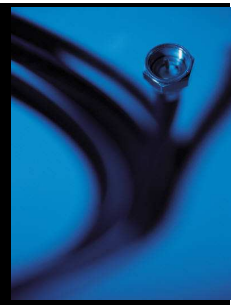
- Współdzielenie zasobów
  - Sprzętu i oprogramowania.
- Otwartość
  - Wykorzystanie sprzętu i oprogramowania pochodzącego od różnych dostawców.
- Współbieżność
  - W celu zwiększenia wydajności.
- Skalowalność
  - Zwiększenie przepustowości przez dodanie nowych zasobów.
- Tolerancja na błędy
  - Zdolność do kontynuowania działań po wystąpieniu błędu.

Problemy systemów rozproszonych



- Systemy rozproszone są bardziej skomplikowane niż te, które działają na pojedynczym procesorze.
- Złożoność związana jest z potrzebą niezależnego zarządzania częściami systemu
- Nie istnieje jeden organ odpowiedzialny za system - odgórne sterowanie jest niemożliwe.

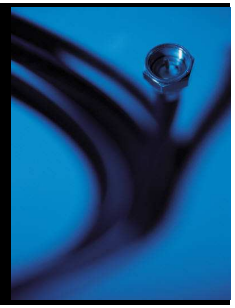
Problemy projektowe



- Przezroczystość (ang. Transparency)
- Otwartość (ang. Openness)
- Skalowalność (ang. Scalability)
- Zabezpieczenie (ang. Security)
- Jakość usługi (ang. Quality of service)
- Zarządzanie awariami (ang. Failure management)

Problemy projektowe (1 z 6)

Przezroczystość (ang. Transparency)

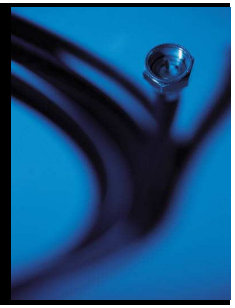


W jakim zakresie system rozproszony powinien być postrzegany (przez jego użytkowników) jako system pojedynczy?

- W wersji idealnej użytkownikom nie powinna być potrzebna świadomość, że system jest rozproszony.
- W praktyce jest to niemożliwe
  - Części systemu są zarządzane niezależnie, pojawiają się opóźnienia,
  - Często lepiej uświadomić użytkownika – bo będzie mógł poradzić sobie z problemem
- Aby osiągnąć przezroczystość:
  - zasoby powinny być abstrakcyjne
  - adresowanie powinno być logiczne
  - za mapowanie pomiędzy reprezentacją logiczną a fizyczną odpowiedzialna jest warstwa pośrednia (middleware).

Problemy projektowe (2 z 6)

Otwartość (ang. Openness)

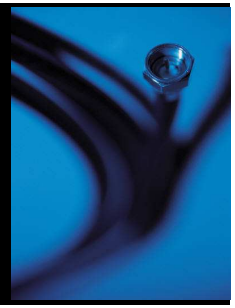


Czy system powinien być projektowany z uwzględnieniem standardowych protokołów wspierających interoperacyjność?

- Otwarty system rozproszony to system, który został zbudowany zgodnie z ogólnie przyjętymi standardami.
- Komponenty dowolnego dostawcy mogą być zintegrowane z systemem i współpracować z pozostałymi.
- Implikacją otwartości jest możliwość niezależnego rozwoju w dowolnym języku programowania. Wystarczy, że komponent jest zgodny ze standardem.
- Przykłady standardów otwartych:
  - CORBA
  - Usługi Internetowe - Web services standards
  - Architektura zorientowana usługowo - Service-oriented architectures.

Problemy projektowe (3 z 6)

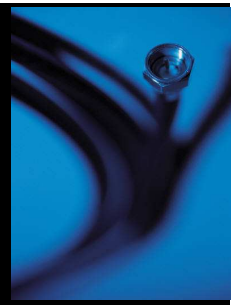
Skalowalność (ang. Scaleability)



Jak system powinien być zbudowany, żeby można go było skalować?

- Skalowalność systemu informuje o jego zdolności do dostarczania usług, o odpowiedniej jakości, w sytuacji gdy wzrasta liczbą żądań
  - ***Rozmiar***
    - Powinna istnieć możliwość zwiększania zasobów systemu w odpowiedzi na zwiększającą się liczbę jego użytkowników.
  - ***Rozproszenie***
    - Powinno być możliwe rozpraszanie geograficzne komponentów systemu bez zmniejszania wydajności.
  - ***Zarządzalność***
    - Powinno być możliwe zarządzanie systemem, zwiększającym swój rozmiar, nawet jeżeli części systemu fizycznie znajdują się w niezależnych organizacjach.
- Odmiany skalowania
  - scaling-up - zwiększanie mocy systemu
  - scaling-out - zwiększanie liczby instancji systemu.

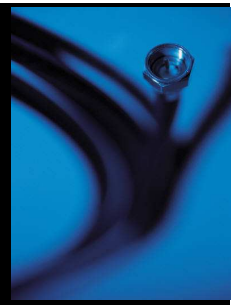
Problemy projektowe (4 z 6)
Zabezpieczenie (ang. Security)



W jaki sposób zdefiniować użyteczną politykę zabezpieczeń i jak ją zaimplementować?

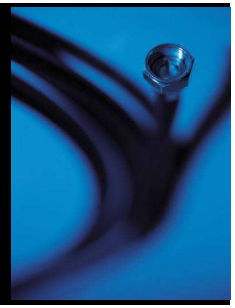
- Rozproszony system jest narażony na dużo więcej rodzajów ataków w porównaniu z systemem scentralizowanym.
- Wystarczy, że sukcesem zakończy się atak tylko na część systemu, żeby otworzyć możliwość dostępu do innych jego elementów.
- Jeżeli części systemu są w posiadaniu różnych organizacji, które mają niekompatybilne polityki zabezpieczeń – może to prowadzić do trudności w zabezpieczaniu systemu.

Rodzaje ataków



- Rodzaje ataków, przed którymi musi bronić się system rozproszony:
  - **Przechwycenie** (ang. Inteception)– komunikacja pomiędzy elementami systemu zostaje przechwycona przez atakującego w taki sposób, że następuje utrata poufności.
  - **Przerwanie** – usługi systemu są atakowane w taki sposób, że nie może ich on dostarczać w oczekiwany sposób.
    - Odmowa usługi (ang. Denial of service) – atak polegający na bombardowaniu usługi zgłoszeniami. Zajęty przetwarzaniem system nie jest w stanie przetwarzać prawdziwych zgłoszeń.
  - **Modyfikacja** – dane i/lub usługi systemu są zmienione przez atakującego.
  - **Fabrykacja** – atakujący generuje informacje, która nie powinna istnieć i wykorzystuje ją następnie do uzyskania uprawnień.

Problemy projektowe (5 z 6)
Jakość usługi (ang. Quality of Service - QoS)



Jak wyspecyfikować jakość usługi?

- Jakość usługi oferowana przez rozproszony system odzwierciedla jego zdolność dostarczania usług:
  - Niezawodnie
  - Z czasem odpowiedzi oraz przepustowością, która jest akceptowalna przez użytkowników.
- QoS jest krytyczne w przypadku systemów związanych z danymi przetwarzanymi w czasie rzeczywistym (np. strumienie audio i video).
  - Brak odpowiedniego poziomu QoS może spowodować degradację sygnału do poziomu niwelującego zrozumiałość informacji.

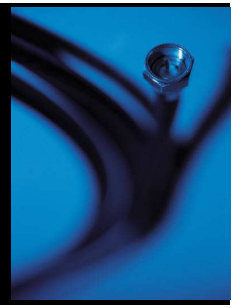
Problemy projektowe (6 z 6)
Zarządzanie awariami (ang. Failure management)

Jak rozpoznawać i naprawiać awarie systemu?

- W systemie rozproszonym nie da się uniknąć uszkodzeń – system musi być zaprojektowany w taki sposób, aby był elastyczny w tym zakresie.
  "You know that you have a distributed system when the crash of a system that you've never heard of stops you getting any work done."

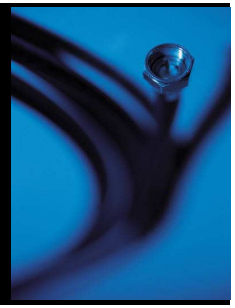
- System rozproszony powinien posiadać mechanizmy pozwalające na wykrycie uszkodzenia komponentu i kontynuację działania dostarczając tyle usług ile jest w stanie, mimo awarii. Powinien również automatycznie się odtwarzać po awarii.

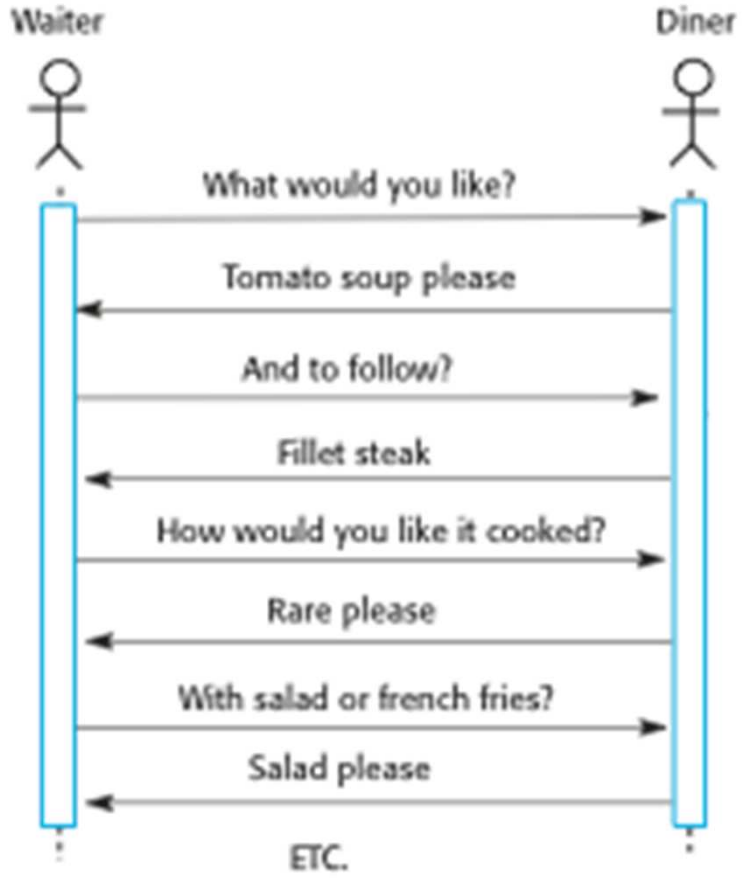
Modele interakcji



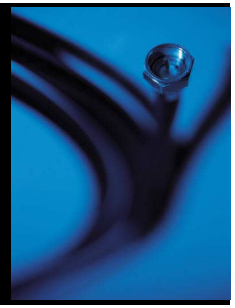
- Podstawowe typy interakcji pomiędzy komponentami w systemie rozproszonym:
  - **Komunikacja proceduralna**
    - Jeden komputer wywołuje znaną mu usługę i (zazwyczaj) oczekuje na odpowiedź.
  - **Interakcja oparta na komunikatach** – jednostka wysyła informacje o tym co jest potrzebne do innej jednostki. Oczekiwanie na odpowiedź nie jest wymagane.

Komunikacja proceduralna pomiędzy klientem a kelnerem



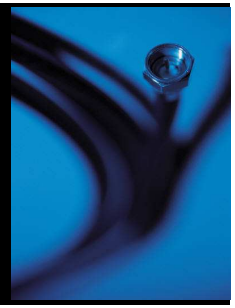


Interakcja oparta na komunikatach pomiędzy kelnerem a kuchnią



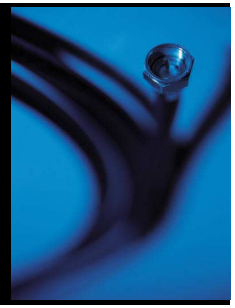
```
<starter>
        <dish name = “soup” type = “tomato” />
        <dish name = “soup” type = “fish” />
        <dish name = “pigeon salad” />
</starter>
<main course>
        <dish name = “steak” type = “sirloin” cooking = “medium” />
        <dish name = “steak” type = “fillet” cooking = “rare” />
        <dish name = “sea bass”>
</main>
<accompaniment>
        <dish name = “french fries” portions = “2” />
        <dish name = “salad” portions = “1” />
</accompaniment>
```

Zdalne wywołanie procedury (ang. Remote procedure call - RPC)



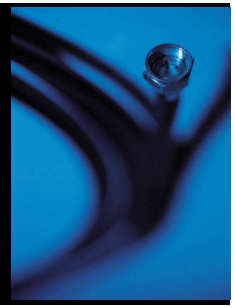
- Komunikacja proceduralna w systemie rozproszonym implementowana jest za pomocą zdalnych wywołań procedur.
- W RPC jeden z komponentów wywołuje inny komponent w taki sam sposób jakby wywoływał procedurę lokalną. Warstwa pośrednia przechwytuje wywołanie i przekazuje je do zdalnego komponentu.
- Zdalny komponent wykonuje obliczenia i, za pośrednictwem warstwy pośredniej, zwraca rezultat wywołującemu.
- Wymagania RPC:
  - wywołujący i wołany muszą być dostępni w czasie komunikacji.
  - wywołujący i wołany muszą wiedzieć jak się ze sobą komunikować.

Przekazywanie komunikatów



- Jeden komponent tworzy komunikat opisujący usługę do wykonania przez inny komponent.
- Za pośrednictwem warstwy pośredniej komunikat wysyłany jest do odbiorcy.
- Odbiorca parsuje komunikat, wykonuje żądane obliczenia. Wysłanie rezultatów obliczeń do odbiorcy wymaga utworzenia komunikatu.
- Nie jest wymagane aby wysyłający i odbiorca mieli świadomość swojego istnienia. Komunikacja odbywa się za pośrednictwem warstwy pośredniej.

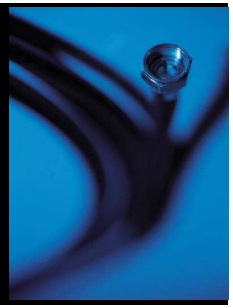
Warstwa pośrednia (Middleware)

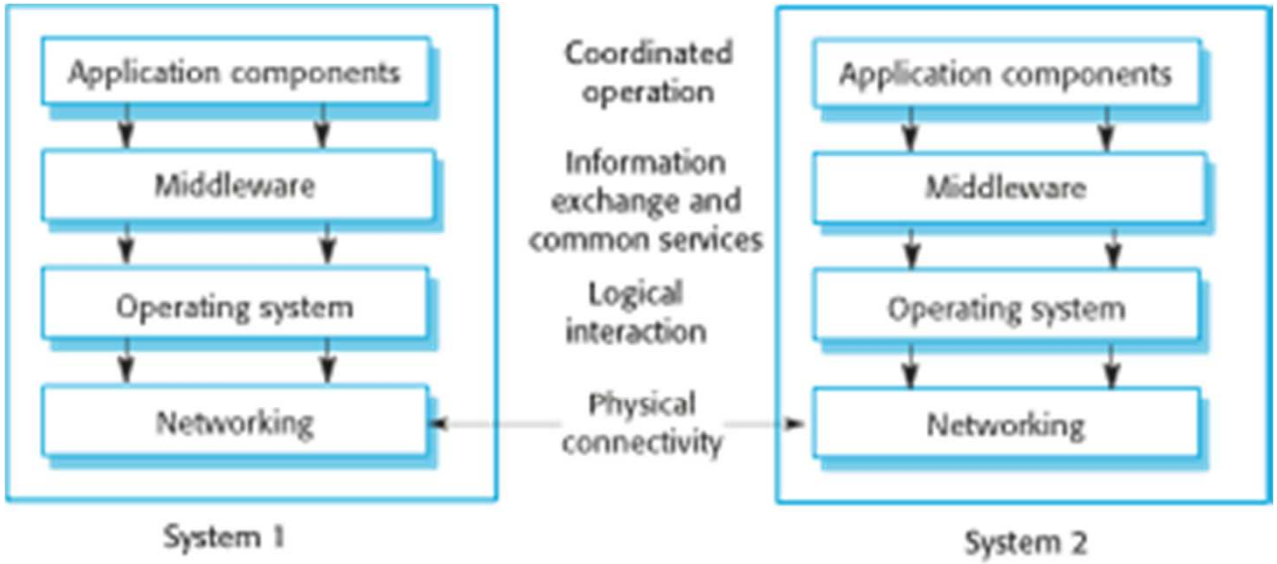


- Komponenty systemu rozproszonego mogą być realizowane z wykorzystaniem różnych języków programowania i mogą być wykonywane na różnych typach procesorów. Również model danych i protokoły komunikacyjne mogą być różne.

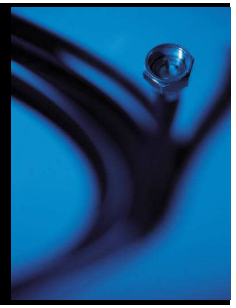
- Warstwa pośrednia to oprogramowanie, które obsługuje tą heterogeniczność aby części systemu mogły się ze sobą komunikować.

Warstwa pośrednia w systemie rozproszonym



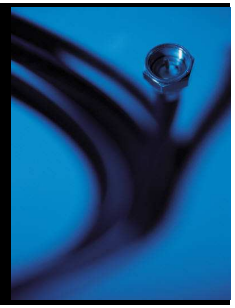


Wsparcie ze strony warstwy pośredniej



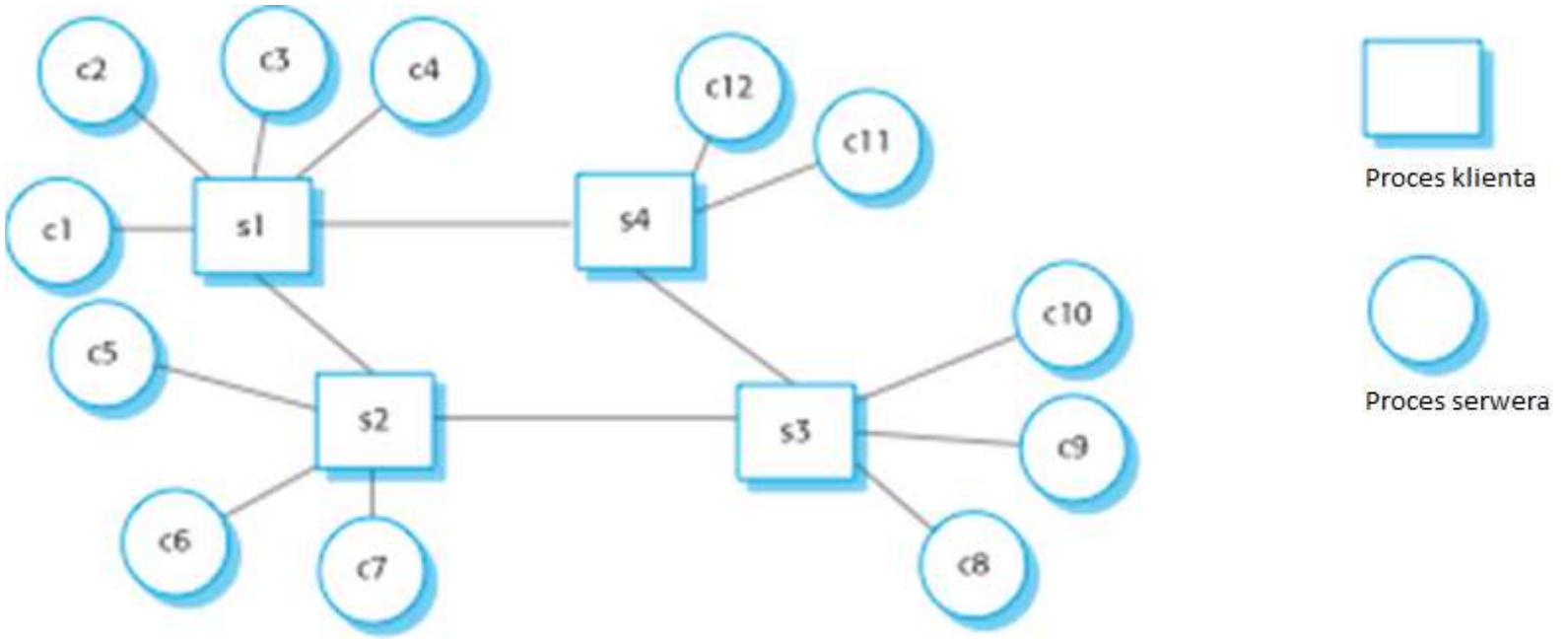
- **Koordynacja interakcji** pomiędzy różnymi komponentami systemu
  - Przezroczystość lokacji – komponenty nie musza znać fizycznej lokacji pozostałych elementów systemu.
- **Świadczenie wspólnych usług**. Warstwa pośrednia udostępnia implementacje usług, które mogą być wymagane przez komponenty systemu
  - Są to zazwyczaj usługi związane z inter-operacyjnością, udostępnianiem usług.

Przetwarzanie typu klient-serwer

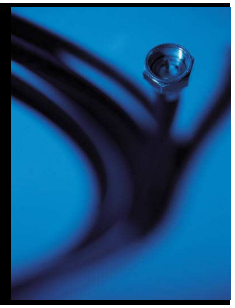


- Systemy rozproszone dostępne za pośrednictwem Internetu są zazwyczaj systemami typu klient-serwer.
- W tego typu systemach użytkownik komunikuje się z aplikacją działającą na lokalnym komputerze (np. przeglądarką internetową). Aplikacja ta komunikuje się z programem działającym za zdalnym komputerze (np. serwerze webowym).
- Zdalny komputer udostępnia usługi dostępne dla klientów.

Interakcja typu klient-serwer



Architektura warstwowa dla aplikacji typu klient-serwer



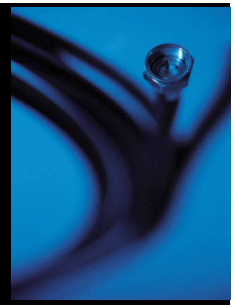
Warstwa prezentacji

Warstwa zarządzania danymi

Warstwa aplikacji

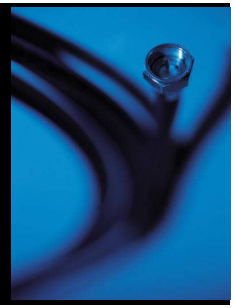
Warstwa bazy danych

Wzorce architektoniczne



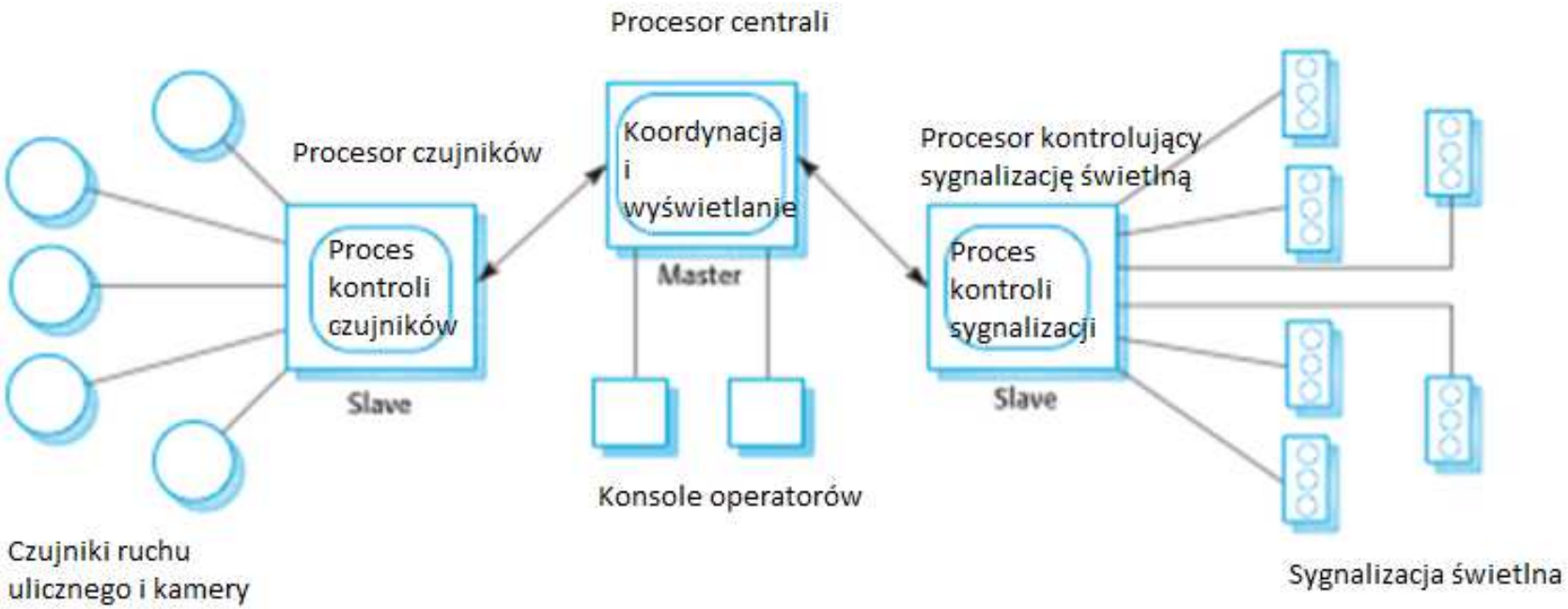
- Do najpopularniejszych wzorców architektonicznych aplikacji rozproszonych należą:
  - Architektura *Master-slave*.
  - Dwuwarstwowa architektura typu klient serwer
  - Wielowarstwowa architektura typu klient-serwer
  - Architektura komponentów rozproszonych
  - Architektura *peer-to-peer*,

Architektura Master-slave

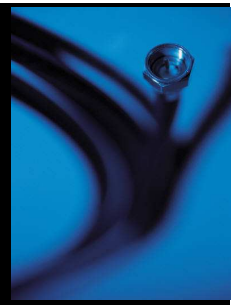


- Zastosowanie:
  - Systemy czasu rzeczywistego, gdzie odrębne procesory powiązane są ze zbieraniem danych ze środowiska, inne za przetwarzanie a jeszcze inne za zarządzanie sygnałami
- Proces ‘master’ jest zazwyczaj odpowiedzialny za obliczenia, koordynację, komunikację i kontrolę procesów „slave”.
- Procesy ‘Slave’ dedykowane są specyficznym akcjom (np. zbieranie danych z czujników).

System zarządzania ruchem ulicznym w architekturze master-slave

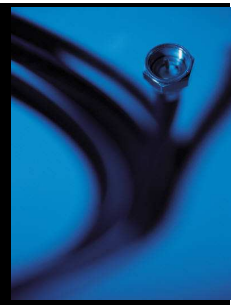


Dwuwarstwowa architektura typu klient-serwer



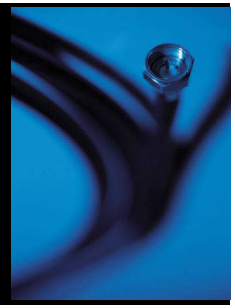
- System jest implementowany jako pojedynczy (logiczny) serwer i niezdefiniowana liczba klientów korzystających z usługi.
  - **Cienki klient** (Thin-client) – warstwa prezentacji implementowana na kliencie a pozostałe implementowane na serwerze.
  - **Gruby klient** (Fat-client) – oprócz warstwy prezentacji klient implementuje wybrane lub wszystkie elementy warstwy aplikacji. Warstwa zarządzania danymi oraz warstwa bazy danych znajduje się po stronie serwera.
- Zastosowanie: Proste systemy klient-serwer oraz systemy, gdzie zabezpieczenie wymaga centralizacji

Model cienkiego klienta



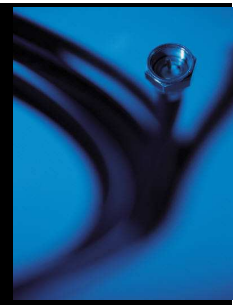
- Wykorzystywany w sytuacji, kiedy przeprowadza się migrację istniejącego systemu do architektury klient-serwer.
  - Istniejący (spadkowy) system działa jako serwer . Interfejs graficzny implementowany jest po stronie klienta.
- Podstawową wadą tego rozwiązania jest duże obciążenie sieci oraz serwera.

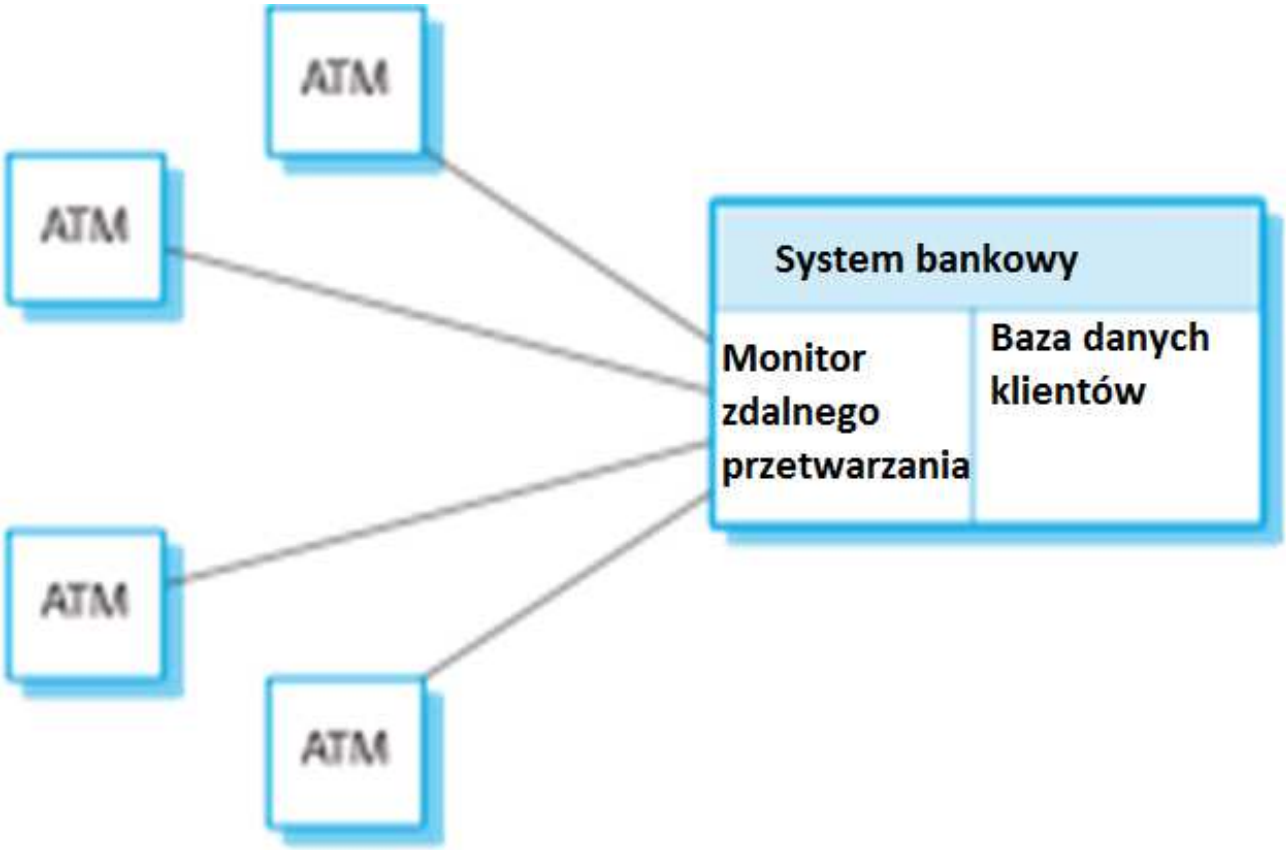
Model grubego klienta



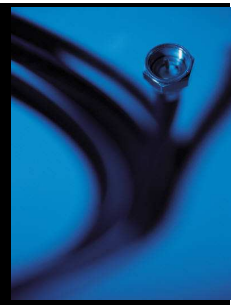
- Więcej przetwarzania przeniesione jest na stronę klienta – aplikacja jest częściowo wykonywana lokalnie.
- Bardziej adekwatne dla nowych systemów klient – serwer, gdzie możliwości klienta są znane a priori.
- Model bardziej skomplikowany niż cienki klient – szczególnie w kontekście zarządzania. Nowe wersje aplikacji muszą być instalowane na kliencie.

Architektura grubego klienta dla systemu bankomatu



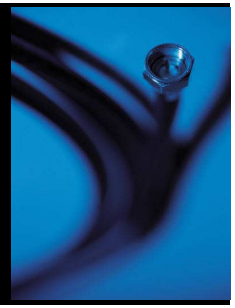


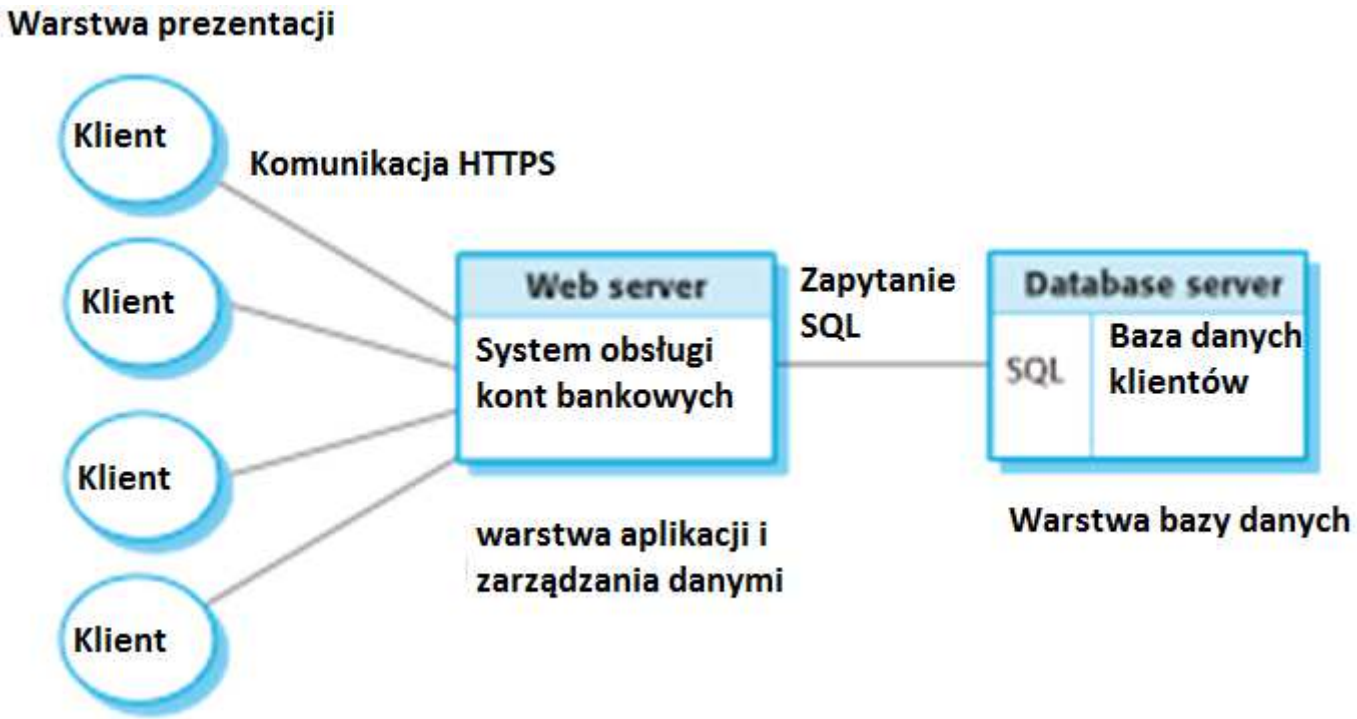
Wielowarstwowa architektura typu klient-serwer



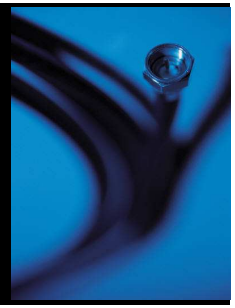
- Różne warstwy systemu (prezentacji, aplikacji, przetwarzania danych i bazy danych) są odrębnymi procesami, które mogą być wykonywane na niezależnych procesorach.
- Celem jest uniknięcie problemów ze skalowalnością i wydajnością (cienki klient w architekturze dwuwarstwowej) lub zarządzaniem systemem (gruby klient).
- Zastosowanie : Duża liczba transakcji, którą musi obsługiwać serwer.

Trójwarstwowa architektura dla systemu bankowości Internetowej



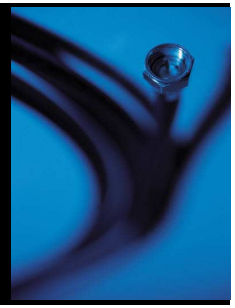


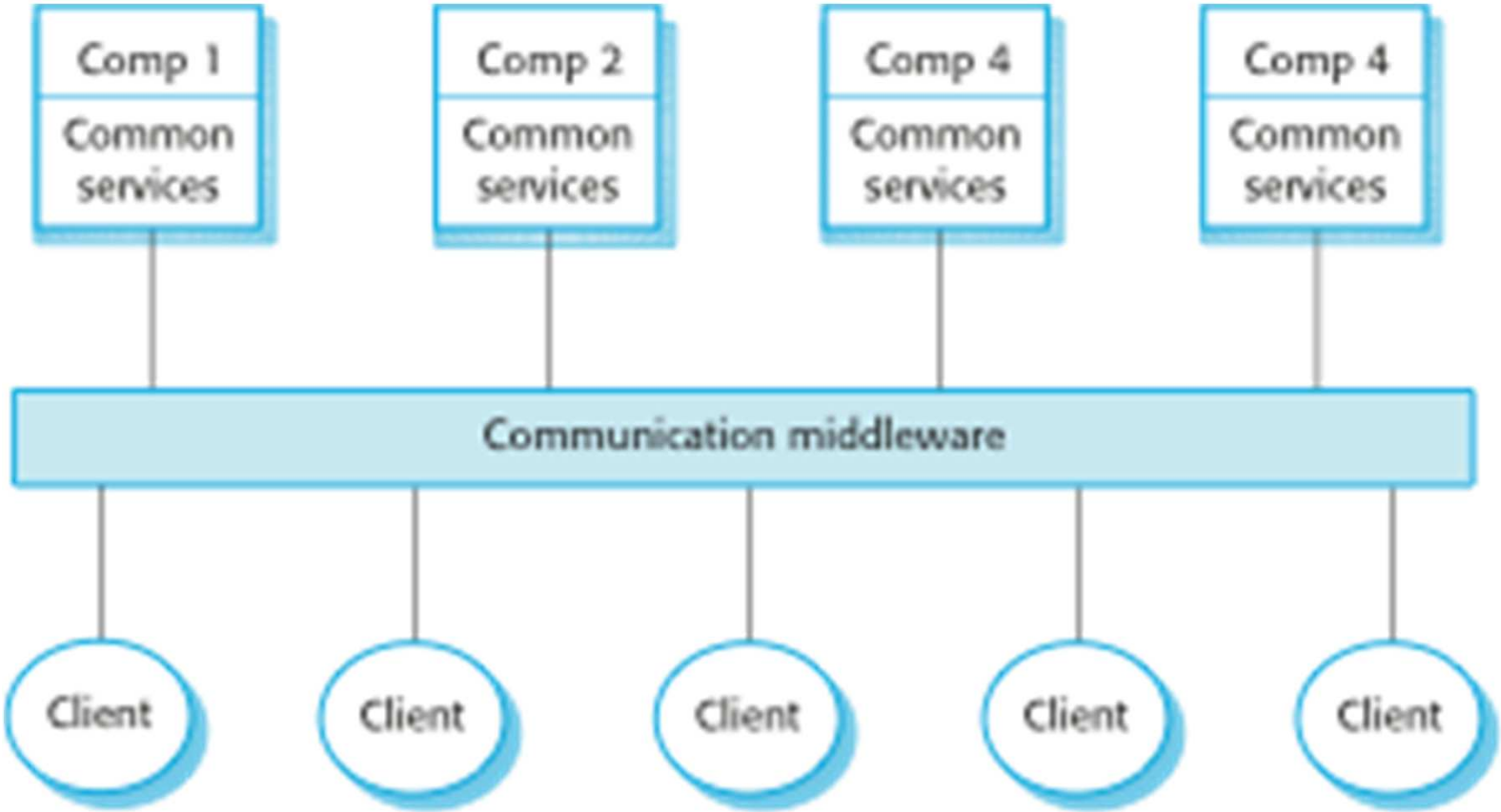
Architektura rozproszonych komponentów



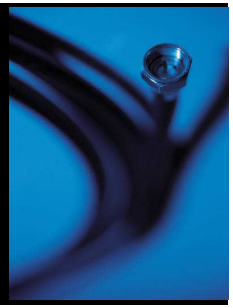
- Nie ma rozróżnienia na klientów i serwery.
- Każda rozproszona jednostka jest obiektem udostępniającym usługi i korzystającym z usług innych komponentów
- Komunikacja odbywa się za pośrednictwem warstwy pośredniej.
- Bardziej skomplikowana od architektury klient-serwer
- Zastosowanie: zasoby różnych systemów i baz danych muszą być połączone. Może również być to model implementacji wielowarstwowej architektury klient serwer.

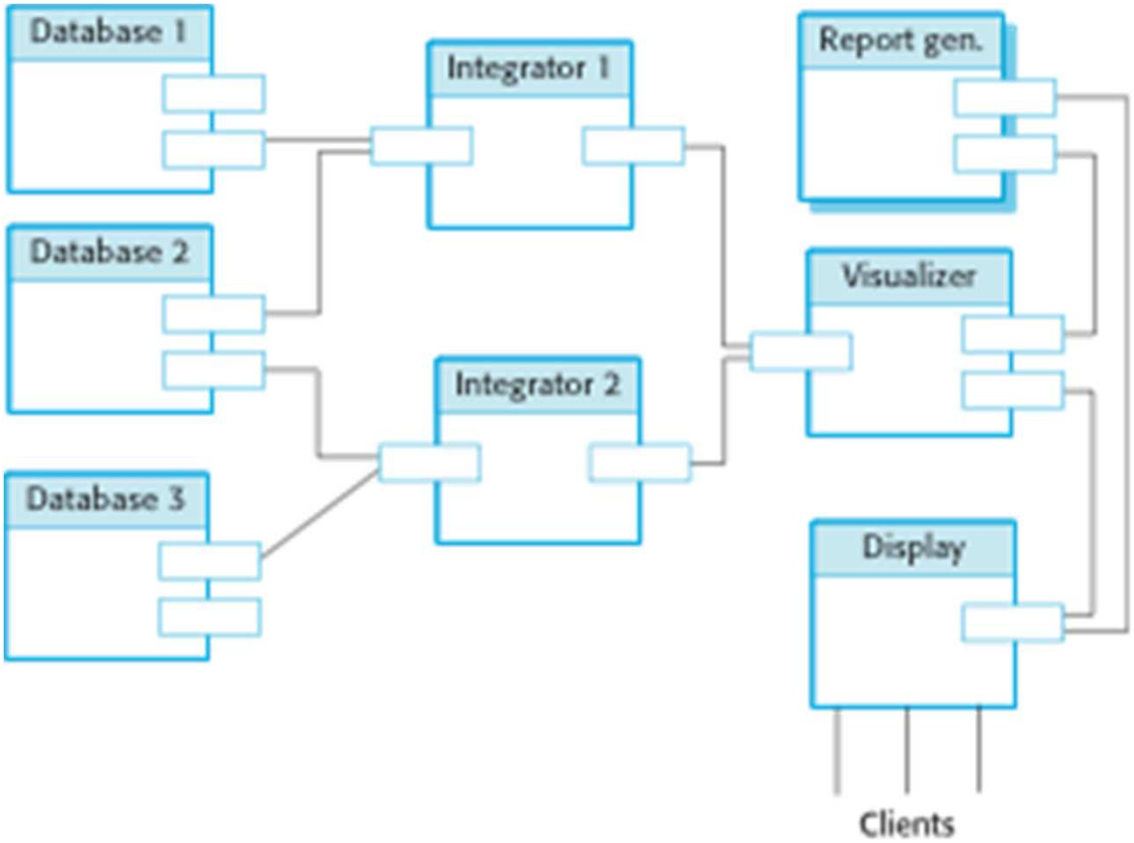
Architektura rozproszonych komponentów



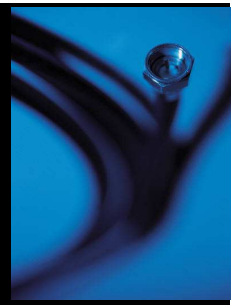


Architektura rozproszonych komponentów dla systemu typu Data Mining



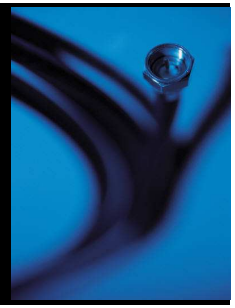


Wady architektury rozproszonych komponentów



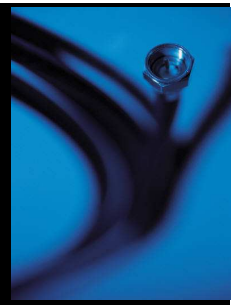
- Skomplikowanie - dużo trudniejsze w projektowaniu w stosunku do architektury klient – serwer. Trudno ją zrozumieć i zwizualizować.
- Żaden standard warstwy pośredniej nie uzyskał szerokiej akceptacji. Różni dostawcy (np. Microsoft czy Sun) opracowali własne niekompatybilne środowiska.
  - W rezultacie tych problemów, w wielu sytuacjach architektura rozproszonych komponentów zastępowana jest architekturą zorientowaną usługowo.

Architektury Peer-to-peer



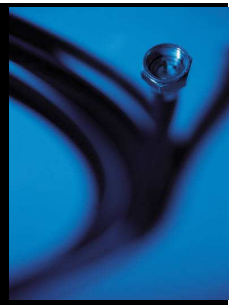
- Systemy Peer to peer (p2p) to systemy zdecentralizowane, w których obliczenia mogą być przeprowadzane na dowolnym węźle w sieci.
- System projektowany jest w taki sposób, aby wykorzystać moc obliczeniową oraz pamięć dużej liczby połączonych sieciowo komputerów.
- Większość systemów p2p stanowią systemy niebiznesowe. Jednak technologia jest również wykorzystywana w systemach biznesowych.

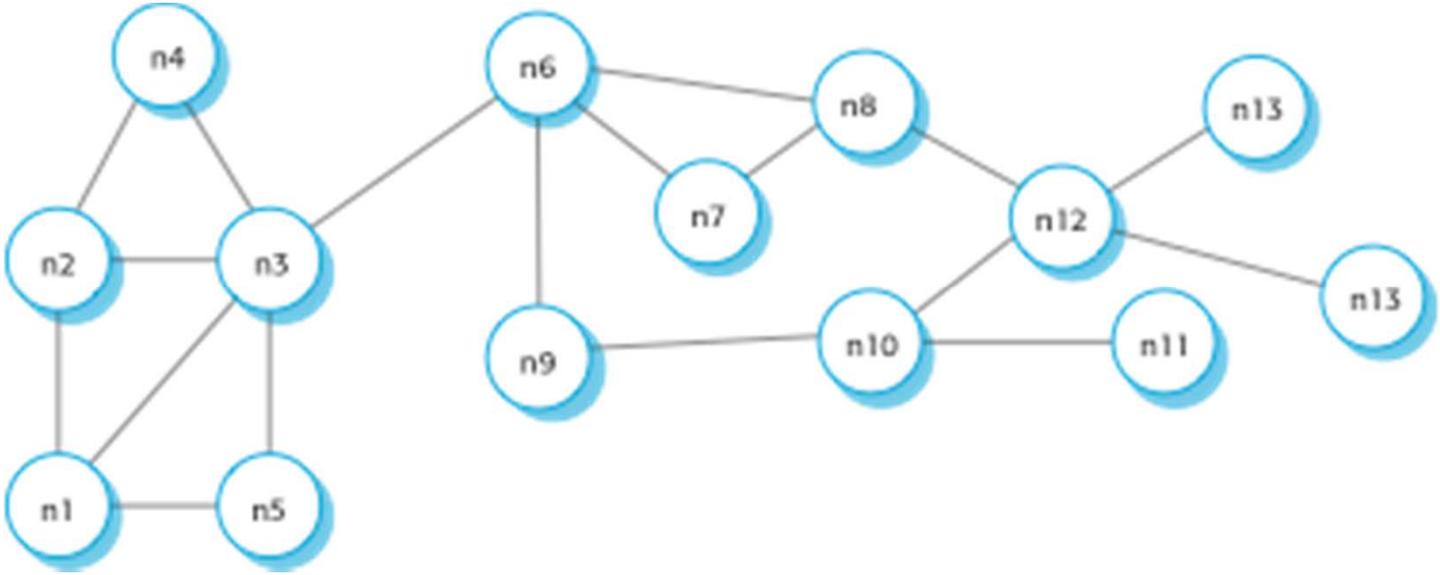
Modele architektury P2P



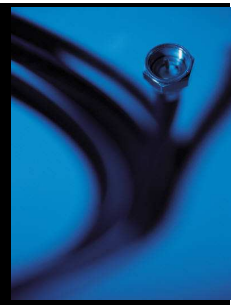
- Logiczna architektura sieci
  - Zdecentralizowana;
  - Pół-scentralizowana.
- Architektura aplikacji
  - Generyczna organizacja komponentów tworzy aplikację p2p.
- Architektura P2P skupia się na architekturze sieci.

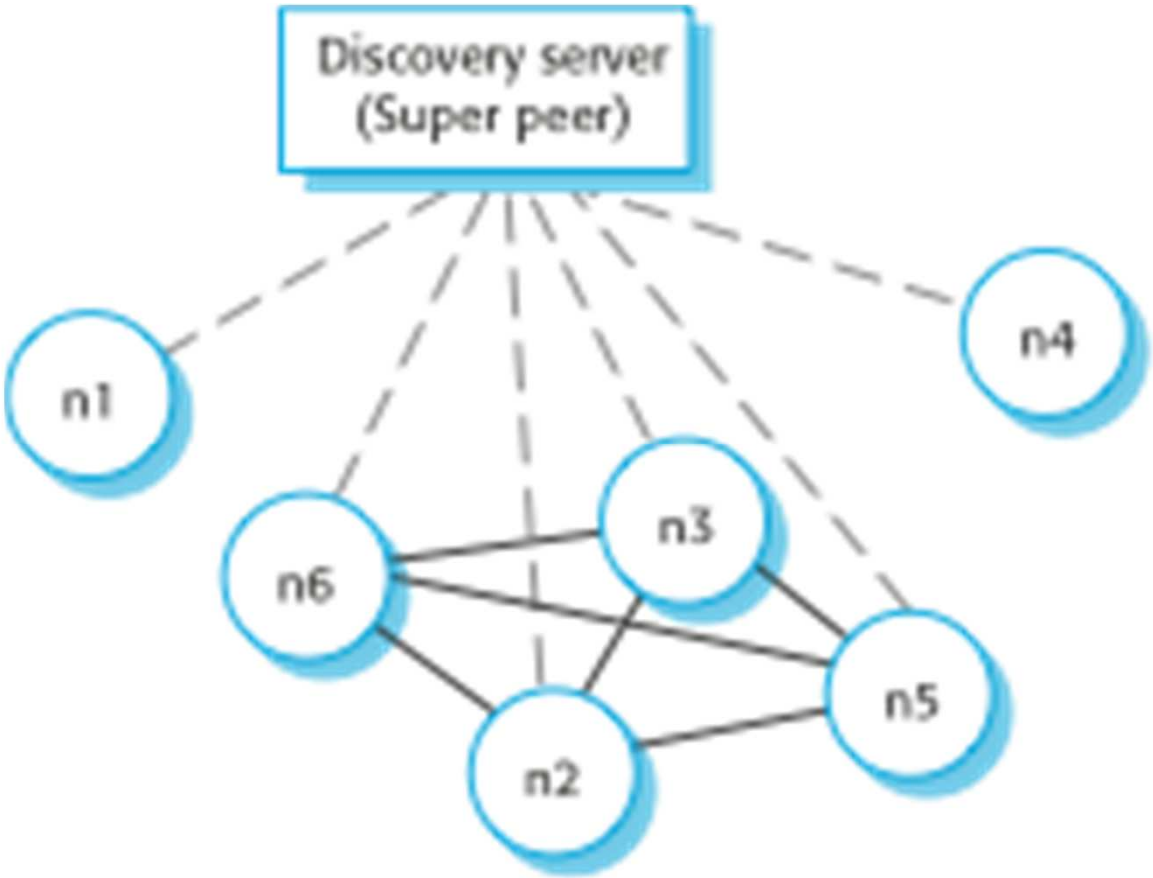
Zdecentralizowana architektura P2P



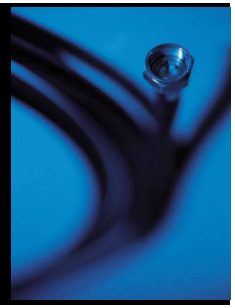


Pół-scentralizowana architektura P2P



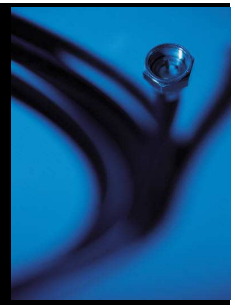


Oprogramowanie jako usługa (Software as a service - SaaS)



- Zdalny hosting oprogramowania i udostępnianie dostępu za pośrednictwem Internetu.
  - Oprogramowanie jest wdrażane na serwerze (zbiorze serwerów) i dostępne za pośrednictwem przeglądarki Internetowej.
  - Oprogramowanie jest najczęściej własnością udostępniającego a nie organizacji, która z niego korzysta.
  - Opłata może być pobierana za zakres wykorzystania, być powiązana z czasową subskrypcją lub innym modelem (np. wymaganie oglądania reklam).
- Jest to jeden z modeli tzw. chmur obliczeniowych (Cloud Computing)

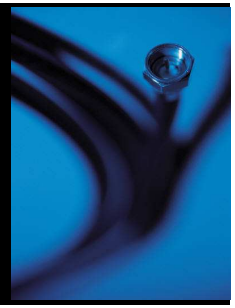
SaaS a SOA



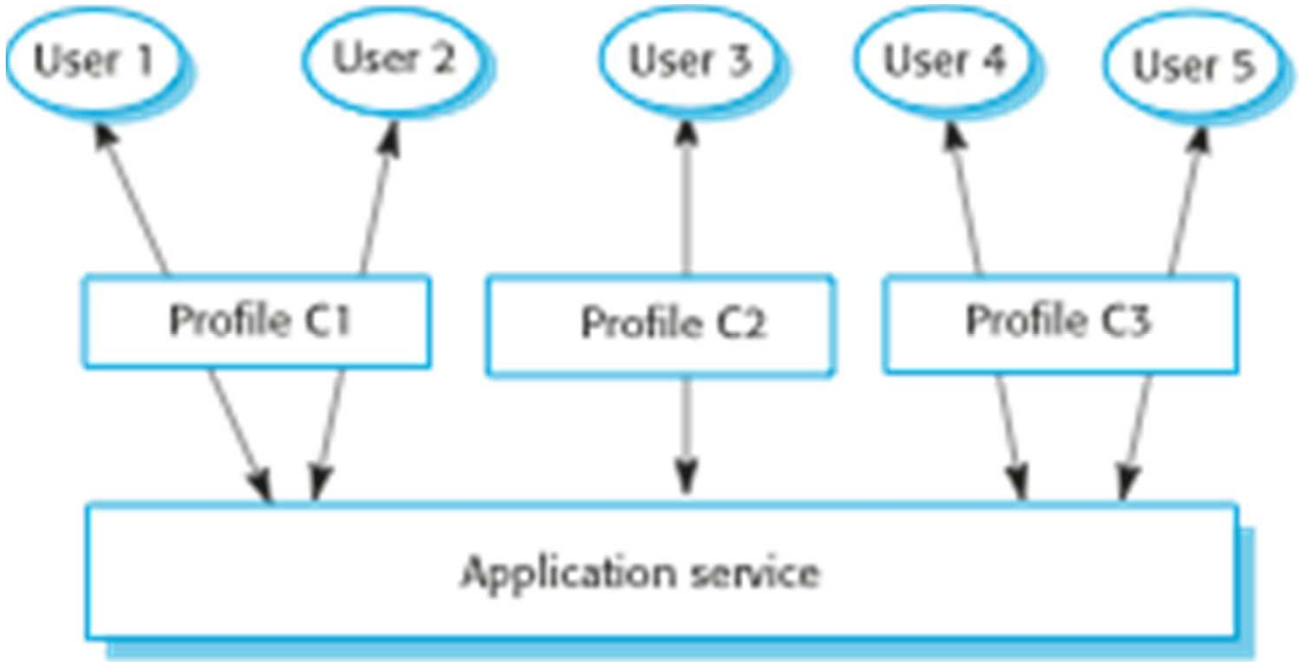
- Oprogramowanie jako usługa to sposób udostępniania funkcjonalności na zdalnym serwerze, do którego klient uzyskuje dostęp za pośrednictwem przeglądarki Internetowej. Serwer zarządza danymi użytkownika oraz stanem przetwarzania podczas sesji. Transakcje są zazwyczaj długie (np. edycja dokumentów).

- Architektura zorientowana usługowa (Service Oriented Architecture - SOA) to podejście polegające na budowaniu systemu w postaci zbioru bezstanowych usług. Usługi te mogą być dostarczane przez różnych dostawców i mogą być rozproszone. W tego typu rozwiązaniach transakcje są zazwyczaj krótkie – usługa jest wołana – wykonuje działanie i zwraca rezultat.

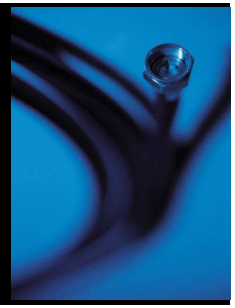
Czynniki implementacyjne dla SaaS (1)



- ***Konfigurowalność*** *- W jaki sposób konfigurować oprogramowanie dla specyficznych wymagań różnych organizacji?*



Czynniki implementacyjne dla SaaS (2)

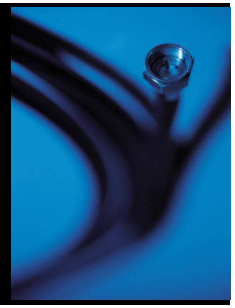


- Wynajmowanie wspólnej puli zasobów (ang. Multi-tenancy)
  - Udostępnianie oprogramowanie użytkownikowi, w taki sposób aby miał wrażenie, że pracuje na swojej własnej kopii systemu
  - Architektura systemu umożliwia efektywnie wykorzystanie zasobów.
  - Wymaga to istnienia bezwzględnej separacji funkcjonalności oraz danych systemu.

Współ-wynajmowana baza danych

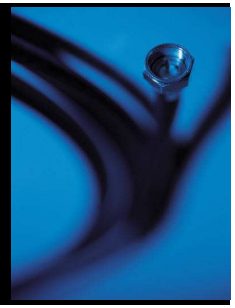
Tenant	Key	Name	Address
234	C100	XYZ Corp	43, Anystreet, Sometown
234	C110	BigCorp	2, Main St, Motown
435	X234	J. Bowie	56, Mill St, Starville
592	PP37	R. Burns	Alloway, Ayrshire

Czynniki implementacyjne dla SaaS (3)



- *Skalowalność – W jaki sposób zaprojektować system aby można go było skalować do nieprzewidywalnie dużej liczby użytkowników?*
  - Komponenty aplikacji implementowane w postaci prostych, bezstanowych usług, które mogą być wykonane na dowolnym serwerze.
  - Asynchroniczna interakcja – aplikacja nie musi czekać na rezultat interakcji (na przykład żądania odczytu).
  - Pula zasobów – zarządzanie zasobami (sieć, połączenia z bazą danych) w postaci puli – minimalizuje to prawdopodobieństwo braku zasobów dla realizacji usługi.
  - Drobnoziarnistość blokowania danych w bazie danych.

Podsumowanie (1 z 2)

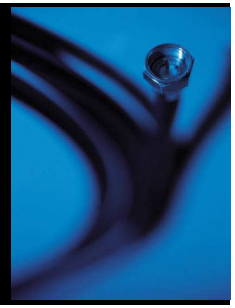


- Zaletą systemów rozproszonych jest możliwość ich skalowania w miarę wzrastających potrzeb. Może również kontynuować udostępnianie usług pomimo awarii oraz pozwala na współdzielenie zasobów.

- Do istotnych zagadnień, które należy wziąć po uwagę w trakcie projektowanie systemów rozproszonych należą: przezroczystość, otwartość, skalowalność, zabezpieczenie, jakość obsługi oraz zarządzanie awariami.

- Systemy typu klient-serwer mają strukturę warstwową. Warstwy mogą być rozproszone na różne komputery

Podsumowanie (2 z 2)



- Do wzorców architektury systemów rozproszonych należą: architektura master-slave, dwuwarstwowa i wielowarstwowa architektura typu klient-serwer, architektura rozproszonych komponentów oraz architektura peer-to-peer.
- Architektura rozproszonych komponentów wymaga istnienia warstwy pośredniej. Jej podstawowym zdaniem jest pośredniczenie w komunikacji pomiędzy komponentami oraz umożliwienie dodawania i usuwania komponentów do/z systemu.
- Architektura peer-to-peer reprezentuje zdecentralizowaną strukturę systemu bez podziału na usługobiorcę (klienta) i usługodawcę (serwer). Obliczenia mogą być rozproszone pomiędzy wiele systemów i organizacji.
- Oprogramowanie jako usługa (Software as a service) to metoda wdrażania systemów z cienkim klientem, w którym klientem jest przeglądarka Internetowa.