Systemy rozproszone

Komunikacja

Marek Libuda
Jerzy Brzeziński
Cezary Sobaniec

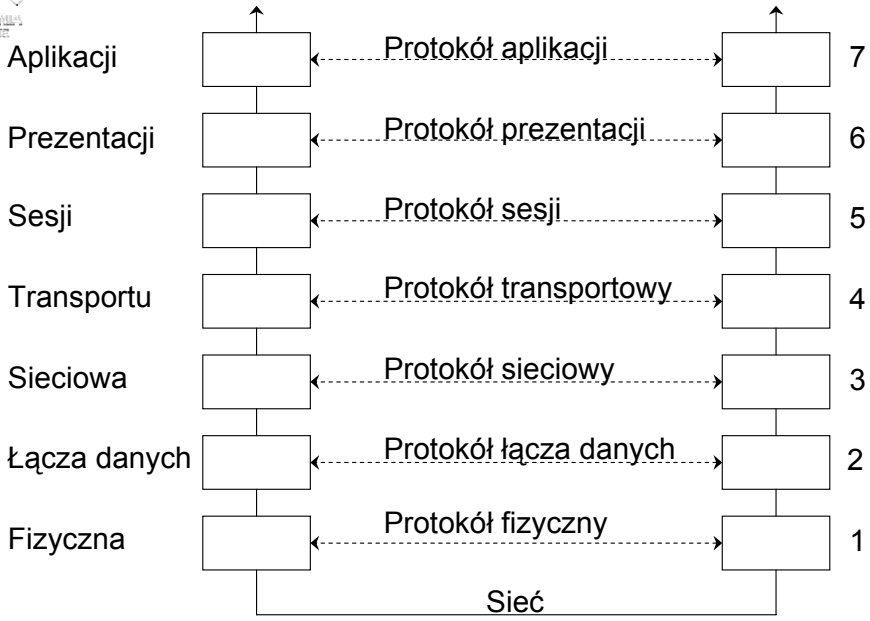
Komunikacja pomiędzy procesami jest centralnym elementem systemu rozproszonego. Niniejszy wykład omawia różne modele komunikacji, zgodnie z ich rosnącym poziomem abstrakcji. Na początku przedstawiono podstawowe koncepcje związane z niskopoziomową wymianą wiadomości w sieciach komputerowych, a następnie mechanizmy wyższego poziomu, dostarczające wygodniejszych niż wysyłanie i odbieranie wiadomości mechanizmów komunikacji takich jak: zdalne wywoływanie procedur (RPC), zdalne wywoływanie metod (RMI), systemy transferu wiadomości oraz komunikację strumieniową.

Wymiana wiadomości sieciowych

- Jest podstawą wszelkiej komunikacji w systemach rozproszonych
- Wszelkie inne modele są nabudową nad wymianą wiadomości
- Jest realizowana przez protokoły sieciowe

Ze względu na brak pamięci współdzielonej, wszelka komunikacja w systemach rozproszonych odbywa się za pomocą wymiany niskopoziomowych wiadomości sieciowych. Należy przy tym uzgodnić wiele szczegółów. Rozważmy następujący przykład. Gdy proces A chce komunikować się z procesem B, buduje w wiadomość swojej przestrzeni adresowej. Następnie wykonuje wywołanie systemu operacyjnego, które powoduje, że system operacyjny przesyła tę wiadomość do procesu B. Jakie napięcie przyjąć dla oznaczenia bitu 0, a jakie dla bitu 1? Jak odbiorca rozpozna ostatni bit wiadomości? Jak może stwierdzić, że wiadomość została uszkodzona, i co może wówczas zrobić? Ile bitów zajmują dane różnego typu: liczby całkowite, rzeczywiste, znaki itd.? Jak widać, uzgodnienia muszą istnieć na różnych poziomach, począwszy od detali niskopoziomowej transmisji danych, a skończywszy na wysokopoziomowych szczegółach reprezentacji danych.

Model warstwowy ISO/OSI



Aby ułatwić ustalanie szczegółów na różnych poziomach abstrakcji, organizacja ISO opublikowała w 1983 roku model odniesienia, który definiuje różne poziomy w postaci warstw, nadaje im standardowe nazwy oraz określa, jaka jest rola każdej warstwy. Model nosi angielską nazwę *Open Systems Interconnection*, w skrócie **OSI** i bywa nazywany modelem warstwowym lub modelem OSI.

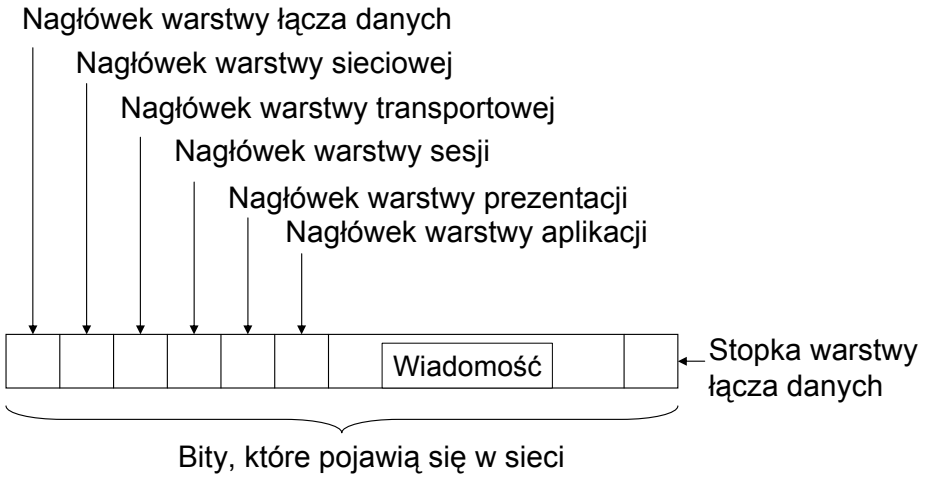
Protokoły wypracowane w ramach modelu OSI nigdy nie były powszechnie używane, ale sam model okazał się bardzo pomocny do zrozumienia modularnej budowy sieci komputerowych oraz użyteczny w dydaktyce.

Pojedyncza warstwa odpowiada za realizację odrębnego zbioru funkcji. Każda warstwa (oprócz najwyższej) dostarcza warstwie będącej bezpośrednio wyżej interfejsu do swoich usług. Komunikacja w sieci zgodnej z modelem warstwowym odbywa się tylko między odpowiadającymi sobie warstwami po obydwu stronach. Warstwa numer N po stronie nadawczej przesyła więc wiadomość tylko do warstwy numer N po stronie odbiorczej. Zestaw mechanizmów definiujących daną warstwę nazywa się **protokołem**. Jak widać na rysunku, protokoły działają tylko w jednej warstwie.

Wyróżnia się dwa typy protokołów. W **protokołach połączeniowych** (ang. *connection-oriented*) przed wymianą danych następuje faza jawnego nawiązania połączenia, a po wymianie danych – faza jawnego zakończenia połączenia. Przykładem komunikacji połączeniowej jest tradycyjna telefonia. W protokołach **bezpołączeniowych** (ang. *connectionless*) strony nie nawiązują połączenia, lecz nadawca wysyła pierwszą wiadomość, gdy tylko jest ona gotowa. Przykładem komunikacji bezpołączeniowej w życiu codziennym jest wysłanie listu pocztą.

Jak widać na rysunku, model OSI wyróżnia siedem warstw. Choć w sieci Internet używa się modelu obejmującego cztery warstwy, w dalszej części skupimy się na modelu OSI, gdyż jest bardziej ogólny. Pokrótce omówimy funkcje wszystkich warstw, miejscami odnosząc się do modelu sieci Internet.

Model warstwowy – kapsułkowanie



Komunikację pomiędzy warstwami umożliwia proces zwany **kapsułkowaniem** (ang. *encapsulation*). Kapsułkowanie polega na umieszczeniu wiadomości warstwy wyższej wewnątrz wiadomości warstwy niższej. Zanim wiadomość po stronie nadawczej zostanie wysłana, przekazywana jest przez aplikację „w dół" stosu warstw; każda kolejna warstwa po otrzymaniu tej wiadomości od warstwy wyższej, dodaje do niej własny nagłówek i (rzadziej) stopkę. U odbiorcy natomiast zachodzi proces odwrotny. Wiadomość przekazywana jest „w górę" stosu warstw i każda kolejna warstwa interpretuje, a następnie usuwa nagłówek dodany poprzednio przez tę samą warstwę u nadawcy. Tak okrojona wiadomość przekazywana jest następnie warstwie wyższej. Aplikacja u odbiorcy otrzymuje zatem oryginalne dane, wysłane wcześniej przez aplikację po stronie nadawczej.

Każda wiadomość jest złożona z dwóch części. Pierwszą z nich stanowi **nagłówek**, czyli informacje sterujące, które dana warstwa dodaje w celu komunikowania się z tą samą warstwą po drugiej stronie. Druga część to **dane**, czyli wszystko to, co dana warstwa otrzymała do wysłania od warstwy wyższej (najczęściej dane te zawierają już kilka nagłówków warstw wyższych).

Na rysunku ukazano typową wiadomość utworzoną przez oprogramowanie realizujące model warstwowy OSI, i znajdującą się w warstwie fizycznej, czyli przygotowaną do fizycznego wysłania. Oryginalna wiadomość została obudowana nagłówkami kolejnych warstw, zgodnie z mechanizmem opisanym powyżej. Widać ponadto, że stopkę na końcu wiadomości dołącza tylko warstwa łącza danych.

Oczywiście wszystkie informacje sterujące w postaci nagłówków i stopek sprawiają, że w sieci zostanie przesłanych więcej bitów, niż zajmuje oryginalna wiadomość.

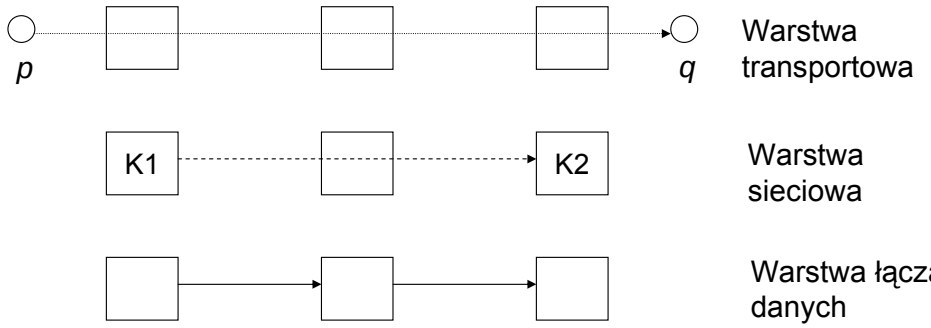
Warstwa fizyczna

- Transmisja bitów przez medium
- Definiuje interfejsy mechaniczne i elektryczne transmisji
- Istnieje wiele standardów warstwy fizycznej, np.
  - RS-232-C
  - RS-449
  - V-35
  - V-90 itd.

Warstwa fizyczna definiuje mechanizmy odpowiedzialne za fizyczną transmisję bitów przez medium komunikacyjne. Specyfikuje również parametry mechaniczne i elektryczne transmisji, które dla warstwy wyższej stanowią interfejs dostępu do warstwy fizycznej.

Warstwy 2 do 4

- Umożliwiają łączenie procesów w rozległej sieci komputerowej



Zadaniem warstw: łącza danych, sieciowej oraz transportowej, jest umożliwienie budowania złożonych sieci urządzeń oraz łączenie procesów w takich sieciach. Ilustruje to rysunek na slajdzie. U dołu rysunku widnieje warstwa 2, łącza danych, której zadaniem jest łączenie *bezpośrednio sąsiadujących* ze sobą urządzeń w niewielkie sieci, zwane sieciami lokalnymi LAN (ang. Local Area Networks). Znajdująca się powyżej warstwa 3, sieciowa, korzysta z funkcjonalności dostarczanej przez warstwę 2, by *łączyć lokalne sieci w większą strukturę sieciową*, w sieć sieci (najbardziej znanym przykładem takiej sieci jest Internet). W warstwach 2 oraz 3 łączenie dotyczy jednak wyłącznie *urządzeń*. Dopiero warstwa 4, transportowa, wykorzystuje funkcjonalność warstwy sieciowej, aby umożliwić łączenie *procesów* pracujących na odległych od siebie komputerach. Można zatem w uproszczeniu powiedzieć, że warstwy 2 i 3 łączą komputery, a warstwa 4 – procesy.

W każdej z warstw wykorzystuje się konkretne protokoły. Przykładowo, w sieci Internet w warstwie łącza danych do najbardziej znanych protokołów sieci LAN należą Ethernet i WiFi, jedynym protokołem w warstwie sieciowej jest IP (ang. *Internet Protocol*), a w warstwie transportowej są dostępne dwa protokoły: TCP (dla transmisji połączeniowej) oraz UDP (dla transmisji bezpołączeniowej).

Warstwy 5 i 6

- Warstwa sesji (5), zarządza dialogiem pomiędzy stronami
- Warstwa prezentacji (6), definiuje struktury wiadomości oraz ich formaty
- W sieci Internet ich funkcje realizuje warstwa aplikacji

Warstwa sesji to praktycznie rozwinięcie warstwy 4 o funkcje zarządzania interakcją pomiędzy komunikującymi się procesami. W tym celu definiuje szereg mechanizmów synchronizacji. Ponadto, ma ona dostarczać pewnych mechanizmów do tworzenia punktów kontrolnych, na wypadek, gdyby jedna ze stron uległa awarii podczas długo trwającej transmisji. Nie trzeba wówczas powtarzać całej transmisji, wystarczy jedynie cofnąć się do ostatniego punktu kontrolnego.

W odróżnieniu od warstw niższych, zainteresowanych jedynie przesłaniem bitów od nadawcy do odbiorcy, warstwa prezentacji zajmuje się *znaczeniem* tych bitów. Definiuje w tym celu struktury wymienianych wiadomości oraz ich formaty, ułatwiające wymianę wiadomości pomiędzy maszynami z różnymi reprezentacjami wewnętrznymi.

W rzeczywistości, warstwy 5 i 6 nie doczekały się realizacji w postaci odrębnych protokołów. Ich funkcje bowiem realizują protokoły aplikacyjne.

Warstwa aplikacji

- Obejmuje wszelkie aplikacje sieciowe:
  - standardowe, jak np. e-mail, przesyłanie plików, www...
  - i niestandardowe, np. systemy rozproszone
- Aplikacje i protokoły aplikacyjne to nie to samo, nawet jeśli mają taką samą nazwę!

Warstwa aplikacji pierwotnie była przeznaczona dla standardowych aplikacji sieciowych, jak np. poczta elektroniczna, transfer plików czy emulacja terminala. Z czasem jednak znalazły w niej miejsce wszystkie te aplikacje, których z różnych powodów nie udało się umieścić w niższych warstwach. Na przykład, z perspektywy modelu OSI praktycznie wszystkie systemy rozproszone są właśnie aplikacjami.

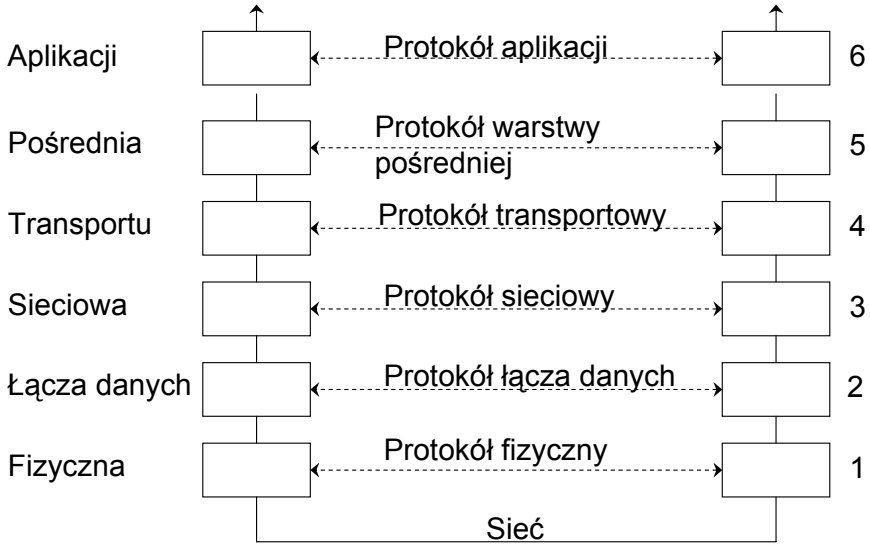
Należy pamiętać, że aplikacja jest czymś innym niż protokół aplikacyjny. Protokół jest bowiem *specyfikacją*, a aplikacja – *działającym programem*. Przykładowo, protokół **FTP** (ang. *File Transfer Protocol*) definiuje protokół do przesyłania plików pomiędzy maszynami klienta i serwera w sieci Internet. Nie można jednak mylić protokołu z programem ftp, będącym aplikacją użytkownika wykorzystywaną do przesyłania plików i realizującą protokół FTP.

Warstwa pośrednia

- Szczególny rodzaj aplikacji – świadczy usługi ogólnego przeznaczenia
- Przykłady usług
  - uwierzytelnianie
  - autoryzacja
  - zatwierdzanie atomowe
  - synchronizacja
  - zwielokrotnianie
  - zarządzanie spójnością kopii

Warstwa pośrednia (ang. *Middleware layer*) to szczególny rodzaj aplikacji – jej miejsce jest wprawdzie w warstwie aplikacji, ale obejmuje ona szereg protokołów ogólnego przeznaczenia, które powinny znaleźć się w odrębnej warstwie w postaci zbioru usług dla warstwy wyższej. Istnieje wiele protokołów warstwy pośredniej, jak np. uwierzytelnianie i autoryzacja, zatwierdzanie atomowe (ang. *Atomic commit*), synchronizacja dostępu do zasobów czy zwielokrotnianie i związane z nim zarządzanie spójnością kopii. Wszystkie one są usługami ogólnego przeznaczenia, czyli niezależnymi od konkretnej aplikacji. Z tego względu wyodrębnia się je w postaci osobnej warstwy, jak to ukazuje następny slajd.

Model z warstwą pośrednią



W porównaniu z modelem OSI, warstwy sesji i prezentacji zostały tu zastąpione warstwą pośrednią, zawierającą protokoły niezależne od aplikacji. Funkcjonalność warstw sesji i prezentacji jest natomiast realizowana przez warstwy pośrednią i aplikacji.

W dalszej części wykładu zostaną omówione cztery wysokopoziomowe usługi warstwy pośredniej: zdalne wywoływanie procedur, zdalne wywoływanie metod, systemy transferu wiadomości oraz komunikacja strumieniowa. Wszystkie te usługi na niższym poziomie wykorzystują oczywiście przesyłanie wiadomości.

RPC – koncepcja

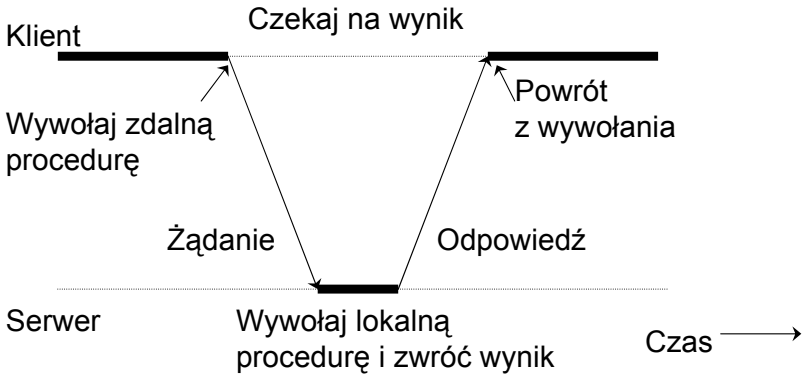
- Cel – ukrycie komunikacji sieciowej przed procesami aplikacyjnymi
- Sposób – umożliwić wywołanie procedury na zdalnej maszynie
- Informacje można przesłać w postaci parametrów
- Skrót RPC od nazwy Remote Procedure Call

Wymiana wiadomości stanowi podstawę komunikacji pomiędzy procesami w systemie rozproszonym. Okazuje się jednak niewystarczająca do modelowania komunikacji z dwóch podstawowych powodów. Nie zapewnia bowiem przezroczystości rozproszenia. Przykładowo, jawne wysłanie wiadomości do innego procesu oznacza, że proces nadawcy musi być świadomy, do kogo wysyła wiadomość, a to oznacza, że jest świadomy rozproszenia. Ponadto, w sytuacji gdy wiadomość przesyłana jest pomiędzy maszynami o różnych wewnętrznych reprezentacjach danych, proces nadawcy musi wysyłaną wiadomość dostosować do reprezentacji odbiorcy, co znacznie utrudnia osiągnięcie przezroczystości dostępu (ang. *Access transparency*).

W 1984 roku zaproponowano nowy sposób komunikacji pomiędzy procesami na różnych maszynach, polegający na wywoływaniu procedur znajdujących się na innej maszynie. Informacje dodatkowe oraz wyniki mogą być przekazane w postaci parametrów. W ten sposób udaje się zupełnie ukryć przed programistą przekazywanie wiadomości. Model ten nazwano **zdalnym wywoływaniem procedur**, w skrócie **RPC** (ang. *Remote Procedure Call*).

Chociaż podstawowa idea wydaje się być prosta, pozostaje kilka poważnych problemów do rozwiązania. Przede wszystkim, wywołująca i wywoływana procedura znajdują się w różnych maszynach i są wykonywane w różnych przestrzeniach adresowych, a to powoduje pewne komplikacje. Należy ponadto przekazać parametry, co również może rodzić komplikacje, szczególnie w przypadku, gdy maszyny nie są identyczne. Wreszcie, każda ze stron może ulec awarii, co rodzi kolejne problemy. Większość wspomnianych tu komplikacji udaje się jednak rozwiązać, a RPC ciągle jest popularną techniką, stanowiącą podstawę wielu systemów rozproszonych.

Interakcja klienta i serwera



Gdy proces na maszynie klienta wywołuje zdalną procedurę, znajdującą się w maszynie serwera, zostaje zawieszony w oczekiwaniu na wynik wywołania. Wywołanie procedury zostaje przetworzone na komunikat sieciowy, który trafia do maszyny serwera. Serwer, po otrzymaniu wiadomości, odtwarza z niej wywoływaną procedurę wraz z wartościami parametrów, i po jej wykonaniu zwraca wynik, który jest następnie przesłany w postaci wiadomości do maszyny klienta.

Pieniek klienta i serwera

- Podnosi przezroczystość dostępu
- Jest procedurą jawiącą się jako lokalna...
- W rzeczywistości jest *interfejsem* do zdalnej procedury
- Przetwarza wywołanie oraz wynik na wiadomość sieciową
- Nazwa „pieniek” od angielskiej *stub*

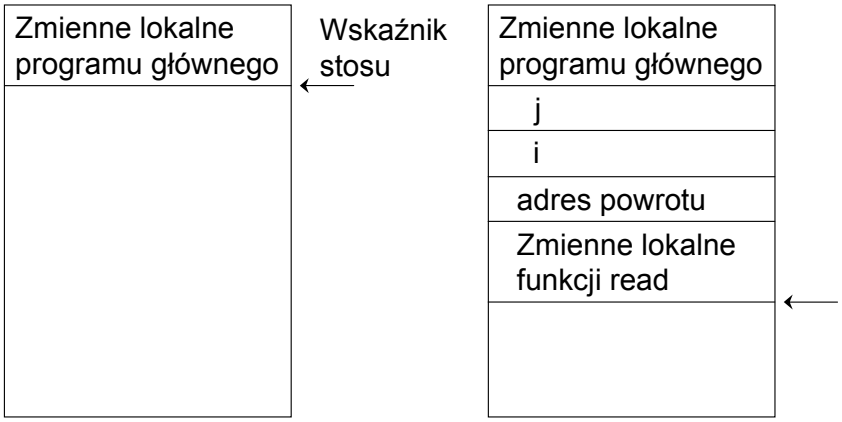
Podstawowy problem jest następujący: *Jak można sprawić, by z punktu widzenia uczestniczących procesów wywołanie zdalne przypominało lokalne?* Innymi słowy, celem jest dostarczenie procesowi wywołującemu wrażenia, że wywołuje zwyczajną, lokalną procedurę, a procesowi obsługującemu – że wywołanie pochodzi z lokalnej maszyny. Najistotniejsze jest ukrycie rozproszenia oraz komunikacji sieciowej.

Rozwiązaniem jest tak zwany **pieniek** (ang. *stub*). **Pieniek klienta** to procedura, którą w rzeczywistości wywołuje klient, a która przetwarza wywołanie i wartości jego parametrów na komunikat sieciowy. Z punktu widzenia procesu klienta wywołanie zdalne przebiega wówczas tak samo, jak lokalne. Komunikat sieciowy, po dotarciu do maszyny serwera, musi zostać na powrót przetworzony do postaci wywołania procedury. Zajmuje się tym **pieniek serwera**. Po odtworzeniu z wiadomości wywołania pieniek serwera dba o jego wykonanie, a następnie wynik, wraz z parametrami przekształca na wiadomość sieciową i wysyła do procesu klienta. Z punktu widzenia zdalnej procedury wydaje się więc, że wywołanie pochodzi z lokalnej maszyny. Wszelką „świadomość rozproszenia” posiadają pieńki po obydwu stronach, a nie same procesy klienta i serwera.

Zastosowanie pieńków nie byłoby atrakcyjne dla programisty, gdyby sam musiał je tworzyć. Okazuje się jednak, że pieńki mogą być w całości wygenerowane automatycznie. Zostanie to omówione na jednym z kolejnych slajdów.

Problem przekazania parametrów

```
k = add(i, j);
```



Na rysunku ukazano stos procesu, który wywołuje lokalną procedurę. Za przykład weźmy wywołanie add w języku C, obliczające sumę dwóch liczb całkowitych, i oraz j. Jeśli wywołanie nastąpi w programie głównym, na krótko przed wywołaniem stos wygląda tak, jak ukazuje lewa część rysunku. Aby wykonać funkcję, program odkłada wartości parametrów na stosie, począwszy od skrajnie prawego (takie rozwiązanie przyjęto dla języka C). Po wykonaniu add, wynik zostaje zapisany w rejestrze, adres powrotu jest usuwany ze stosu, a sterowanie powraca do programu głównego. Następnie program główny usuwa parametry ze stosu, przywracając go do postaci, którą miał przed wywołaniem.

Dla przykładu, w języku C parametry przekazywane są przez wartość (*call-by-value*) lub przez wskaźnik/odniesienie (*call-by-reference*). Przekazanie przez wartość polega na utworzeniu kopii wartości parametru na stosie, a przekazanie przez wskaźnik odbywa się przez zapisanie na stosie *adresu w pamięci*, przeznaczonego na przechowanie wartości parametru. W języku C wartości typów prostych przekazywane są zawsze przez wartość, a wartości typów tablicowych – przez wskaźnik.

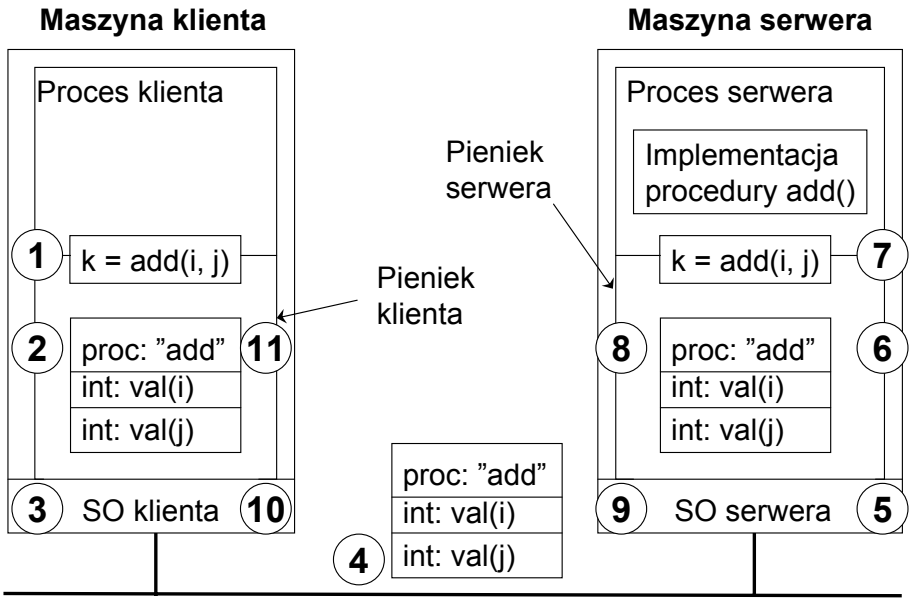
Powstaje problem, jak przekazywać parametry zdalnej procedurze. Przekazanie parametrów typów prostych odbywa się przez kopię. Pieniek klienta umieszcza wartość parametru w komunikacie sieciowym, wysyłanym do zdalnej procedury, a pieniek serwera ją później odtwarza. Wszystko to musi się po obydwu stronach odbywać zgodnie z ustalonym wcześniej protokołem przetwarzania wywołania do- i z wiadomości sieciowej.

Przekazywanie parametrów typów tablicowych o znanych rozmiarach w zasadzie odbywa się podobnie, z tym, przekazanie kopii następuje tutaj dwukrotnie. W pierwszej kolejności kopia wartości elementów tablicy jest przesyłana do pieńka serwera. Później, po wykonaniu procedury, pieniek serwera odsyła klientowi kopię tablicy, zawierającej ewentualne zmiany wprowadzone przez procedurę. Jest to technika znana pod angielską nazwą *copy-restore*. Dla parametrów o charakterze wyłącznie wejściowym lub wyłącznie wyjściowym można ją optymalizować, rezygnując z przesłania kopii w jedną stronę.

Dużo bardziej skomplikowane jest przekazywanie wartości złożonych typów danych, których rozmiar nie jest znany. Niektóre systemy próbują rozwiązać ten problem, wymagając od programisty dostarczenia dodatkowych procedur do przekazania takich parametrów.

Drugi problem bierze się z faktu, że w różnych architekturach maszyn stosuje się różne reprezentacje elementarnych typów danych. Rozwiązanie polega najczęściej na wprowadzeniu *zewnętrznej, uniwersalnej* reprezentacji, do której muszą się dostosować obie komunikujące się strony.

RPC krok po kroku



Na rysunku ukazano kolejne kroki wykonywania zdalnej procedury add, zachodzące od momentu wywołania przez klienta jego pieńka do momentu faktycznego wykonania procedury po stronie serwera. Poniżej wypunktowano te kroki, z uzupełnieniem ich o etap przesyłania wyniku do klienta.

1. Klient wywołuje procedurę add, która w rzeczywistości jest jego pieńkiem.
2. Pieniek klienta przetwarza wywołanie na wiadomość sieciową (ID procedury: „add”, parametr 1: typ int, wartość=i, parametr 2: typ int, wartość = j).
3. System operacyjny klienta wysyła wiadomość sieciową.
4. Wiadomość jest przesyłana przez sieć komputerową do maszyny serwera.
5. System operacyjny serwera przekazuje wiadomość pieńkowi serwera.
6. Pieniek odtwarza wywołanie z wiadomości sieciowej.
7. Pieniek wywołuje lokalnie procedurę add i zapamiętuje wynik.
8. Pieniek serwera przetwarza wynik wywołania na wiadomość sieciową i przekazuje swojemu systemowi operacyjnemu w celu wysłania przez sieć.
9. System operacyjny serwera wysyła wiadomość do systemu operacyjnego klienta.
10. System operacyjny klienta przekazuje wiadomość pieńkowi klienta.
11. Pieniek klienta przetwarza wiadomość sieciową na wynik wywołania i zwraca ten wynik procesowi klienta.

Generacja pieńków

- Kod pieńka zależy tylko od interfejsu zdalnej procedury
- Można zatem automatycznie generować kod pieńka, na podstawie definicji interfejsu procedury
- Język opisu interfejsu
  IDL (ang. *Interface Definition Language*)
- Definicja interfejsu w IDL jest kompilowana na pieńki klienta i serwera
- W praktyce wszystkie implementacje RPC dostarczają jakiegoś języka opisu interfejsu

Ponieważ funkcje pieńka klienta i serwera zależą wyłącznie od interfejsu procedury, kod programu realizujący pieńka można automatycznie wygenerować na podstawie definicji tego interfejsu. W praktyce wszystkie implementacje RPC dostarczają własnej odmiany języka do definiowania interfejsu. Jednym z takich języków jest IDL, od angielskiej nazwy *Interface Definition Language*. Definicja interfejsu w języku IDL jest kompilowana na kod pieńków klienta i serwera. Pozwala to zwolnić programistę z obowiązku samodzielnego tworzenia pieńka i tym samym znacząco uprościć i przyspieszyć konstrukcję systemu rozproszonego.

RMI – koncepcja

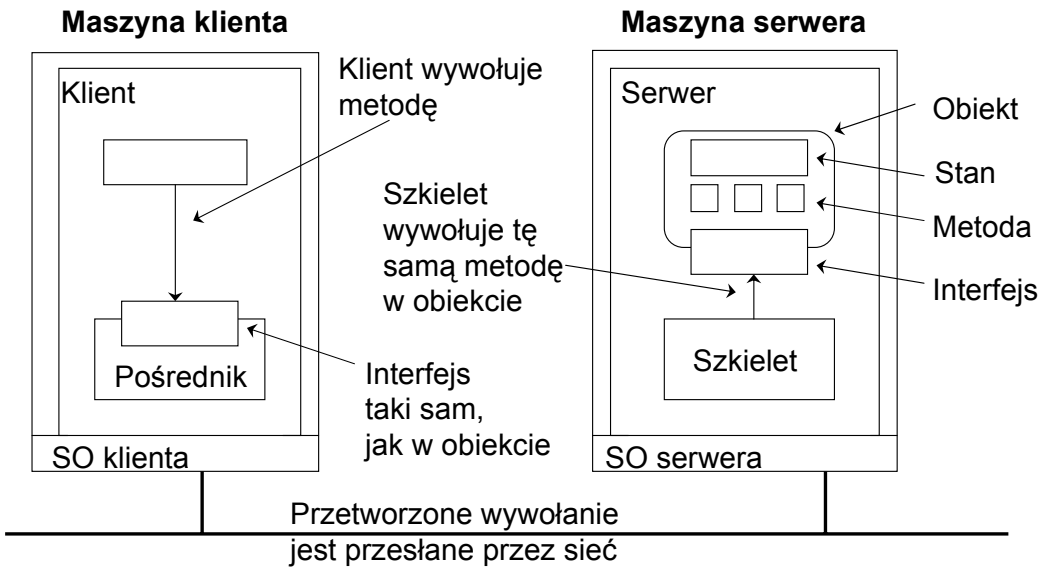
- Obiektowe RPC
- Dostarczyć klientowi interfejs do obiektu
- Ukryć przed klientem implementację obiektu
- Skrót RMI od angielskiej nazwy *Remote Method Invocation*

Technologia obiektowa okazała się bardzo atrakcyjnym narzędziem budowy aplikacji. Jedną z najistotniejszych właściwości obiektu jest to, że jego stan jest ukryty przed światem zewnętrznym, natomiast dostęp do niego odbywa się tylko przez jego operacje, które udostępnia na zewnątrz. Operacje te tworzą interfejs obiektu. Takie podejście pozwala zmieniać w razie potrzeby implementację obiektu, pod warunkiem że interfejs obiektu pozostaje bez zmian.

Z czasem, gdy RPC stało się najpopularniejszą techniką komunikacji procesów w systemach rozproszonych, pojawił się pomysł, by połączyć je z przetwarzaniem obiektowym, pozwoli to bowiem podnieść poziom przezroczystości rozproszenia względem tradycyjnego RPC. Ten nowy model obiektowego RPC nazwano **zdalnym wywoływaniem metod**, w skrócie **RMI**, od angielskiej nazwy *Remote Method Invocation*.

W dalszej części wykładu skupimy się na stosunkowo prostym modelu RMI, w którym cały stan obiektu znajduje się w jednym miejscu, a więc nie jest rozproszony.

Organizacja zdalnego obiektu



Kluczową cechą obiektu jest jego budowa. Obiekt tworzą dane, nazywane **stanem**, oraz operacje, które manipulują tymi danymi, nazywane **metodami**. Dostęp do danych obiektu odbywa się wyłącznie za pośrednictwem metod. Zbiór metod obiektu udostępnionych na zewnątrz to tak zwany **interfejs** obiektu. Jeden obiekt może implementować kilka interfejsów, i na odwrót – jeden interfejs może być implementowany przez wiele obiektów. Podobnie jak w RPC, interfejs obiektu jest zazwyczaj definiowany w specjalnym języku deklaratywnym, IDL. Proces na maszynie serwera, który przechowuje dany obiekt, będziemy nazywać serwerem; pojęcia serwera i obiektu będziemy często używać zamiennie.

Rozgraniczenie pomiędzy interfejsami a obiektami je implementującymi ma w systemach rozproszonych kluczowe znaczenie. Możliwe jest bowiem umieszczenie interfejsu i obiektu *na różnych maszynach*. Taka organizacja jest powszechnie nazywana **rozproszonym obiektem** lub **zdalnym obiektem**. Należy zaznaczyć, że wbrew nazwie stan obiektu rozproszonego nie musi być rozproszony.

Kiedy klient **dowiązuje się** (ang. *bind*) do obiektu, implementacja interfejsu obiektu po stronie klienta, nazywana **pośrednikiem** (ang. *proxy*), jest ładowana do przestrzeni adresowej klienta. Pośrednik jest tym samym, co pieniek klienta w modelu RPC. Jedynym jego zadaniem jest przetwarzanie wywołań metod zdalnego obiektu do postaci komunikatu sieciowego oraz odtwarzanie wyników tych wywołań z odbieranych od serwera komunikatów.

Rzeczywisty obiekt znajduje się na maszynie serwera. Obiekt ten udostępnia taki sam interfejs, jak pośrednik klienta. Rolę pieńka serwera pełni tak zwany **szkielet** (ang. *skeleton*), który przetwarza odbierane z sieci wiadomości na wywołania metod oraz wyniki tych wywołań na wiadomości sieciowe, odsyłane klientowi.

Kod pośrednika oraz szkieletu jest, podobnie jak w przypadku RPC, jest automatycznie generowany na podstawie definicji interfejsu obiektu.

Odniesienie do obiektu

- Globalny w skali systemu adres obiektu
- Odniesienie można przekazywać dalej
- Problemy implementacyjne
  - Jak zabezpieczyć odniesienie przed unieważnieniem na skutek awarii serwera?
  - Jaką informację adresową zaszyć w odniesieniu?

Zasadnicza różnica pomiędzy tradycyjnym modelem RPC a modelem RMI tkwi w tym, że RMI wprowadza pojęcie globalnego w skali systemu adresu obiektu, nazywanego **odniesieniem obiektowym** (ang. *object reference*). Takie odniesienie może być swobodnie przesyłane pomiędzy procesami pracującymi na różnych maszynach, na przykład w postaci parametru wywołania.

Ponadto, odniesienie najczęściej jest implementowane tak, by jego wnętrze było nieczytelne, tzn. by proces nie mógł odczytać z niego, gdzie znajduje się obiekt. Pozwala to zwiększyć stopień przezroczystości rozproszenia zasobów systemu.

Jednym z najważniejszych problemów projektowych jest ustalenie, jakie informacje adresowe zostaną zapisane w odniesieniu. Przykładowo, jeśli w odniesieniu znajdzie się adres IP serwera oraz numer portu, na którym obiekt (serwer) oczekuje na zgłoszenia, takie odniesienie będzie bardzo wrażliwe na ewentualną awarię serwera. Gdyby do niej doszło, zdalny obiekt może po wznowieniu pracy używać innego numeru portu, a to by oznaczało, że wszystkie dotychczasowe odniesienia do niego stają się nieważne.

Przykładowe rozwiązanie tego problemu polega na uruchomieniu na maszynie serwera wyróżnionego procesu, będącego rejestrem obiektów, oczekującego na wywołania na dobrze znanym numerze portu, zarezerwowanym tylko dla siebie. Rejestr przyjmowałby wywołania do obiektów, identyfikował wywoływany obiekt i przekazywał mu wywołanie. W odniesieniu znalazłby się wówczas tylko adres procesu-rejestru oraz identyfikator wywoływanego obiektu. Takie podejście (zastosowane zresztą w systemie DCE) pozwala częściowo uniezależnić odniesienie od dokładnego adresu obiektu, konkretnie od numeru portu. Nie umożliwia jednak przenoszenia procesu serwera wraz z jego obiektami na inną maszynę.

Oczywistym rozwiązaniem wydaje się być zastosowanie **serwera lokalizacji**, pamiętającego adres maszyny, na której aktualnie działa poszukiwany obiekt. Odniesienie zawierałoby wówczas wyłącznie adres serwera lokalizacji oraz identyfikator pozwalający odnaleźć szukany obiekt. Jednak i to podejście ma poważne ograniczenia, związane głównie z kwestią przeciążenia serwera lokalizacji, a tym samym z kwestią skalowalności systemu.

Rodzaje obiektów

- Implementacja obiektu
  - obiektowa
  - nieobiektowa – wymagany adapter obiektu
- Trwałość obiektów
  - obiekty trwałe – mogą przetrwać awarię serwera
  - obiekty ulotne

Należy podkreślić, że pojęcie *obiektu* w modelu RMI dotyczy głównie *wrażenia*, jakie system dostarcza aplikacji klienta. Najważniejsze jest to, by klientowi wydawało się, że pracuje z obiektem. Kwestia rzeczywistej realizacji tej abstrakcji obiektu jest drugorzędna. Chociaż najprostszym podejściem wydaje się być takie, w którym zdalne obiekty są realizowane przez obiekty języka programowania, w ogólności tak być nie musi. Obiekt zdalny może zostać zaimplementowany w dowolny sposób, pod warunkiem, że klient będzie miał złudzenie pracy z obiektem. W przypadku zdalnych obiektów realizowanych w nieobiektowych językach programowania, za dostarczanie klientowi tego złudzenia odpowiada **adapter obiektu** (ang. *Object adapter*). Jego rolą jest dostosowanie rzeczywistej implementacji obiektu do postaci, której oczekuje klient.

Obiekty są również klasyfikowane ze względu na ich trwałość. **Obiekty trwałe** (ang. *persistent*) charakteryzują się tym, że istnieją nawet wtedy, gdy nie znajdują się aktualnie w przestrzeni adresowej procesu serwera. Innymi słowy, są niezależne od bieżącego serwera i mogą przetrwać jego awarię. W praktyce oznacza to, że stan takich obiektów musi zostać przechowany w pamięci trwałej. W przeciwieństwie do nich, **obiekty ulotne** (ang. *transient*) istnieją tylko tak długo, jak długo działa utrzymujący je serwer.

Wywołania statyczne i dynamiczne

- Wywołania statyczne
  - z pomocą statycznego pieńka
- Wywołania dynamiczne
  - bez statycznej wiedzy o obiekcie
  - przykładowe zastosowanie – przeglądarka obiektów
- Ogólna postać wywołania dynamicznego

  `invoke(obiekt, metoda, arg_we, arg_wy)`

  np. dla wywołania `MojObiekt.MojaMetoda(i)`:

  `invoke(MojObiekt, ID(MojaMetoda), i, NULL)`

Wywołania statyczne dokonywane są przez klienta za pomocą statycznego pośrednika (pieńka). Wywołania dynamiczne zaś nie wykorzystują statycznego pośrednika. Polegają natomiast na użyciu po stronie klienta ogólnej funkcji, umożliwiającej dynamiczne utworzenie wywołania. Innymi słowy, korzystając z wywołań dynamicznych, klient w dużej mierze sam robi to, co zrobiłby za niego jego pośrednik.

Na slajdzie ukazano ogólną funkcję invoke, umożliwiającą dynamiczne utworzenie dowolnego wywołania zdalnej metody. W ramach jej parametrów należy określić szukany obiekt, wywoływaną w nim metodę, oraz wartości parametrów wejściowych i ewentualnie wyjściowych.

Przykładowo, dla wywołania statycznego postaci:

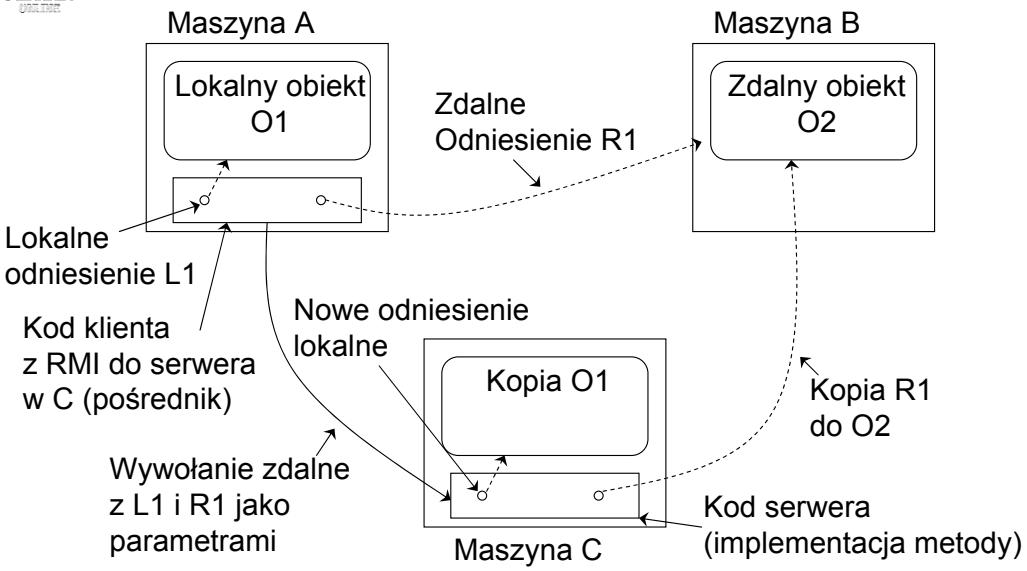
MójObiekt.MojaMetoda(i)

odpowiadające mu wywołanie dynamiczne wyglądałoby następująco:

Invoke(MójObiekt, ID(MojaMetoda), i, NULL).

W wielu sytuacjach wykorzystanie wywołań dynamicznych okazuje się koniecznością. Dzieje się tak na przykład w systemach, które przekierowują wywołania, powielają je do większej liczby odbiorców czy też w przeglądarkach obiektów.

Przekazywanie parametrów

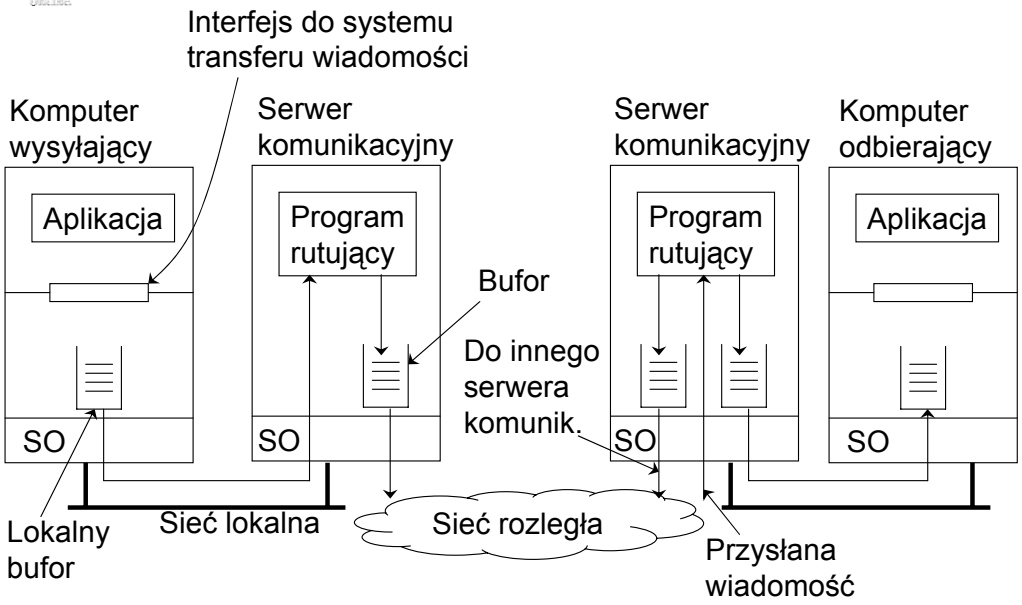


Dopóki przekazywanym parametrem nie będzie obiekt, stosuje się rozwiązania podobne do tych, które przedstawiono przy okazji tradycyjnego RPC. W modelu RMI pojawia się jednak nowy typ parametru, mianowicie obiekt.

Obiekty przekazywane są na dwa sposoby. Dla obiektów zdalnych, czyli takich, które powinny znajdować się tylko w jednej lokalizacji i być dostępne zdalnie, jedyną możliwością jest przekazanie *przez odniesienie*. Przekazywana jest kopia odniesienia do zdalnego obiektu. Proces otrzymujący kopię odniesienia posiada więc wskazanie na ten sam zdalny obiekt. Na przedstawionym rysunku widać, jak proces działający w maszynie A wywołuje zdalną metodę procesu maszyny C i przekazuje jako parametr odniesienie R1 do zdalnego obiektu O2. Od tej chwili na obiekt O2 wskazują dwa identyczne odniesienia. Wszelkie wywołania metod obiektu O2 mają zawsze charakter zdalny.

Dla obiektów lokalnych, a więc takich, które nie są dostępne zdalnie, jedyną opcją jest przekazanie ich *przez wartość* (in. przez kopię). Stan takich obiektów jest w całości kopiowany i przesyłany jako parametr metody. Na rysunku widać, jak lokalny obiekt O1 zostaje przekazany jako parametr wywołania zdalnego, dokonywanego przez proces maszyny A na obiekcie w maszynie C – w serwerze maszyny C tworzona jest kopia obiektu O1, a późniejsze wywołania metod obiektu O1 mają charakter lokalny.

System transferu wiadomości – organizacja



Istnieje szereg zastosowań, dla których modele RPC i RMI okazują się niewystarczające. Dotyczy to szczególnie aplikacji, w których nie można zakładać, że odbiorca wiadomości jest aktywny podczas jej przesyłania. Ponadto, synchroniczny charakter wywołań procedur czy metod, przy których klient jest blokowany do momentu otrzymania wyniku, często stanowi zbyt wielkie ograniczenie.

Odpowiedzią na te ograniczenia są **systemy transferu wiadomości**. Niniejsza część wykładu prezentuje komunikację zorientowaną na wiadomości. Omawia dostępne rodzaje komunikacji oraz ich przykładowe realizacje.

Na rysunku przedstawiono ogólną organizację systemu, w którym przesyłane są wiadomości. Aplikacje są zawsze wykonywane na komputerach końcowych, udostępniających interfejs, przez który wiadomości mogą być wysyłane. Komputery końcowe są połączone ze sobą za pośrednictwem sieci serwerów komunikacyjnych, odpowiedzialnych za przekazywanie (i routowanie) wiadomości. Zarówno komputery końcowe, jak i serwery komunikacyjne posiadają bufory, w których zapamiętują przesyłane wiadomości.

Rodzaje komunikacji

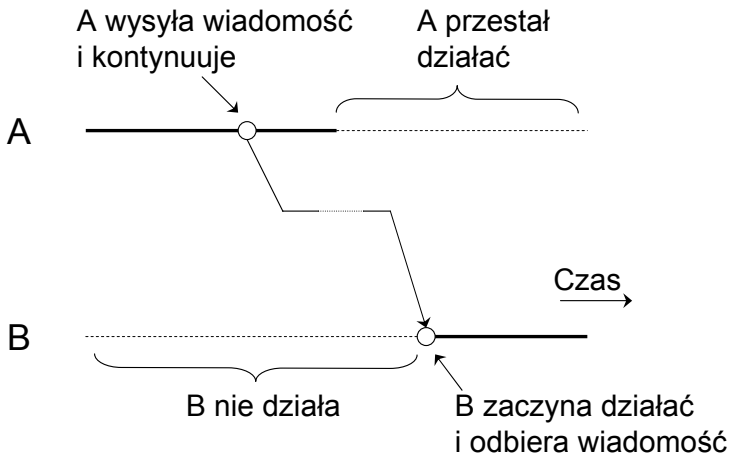
- Trwałość komunikacji
  - Trwała lub nietrwała
  - Czy nadawca i odbiorca muszą być aktywni podczas transferu?
- Synchronizm komunikacji
  - Synchroniczna lub asynchroniczna
  - Jak długo trwa operacja wysłania?

Z punktu widzenia trwałości wiadomości wyróżniamy dwa rodzaje komunikacji. Komunikacja jest **trwała** (ang. *persistent*), jeśli wysłana wiadomość jest pamiętana w systemie komunikacyjnym do momentu jej dostarczenia do odbiorcy. Odnosząc się do rysunku z poprzedniego slajdu można powiedzieć, że wiadomość jest pamiętana w jednym serwerze komunikacyjnym tak długo, aż zostanie dostarczona do następnego serwera komunikacyjnego. Nie ma zatem potrzeby, aby aplikacja wysyłająca kontynuowała działanie po przekazaniu wysyłanej wiadomości systemowi. Przykładem z życia codziennego usługi realizującej trwałą komunikację jest tradycyjna poczta.

W komunikacji **nietrwałej** (ang. *transient*) wiadomość jest pamiętana przez system komunikacyjny tylko tak długo, jak długo działają nadawca lub odbiorca. Odnosząc się znów do rysunku na poprzednim slajdzie, można bardziej precyzyjnie powiedzieć, że jeśli jeden serwer komunikacyjny nie będzie w stanie dostarczyć wiadomości do następnego serwera lub do odbiorcy, wiadomość zostanie porzucona. Ten sposób komunikacji przypomina działanie tradycyjnej telefonii z funkcją poczty głosowej lub automatycznej sekretarki.

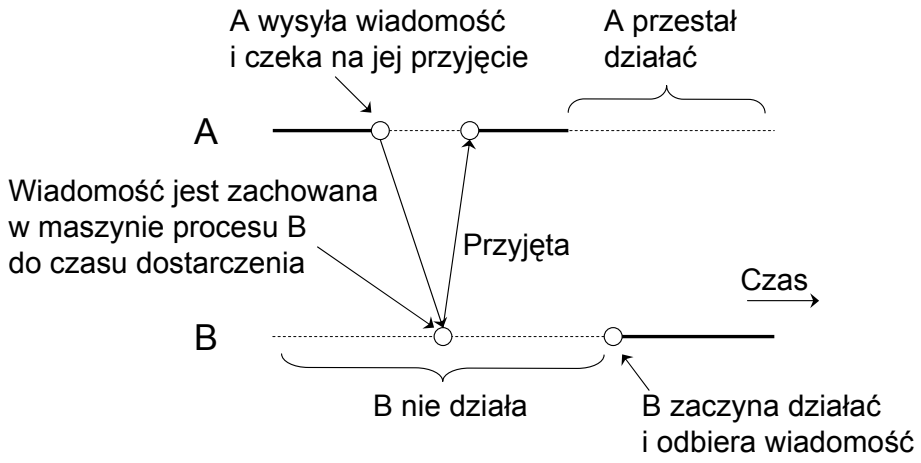
Druga linia podziału usług komunikacyjnych ma związek z czasem trwania operacji wysłania wiadomości. Cechą charakterystyczną komunikacji **asynchronicznej** jest to, że nadawca kontynuuje przetwarzanie natychmiast po przedłożeniu wysyłanej wiadomości w systemie. Wiadomość zostaje zapamiętana w komputerze wysyłającym lub w najbliższym serwerze komunikacyjnym. Nadawca nie jest zatem w ogóle blokowany w oczekiwaniu na jakiekolwiek potwierdzenie, przez co operacja wysłania trwa bardzo krótko. W komunikacji **synchronicznej** zaś nadawca jest blokowany do momentu, aż wiadomość znajdzie się w buforze komputera odbiorcy lub zostanie dostarczona do samego odbiorcy. Istnieje kilka odmian komunikacji synchronicznej, a w najsilniejszej z nich nadawca jest blokowany do momentu otrzymania potwierdzenia *przetworzenia* wiadomości przez odbiorcę. W dalszej części omówione zostały różne formy komunikacji, łączące różne poziomy trwałości i synchronizmu.

Trwała asynchroniczna



W trwałej asynchronicznej komunikacji wiadomość jest trwale zapamiętana w komputerze wysyłającym lub w najbliższym serwerze komunikacyjnym. Poczta elektroniczna jest jednym z przykładów tego typu komunikacji. Na rysunku pokazano zasadę działania trwałej komunikacji asynchronicznej. Podczas samego transferu wiadomości ani nadawca, ani odbiorca nie musi pozostawać aktywny, gdyż wiadomość jest trwale zapamiętana w systemie komunikacyjnym.

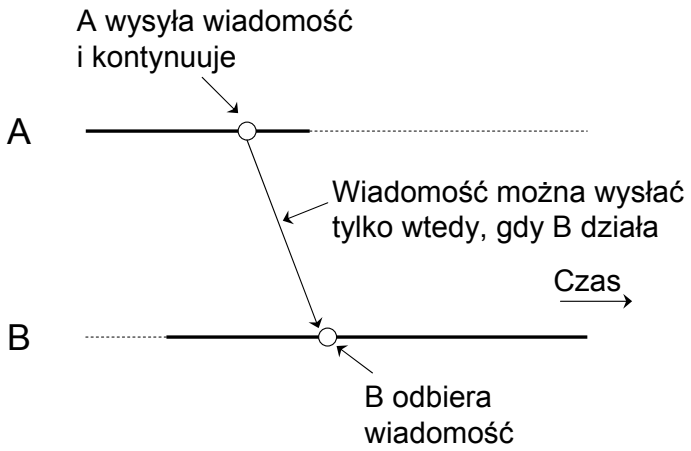
Trwała synchroniczna



Przy komunikacji trwałej synchronicznej wiadomości mogą być trwale zapamiętane tylko w komputerze odbiorcy. Nadawca jest blokowany do momentu otrzymania potwierdzenia, że wiadomość została tam zachowana. Należy podkreślić, że aplikacja odbiorcy nie musi działać w momencie przybycia wiadomości.

Słabszą odmianą trwałej synchronicznej komunikacji jest usługa, w której nadawca jest blokowany do momentu, gdy wiadomość zostanie zachowana w serwerze komunikacyjnym połączonym z komputerem odbiorcy.

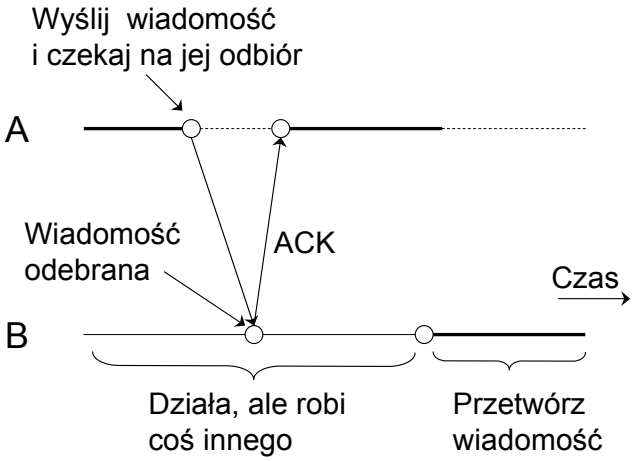
Nietrwała asynchroniczna



W komunikacji nietrwałej asynchronicznej, po przekazaniu przez nadawcę wysyłanej wiadomości systemowi komunikacyjnemu, wiadomość ta zostaje tymczasowo zapamiętana w lokalnym buforze komputera wysyłającego, a aplikacja nadawcy natychmiast kontynuuje swoją pracę. W tym samym czasie system komunikacyjny kieruje wiadomość do komputera odbiorcy, gdzie aplikacja odbiorcy powinna działać i oczekiwać na wiadomość. Jeśli odbiorca nie działa w momencie dostarczenia wiadomości, transfer się nie udaje.

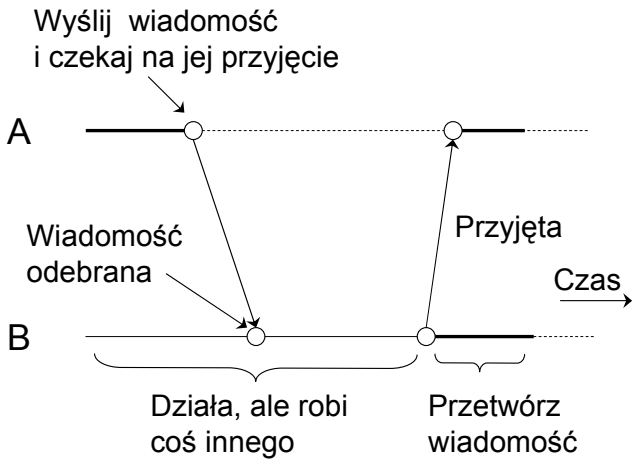
Do przykładów tego typu usługi należą: protokół UDP w warstwie transportowej oraz odmiana zdalnego wywoływania procedur, znana pod nazwą asynchronicznego RPC.

Nietrwała synchroniczna typu odbiór



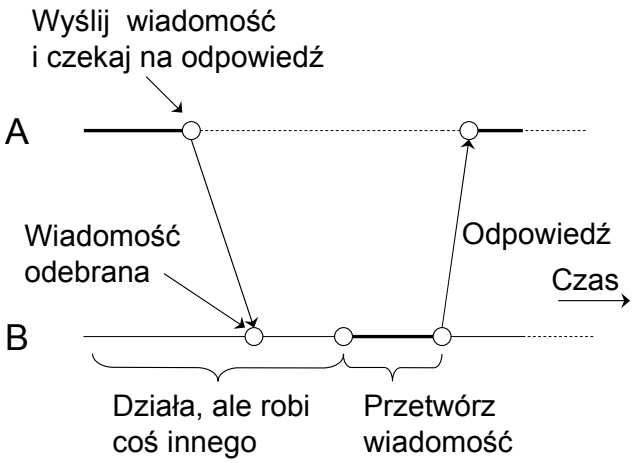
Istnieje kilka odmian komunikacji nietrwałej synchronicznej. Najmniej wymagającą z nich jest odmiana typu odbiór, to znaczy taka, w której nadawca jest blokowany do momentu otrzymania potwierdzenia, że wiadomość została zapamiętana w lokalnym buforze komputera odbiorcy. Nadawca, po otrzymaniu takiego potwierdzenia, kontynuuje pracę.

Nietrwała synchroniczna typu dostarczenie



W bardziej wymagającej odmianie komunikacji nietrwałej synchronicznej nadawca jest blokowany do momentu otrzymania potwierdzenia, że wiadomość została dostarczona do aplikacji odbiorcy. Potwierdzenie takie może zostać wysłane jeszcze przed przetworzeniem wiadomości przez odbiorcę.

Nietrwała synchroniczna typu odpowiedź



W najbardziej wymagającej odmianie komunikacji nietrwałej synchronicznej nadawca jest blokowany do momentu, aż otrzyma potwierdzenie przetworzenia wiadomości przez aplikację odbiorcy. Do najbardziej znanych usług wykorzystujących ten typ komunikacji należą RPC oraz RMI.

Komunikacja strumieniowa – definicje

- Komunikacja, w której istotne są parametry czasowe
- Medium
  - ciągłe
  - dyskretne
- Tryb transmisji
  - asynchroniczny
  - synchroniczny
  - izochroniczny
- Strumień
  - prosty
  - złożony

Dotychczas przedstawiane mechanizmy komunikacji koncentrowały się na wymianie odrębnych, kompletnych porcji informacji, na przykład wywołanie metody i późniejsza odpowiedź na wywołanie. Charakterystyczną cechą tego typu komunikacji jest to, że jej poprawność nie zależy od parametrów czasowych. Nie ma znaczenia, czy system działa za wolno lub za szybko, liczy się tylko fakt dotarcia wiadomości do celu.

Istnieją jednak takie formy komunikacji, w których parametry czasowe odgrywają kluczową rolę. Rozważmy na przykład strumień audio, składający się z ciągu małych próbek, przesyłanych od nadawcy do odbiorcy. Każda próbka strumienia reprezentuje amplitudę fali dźwiękowej. Aby poprawnie odtworzyć strumień u odbiorcy, nie wystarczy zachować oryginalnej kolejności próbek – należy utrzymać także taką samą częstotliwość ich występowania, jaka była u nadawcy. Odtwarzanie próbek z inną prędkością będzie skutkowało powstaniem niepoprawnej wersji oryginalnego dźwięku.

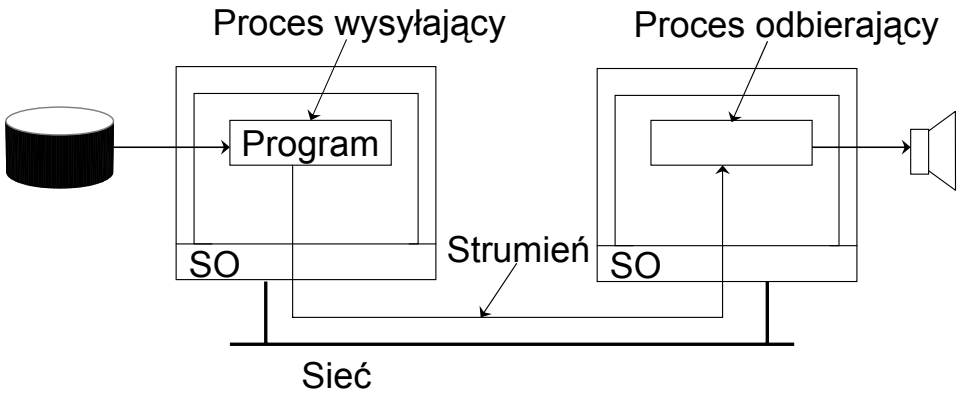
Przez **medium** będziemy rozumieli konkretny sposób prezentowania informacji, np. dźwięk, obraz nieruchomy, obraz ruchomy, tekst itd. Istnieją dwa rodzaje mediów. W **mediach ciągłych** czasowe relacje pomiędzy kolejnymi jednostkami danych mają kluczowe znaczenie dla poprawności interpretacji danych. Do przykładów mediów ciągłych można zaliczyć sygnał dźwiękowy czy obraz ruchomy. W **mediach dyskretnych** te relacje czasowe nie mają fundamentalnego znaczenia dla poprawności transmisji. Przykładami mediów dyskretnych są: tekst, obraz nieruchomy czy kod programu. Połączenie naraz kilku form prezentacji określa się mianem komunikacji multimedialnej. W dalszej części wykładu poświęcimy uwagę wyłącznie mediom ciągłym.

W sieciach komputerowych oraz systemach rozproszonych dane zależne od parametrów czasowych modeluje się za pomocą pojęcia **strumienia**. Strumień jest po prostu sekwencją jednostek danych.

Dla strumienia ciągłego, to jest strumienia istniejącego w ramach ciągłego medium, wyróżnia się różne tryby transmisji. W trybie **asynchronicznym** jednostki danych tworzące strumień przesyłane są w konkretnej kolejności, ale nie istnieją żadne dodatkowe ograniczenia czasowe. W trybie **synchronicznym** dla każdej jednostki danych strumienia wyróżnia się maksymalne opóźnienie od momentu wysłania do momentu odebrania. Najbardziej wymagającym jest tryb **izochroniczny**, w którym jednostki danych należy dostarczać w ściśle ustalonym momencie; oznacza to, że istnieje w nim zarówno górna, jak i dolna granica opóźnienia.

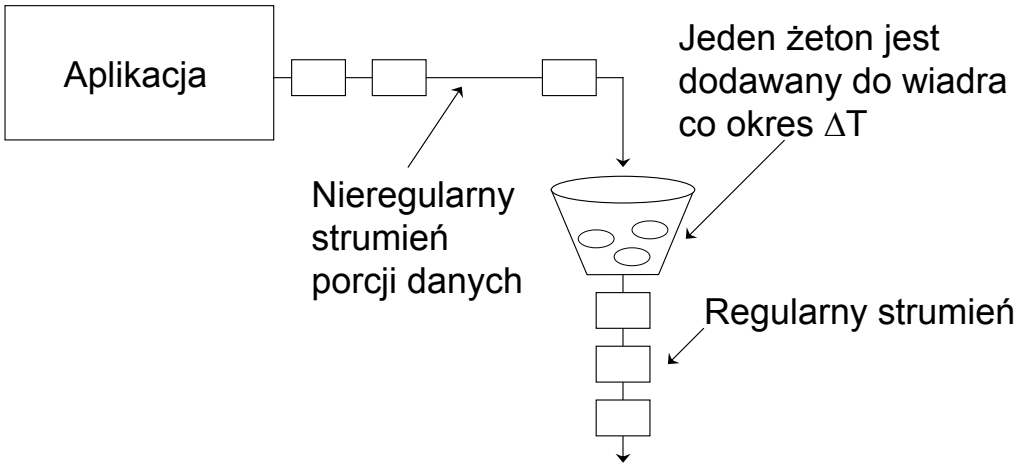
Strumienie dzielimy na **proste** i **złożone**. Strumień prosty jest złożony z jednej tylko sekwencji jednostek informacji, podczas gdy strumień złożony utworzony jest przez kilka powiązanych ze sobą strumieni prostych. Przykładem strumienia prostego jest monofoniczny sygnał audio, a przykładem strumienia złożonego – film, obejmujący obraz i dźwięk.

Architektura systemu



Na rysunku ukazano przykład architektury systemu, w którym wykorzystano komunikację strumieniową. Nadawaniem i odbieraniem zajmują się tu specjalizowane procesy, współpracujące z urządzeniami nadawczo-odbiorczymi. Możliwe jest również utworzenie strumienia bezpośrednio pomiędzy urządzeniami, jeśli tylko system operacyjny jest wyposażony w oprogramowanie potrafiące z nimi współpracować.

Jakość usług – algorytm wiadra z żetonami



Od aplikacji nadawczej, generującej strumień danych, system komunikacyjny oczekuje zachowania pewnych własności strumienia. Najczęściej stosowanym modelem do opisania takich własności jest algorytm wiadra z żetonami (ang. *Token bucket algorithm*). Celem jego zastosowania jest osiągnięcie na wyjściu aplikacji nadawczej w miarę równomiernego tempa dostarczania systemowi kolejnych jednostek danych.

Zasada algorytmu wiadra z żetonami została ukazana na rysunku. W równych odstępach czasu ΔT do wiadra o skończonej pojemności dostarczane są żetony. Każdy żeton reprezentuje stałą liczbę *k* bajtów, które aplikacja może wysłać przez sieć. Żetony są w wiadrze buforowane, a jeśli ich ilość przekroczy dostępną pojemność wiadra, są po prostu porzucane. Za każdym razem, gdy aplikacja usiłuje przesłać do sieci jednostkę danych o rozmiarze *N*, musi usunąć co najmniej *N*/*k* żetonów z wiadra. Efektem zastosowania algorytmu jest uzyskanie względnie równomiernego tempa na wyjściu aplikacji, a więc – na wejściu do systemu komunikacyjnego. Algorytm pozwala jednak na przesłanie większej porcji danych naraz, jeśli w wiadrze nagromadzonych zostało wystarczająco dużo żetonów.

Wymagania stawiane aplikacji stanowią jeden z dwóch aspektów **jakości usług** (ang. *Quality of Service*), pojęcia związanego z definiowaniem i gwarantowaniem parametrów jakościowych transmisji. Drugi jego aspekt, wymagania stawiane systemowi komunikacyjnemu przez aplikację nadawczą, omówiono na kolejnym slajdzie.

Jakość usług – specyfikacja

Charakterystyka wejścia	Wymagana usługa
• Maksymalny rozmiar jednostki danych (bajty) • Szybkość napływu żetonów do wiadra (bajty/sek) • Rozmiar wiadra (bajty) • Maksymalna szybkość przesyłania(bajty/sek)	• Podatność na straty (bajty) • Przedział strat (μsek) • Podatność na straty skokowe (jednostki danych) • Minimalne zauważalne opóźnienie (μsek) • Maksymalne wahanie (wariancja) opóźnienia (μsek) • Jakość gwarancji

Wymaganą przez obie strony jakość usługi najczęściej specyfikuje się w dokumencie nazywanym **specyfikacją przepływu** (ang. *flow specification*). Jest to kontrakt pomiędzy aplikacją a systemem komunikacyjnym, zobowiązujący obie strony do zachowania określonych parametrów jakościowych transmisji. Po jednej stronie opisana jest charakterystyka wejścia, to znaczy ruch dostarczany przez aplikację na wejście systemu komunikacyjnego. Charakterystyka wejścia jest zazwyczaj zgodna z modelem algorytmu wiadra z żetonami.

Po drugiej zaś stronie określone są wymogi stawiane systemowi komunikacyjnemu przez aplikację. Wśród najważniejszych parametrów wyróżnia się tu:

-*Podatność na straty*, która w połączeniu z *przedziałem strat* określa maksymalne akceptowalne tempo strat (przykładowo, 1 bajt na minutę)

-*Podatność na straty skokowe* - mówi, jak wiele następujących po sobie jednostek danych może zostać utraconych

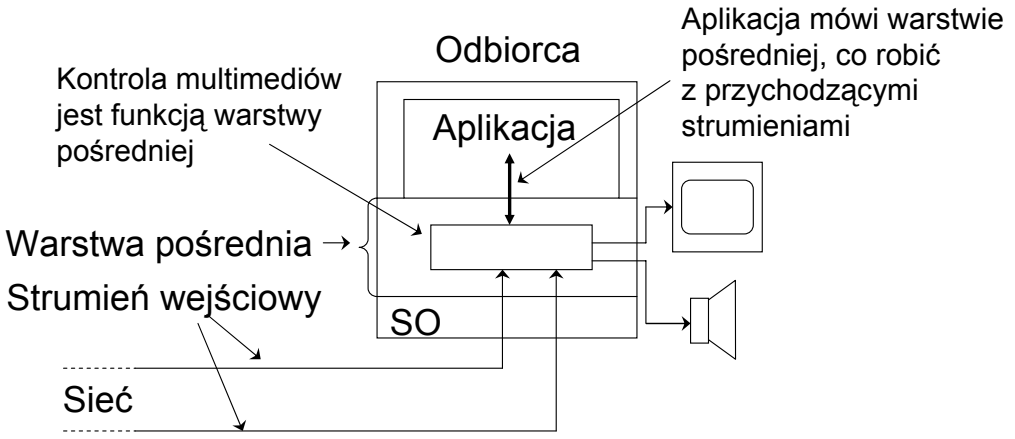
-*Minimalne zauważalne opóźnienie* - określa, jak długo sieć może opóźniać dostarczenie jednostki danych, zanim zauważy to odbiorca. Z tym parametrem wiąże się również *maksymalne wahanie opóźnienia*, określające maksymalne akceptowane wahanie opóźnienia

-Wreszcie, jakość gwarancji jest parametrem liczbowym wskazującym, jak poważnie wszystkie te wymogi powinny być traktowane. Im większa jest jego wartość, tym większe są potencjalne straty klienta na wypadek niedotrzymania przez system komunikacyjny ustalonych wartości parametrów.

Ponieważ użytkownikowi aplikacji na ogół trudno jest zaproponować specyfikację przepływu, w praktyce stosuje się podejście uproszczone. Polega ono na tym, że użytkownik najpierw po prostu klasyfikuje jakość swojego strumienia, a następnie zostaje mu zaproponowana specyfikacja usług z domyślnymi wartościami parametrów dla wskazanego przez niego poziomu jakości.

Synchronizacja strumieni

- Niskopoziomowa – w aplikacji odbierającej
- Lepsze rozwiązanie – w warstwie pośredniej



Aby informacja przenoszona przez strumień złożony była czytelna, jego strumienie proste muszą być ze sobą synchronizowane. Istnieją dwa rozwiązania takiej synchronizacji. W pierwszym odpowiada za nią w całości aplikacja odbiorcza, która w określonych odstępach czasu wysyła na wyjście odpowiednie jednostki danych ze strumieni prostych. Cały mechanizm synchronizacji jest realizowany w aplikacji. Podstawową wadą tego podejścia jest jego niskopoziomowy charakter – aplikacja musi być bardzo mocno wyspecjalizowana do obsługi konkretnego strumienia, a ponadto sama odpowiada za odczytywanie i zapisywanie kolejnych jednostek informacji (np. obrazów tworzących film).

Lepsze podejście, ukazane na rysunku, zakłada wykorzystanie do synchronizacji warstwy pośredniej, do której przeniesione zostają niskopoziomowe mechanizmy. Aplikacja otrzymuje interfejs, pozwalający jej kontrolować strumienie oraz urządzenia. Poprzez ten interfejs aplikacja określa tylko parametry czasowe strumienia, a całą resztę funkcji – głównie odczyt kolejnych jednostek informacji oraz wysłanie ich na wyjście we właściwych momentach – realizuje warstwa pośrednia. Ponieważ warstwa pośrednia zawiera wiele interfejsów dla różnych typów strumieni oraz różnych urządzeń i oferuje je aplikacji, dana aplikacja może współpracować z wieloma różnymi typami strumieni.