

Inżynieria oprogramowania 1

Architektura oprogramowania

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 20 listopada 2024

Plan wykładu

- 1 Motto
- 2 Wstęp
- 3 Strukturalizacja systemu
 - Model repozytorium
 - Model klient-serwer
 - Model maszyny abstrakcyjnej
- 4 Modele sterowania
 - Sterowanie scentralizowane
 - Model sterowania z menedżerem
 - Sterowanie zdarzeniami
- 5 Rozkład na moduły
- 6 Architektury specjalizowane

Motto

„Lekarz może pogrzebać swoje pomyłki, ale architekt może tylko doradzić klientowi by posadził winnice.”

Frank Lloyd Wright

Wstęp

Wśród ekspertów nie ma jednoznacznego stanowiska w sprawie tego, czym tak naprawdę jest architektura oprogramowania, dlatego tutaj przytoczone zostanie kilka definicji (tłumaczenia własne):

Definicja Ralpa Johnsona

Architektura oprogramowania (ang. *software architecture*) jest wspólnym poglądem doświadczonych inżynierów oprogramowania na projekt danego systemu.

<https://martinfowler.com/architecture/>

Definicja z książki „Architektura oprogramowania w praktyce”

Architektura systemu oprogramowania to zestaw struktur potrzebnych do analizowania tego systemu. Te struktury obejmują elementy oprogramowania, relacje między nimi i cechy tych elementów.

Len Bass i inni

Wstęp

Definicja Roberta C. Martina

Architektura oprogramowania systemu to kształt, który nadają mu wszystkie pracujące nad nim osoby. Kształt systemu wynika z podziału na komponenty, ułożenia tych komponentów oraz sposobów, w jakie się one ze sobą komunikują.

„Czysta architektura”

Definicja opracowana w SEI CMU

Architektura oprogramowania systemu reprezentuje decyzje projektowe związane z jego strukturą i zachowaniem.

<https://www.sei.cmu.edu/our-work/software-architecture/>

Wstęp

Definicja

Projektowanie architektoniczne jest wstępną fazą projektowania, w której wyróżnia się podsystemy, ustala schemat sterowania i sposób komunikacji podsystemów.

Zalety

Jawne projektowanie i dokumentowanie architektury oprogramowania posiada następujące zalety:

- 1 Komunikacja z interesariuszami — architektura opisuje cechy wysokopoziomowe projektu i dlatego może służyć jako punkt wyjścia do dyskusji z różnymi interesariuszami.
- 2 Analiza systemu — opracowanie we wstępnej fazie architektury projektu umożliwia wstępną analizę systemu i określenie, czy jego zakładana struktura pozwala na spełnienie stawianych przed nim wymagań niefunkcjonalnych.
- 3 Wielokrotne użycie w szerokiej skali — opis architektoniczny jest ze względu na swą wysokopoziomowość dosyć uniwersalny i może służyć jako podstawa do budowy systemów o podobnych założeniach.

Zalety

Jawne projektowanie i dokumentowanie architektury oprogramowania posiada następujące zalety:

- 1 Komunikacja z interesariuszami — architektura opisuje cechy wysokopoziomowe projektu i dlatego może służyć jako punkt wyjścia do dyskusji z różnymi interesariuszami.
- 2 Analiza systemu — opracowanie we wstępnej fazie architektury projektu umożliwia wstępną analizę systemu i określenie, czy jego zakładana struktura pozwala na spełnienie stawianych przed nim wymagań niefunkcjonalnych.
- 3 Wielokrotne użycie w szerokiej skali — opis architektoniczny jest ze względu na swą wysokopoziomowość dosyć uniwersalny i może służyć jako podstawa do budowy systemów o podobnych założeniach.

Zalety

Jawne projektowanie i dokumentowanie architektury oprogramowania posiada następujące zalety:

- 1 Komunikacja z interesariuszami — architektura opisuje cechy wysokopoziomowe projektu i dlatego może służyć jako punkt wyjścia do dyskusji z różnymi interesariuszami.
- 2 Analiza systemu — opracowanie we wstępnej fazie architektury projektu umożliwia wstępną analizę systemu i określenie, czy jego zakładana struktura pozwala na spełnienie stawianych przed nim wymagań niefunkcjonalnych.
- 3 Wielokrotne użycie w szerokiej skali — opis architektoniczny jest ze względu na swą wysokopoziomowość dosyć uniwersalny i może służyć jako podstawa do budowy systemów o podobnych założeniach.

Proces projektowania architektonicznego

W procesie projektowania architektonicznego wyróżniamy trzy podstawowe czynności:

- 1 Strukturalizacja systemu — system dzieli się na podsystemy i określa się sposób komunikacji między nimi.
- 2 Modelowanie sterowania — określa się ogólny model związków sterowania między częściami systemu.
- 3 Podział na moduły — każdy podsystem dzielony jest na moduły.

Proces projektowania architektonicznego

W procesie projektowania architektonicznego wyróżniamy trzy podstawowe czynności:

- 1 Strukturalizacja systemu — system dzieli się na podsystemy i określa się sposób komunikacji między nimi.
- 2 Modelowanie sterowania — określa się ogólny model związków sterowania między częściami systemu.
- 3 Podział na moduły — każdy podsystem dzielony jest na moduły.

Proces projektowania architektonicznego

W procesie projektowania architektonicznego wyróżniamy trzy podstawowe czynności:

- 1 Strukturalizacja systemu — system dzieli się na podsystemy i określa się sposób komunikacji między nimi.
- 2 Modelowanie sterowania — określa się ogólny model związków sterowania między częściami systemu.
- 3 Podział na moduły — każdy podsystem dzielony jest na moduły.

Podsystemy i moduły

Podsystem

Podsystem jest częścią systemu, której usługi nie zależą od innych usług, oferowanych przez inne podsystemy. Podsystemy składają się z modułów i mają określony interfejs służący do komunikowania się z innymi podsystemami. Pojedynczy podsystem może być rozpatrywany jako samodzielny system.

Moduł

Moduł jest komponentem systemu, oferującym co najmniej jedną usługę. Korzysta on z usług innego modułu i zazwyczaj nie może być rozpatrywany jako niezależny system. Pojedynczy moduł zwykle składa się z innych modułów.

Dokumentacja architektury systemu

Dokumentacja architektury systemu może składać się z następujących graficznych przedstawień modeli:

- 1 Statyczny model strukturalny obejmuje komponenty lub podsystemy, które można zbudować jako niezależne jednostki.
- 2 Model dynamiczny procesu, w którym przedstawia się podział systemu na procesy czasu wykonania.
- 3 Model interfejsów, w którym definiuje się usługi oferowane przez każdy podsystem za pośrednictwem jego interfejsu publicznego.
- 4 Model związków, który obejmuje związki, takie jak przepływ danych między podsystemami.

Najważniejszą formą dokumentacji architektury oprogramowania jest kod. Architektura *musi* być odwzorowana w kodzie.

Dokumentacja architektury systemu

Dokumentacja architektury systemu może składać się z następujących graficznych przedstawień modeli:

- 1 Statyczny model strukturalny obejmuje komponenty lub podsystemy, które można zbudować jako niezależne jednostki.
- 2 Model dynamiczny procesu, w którym przedstawia się podział systemu na procesy czasu wykonania.
- 3 Model interfejsów, w którym definiuje się usługi oferowane przez każdy podsystem za pośrednictwem jego interfejsu publicznego.
- 4 Model związków, który obejmuje związki, takie jak przepływ danych między podsystemami.

Najważniejszą formą dokumentacji architektury oprogramowania jest kod. Architektura *musi* być odwzorowana w kodzie.

Dokumentacja architektury systemu

Dokumentacja architektury systemu może składać się z następujących graficznych przedstawień modeli:

- 1 Statyczny model strukturalny obejmuje komponenty lub podsystemy, które można zbudować jako niezależne jednostki.
- 2 Model dynamiczny procesu, w którym przedstawia się podział systemu na procesy czasu wykonania.
- 3 Model interfejsów, w którym definiuje się usługi oferowane przez każdy podsystem za pośrednictwem jego interfejsu publicznego.
- 4 Model związków, który obejmuje związki, takie jak przepływ danych między podsystemami.

Najważniejszą formą dokumentacji architektury oprogramowania jest kod. Architektura *musi* być odwzorowana w kodzie.

Dokumentacja architektury systemu

Dokumentacja architektury systemu może składać się z następujących graficznych przedstawień modeli:

- 1 Statyczny model strukturalny obejmuje komponenty lub podsystemy, które można zbudować jako niezależne jednostki.
- 2 Model dynamiczny procesu, w którym przedstawia się podział systemu na procesy czasu wykonania.
- 3 Model interfejsów, w którym definiuje się usługi oferowane przez każdy podsystem za pośrednictwem jego interfejsu publicznego.
- 4 Model związków, który obejmuje związki, takie jak przepływ danych między podsystemami.

Najważniejszą formą dokumentacji architektury oprogramowania jest kod. Architektura *musi* być odwzorowana w kodzie.

Zależności

Od architektury systemu mogą zależeć następujące wymagania niefunkcjonalne:

- 1 Efektywność — najczęściej podnosi się poprzez zastosowanie do wykonywania krytycznych operacji niewielkiej liczby komponentów gruboziarnistych, które rzadko się komunikują.
- 2 Bezpieczeństwo (poufność) — zazwyczaj osiąga się poprzez zastosowanie struktury warstwowej. Najbardziej krytyczne elementy umieszcza się w warstwach wewnętrznych, gdzie należy uwzględnić wysoki poziom weryfikacji.
- 3 Bezpieczeństwo (niezawodność) operacje dotyczące bezpieczeństwa należy zamknąć w jednym podsystemie lub w niewielkiej liczbie podsystemów.
- 4 Dostępność — stosuje się komponenty redundantne.
- 5 Konserwacja — stosuje się dużą liczbę drobnoziarnistych, samodzielnych komponentów, które łatwo zmieniać.

Zależności

Od architektury systemu mogą zależeć następujące wymagania niefunkcjonalne:

- 1 Efektywność — najczęściej podnosi się poprzez zastosowanie do wykonywania krytycznych operacji niewielkiej liczby komponentów gruboziarnistych, które rzadko się komunikują.
- 2 Bezpieczeństwo (poufność) — zazwyczaj osiąga się poprzez zastosowanie struktury warstwowej. Najbardziej krytyczne elementy umieszcza się w warstwach wewnętrznych, gdzie należy uwzględnić wysoki poziom weryfikacji.
- 3 Bezpieczeństwo (niezawodność) operacje dotyczące bezpieczeństwa należy zamknąć w jednym podsystemie lub w niewielkiej liczbie podsystemów.
- 4 Dostępność — stosuje się komponenty redundantne.
- 5 Konserwacja — stosuje się dużą liczbę drobnoziarnistych, samodzielnych komponentów, które łatwo zmieniać.

Zależności

Od architektury systemu mogą zależeć następujące wymagania niefunkcjonalne:

- 1 Efektywność — najczęściej podnosi się poprzez zastosowanie do wykonywania krytycznych operacji niewielkiej liczby komponentów gruboziarnistych, które rzadko się komunikują.
- 2 Bezpieczeństwo (poufność) — zazwyczaj osiąga się poprzez zastosowanie struktury warstwowej. Najbardziej krytyczne elementy umieszcza się w warstwach wewnętrznych, gdzie należy uwzględnić wysoki poziom weryfikacji.
- 3 Bezpieczeństwo (niezawodność) operacje dotyczące bezpieczeństwa należy zamknąć w jednym podsystemie lub w niewielkiej liczbie podsystemów.
- 4 Dostępność — stosuje się komponenty redundantne.
- 5 Konserwacja — stosuje się dużą liczbę drobnoziarnistych, samodzielnych komponentów, które łatwo zmieniać.

Zależności

Od architektury systemu mogą zależeć następujące wymagania niefunkcjonalne:

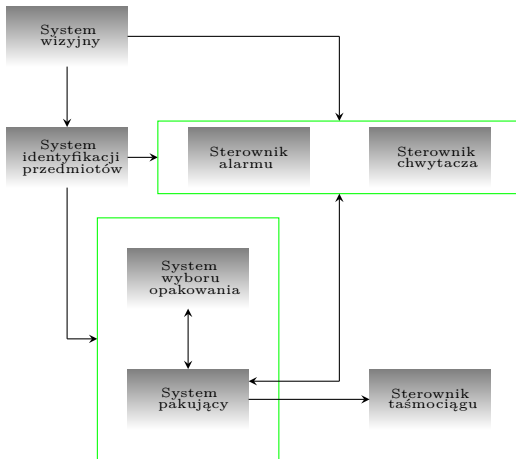
- 1 Efektywność — najczęściej podnosi się poprzez zastosowanie do wykonywania krytycznych operacji niewielkiej liczby komponentów gruboziarnistych, które rzadko się komunikują.
- 2 Bezpieczeństwo (poufność) — zazwyczaj osiąga się poprzez zastosowanie struktury warstwowej. Najbardziej krytyczne elementy umieszcza się w warstwach wewnętrznych, gdzie należy uwzględnić wysoki poziom weryfikacji.
- 3 Bezpieczeństwo (niezawodność) operacje dotyczące bezpieczeństwa należy zamknąć w jednym podsystemie lub w niewielkiej liczbie podsystemów.
- 4 Dostępność — stosuje się komponenty redundantne.
- 5 Konserwacja — stosuje się dużą liczbę drobnoziarnistych, samodzielnych komponentów, które łatwo zmieniać.

Zależności

Od architektury systemu mogą zależeć następujące wymagania niefunkcjonalne:

- 1 Efektywność — najczęściej podnosi się poprzez zastosowanie do wykonywania krytycznych operacji niewielkiej liczby komponentów gruboziarnistych, które rzadko się komunikują.
- 2 Bezpieczeństwo (poufność) — zazwyczaj osiąga się poprzez zastosowanie struktury warstwowej. Najbardziej krytyczne elementy umieszcza się w warstwach wewnętrznych, gdzie należy uwzględnić wysoki poziom weryfikacji.
- 3 Bezpieczeństwo (niezawodność) operacje dotyczące bezpieczeństwa należy zamknąć w jednym podsystemie lub w niewielkiej liczbie podsystemów.
- 4 Dostępność — stosuje się komponenty redundantne.
- 5 Konserwacja — stosuje się dużą liczbę drobnoziarnistych, samodzielnych komponentów, które łatwo zmieniać.

Przykład diagramu blokowego — system sterowania robotem



Miara niestabilności

Robert C. Martin, w swojej książce o *czystej architekturze* zaproponował wprowadzenie *miary niestabilności*, które pozwala programiście oszacować jak bardzo jest prawdopodobne to, że dany komponent ulegnie zmianom w przyszłości. By wyznaczyć tę wielkość należy znać liczbę zależności wchodzących (*fan-in*) i wychodzących *fan-out* z tego komponentu:

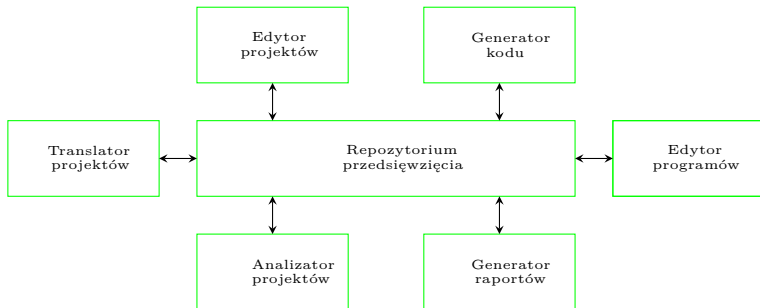
$$I = \frac{\text{Fan-in}}{\text{Fan-in} + \text{Fan-out}}$$

Im wynik jest bliższy 1 tym komponent jest bardziej niestabilny, a im bliższy 0, tym komponent jest bardziej niezależny i odpowiedzialny. Przykładowo, *System wizyjny* z poprzedniego slajdu jest maksymalnie niezależny i odpowiedzialny ($I = 0$), a *Sterownik taśmociągu* niestabilny ($I = 1$). Architektura nie może być zbudowana wyłącznie z komponentów niezależnych i odpowiedzialnych, bo nie można by jej modyfikować. Z drugiej strony architektura zbudowana wyłącznie z komponentów niestabilnych także byłaby niestabilna. W architekturze trzeba dążyć do równowagi w liczbie komponentów stabilnych i niestabilnych.

Model repozytorium

Większość systemów użytkujących duże ilości danych jest zbudowana wokół centralnej bazy danych. Taki model architektury nazywamy modelem repozytorium. Jest on przystosowany do systemów, w których dane są generowane przez jeden podsystem, a użytkowane przez inny.

Model repozytorium — przykład



Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Wady i zalety

- + Efektywny sposób współdzielenia dużych ilości danych.
 - Konieczny jest wspólny model danych repozytorium, który należy narzucić wszystkim podsystemom.
- + Podsystemy produkujące dane nie muszą zajmować się sposobem użycia tych danych przez inne podsystemy.
 - Ewolucja systemu może być trudna ze względu na narzucony model danych.
- + Scentralizowanie czynności związanych z tworzeniem kopii zapasowych, sterowanie zabezpieczeniami, itp.
 - Model repozytorium wymusza te same strategie dotyczące zabezpieczeń, sporządzania kopii zapasowych, itp.
- + Model współdzielenia jest widoczny przez repozytorium, więc integracja nowych narzędzi jest prosta, o ile obsługują ustalony model danych.
 - Wykonanie rozproszonej wersji repozytorium może być trudne.

Model klient-serwer

Główne komponenty modelu klient-serwer:

- zbiór samodzielnych serwerów oferujących usługi innym podsystemom.
- zbiór klientów korzystających z usług oferowanych przez serwery.
- sieć, która służy do komunikacji między serwerami, a klientami; nie zawsze jest konieczna.

Model klient-serwer

Główne komponenty modelu klient-serwer:

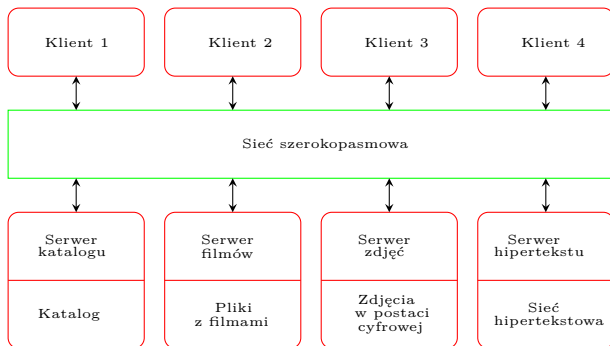
- zbiór samodzielnych serwerów oferujących usługi innym podsystemom.
- zbiór klientów korzystających z usług oferowanych przez serwery.
- sieć, która służy do komunikacji między serwerami, a klientami; nie zawsze jest konieczna.

Model klient-serwer

Główne komponenty modelu klient-serwer:

- zbiór samodzielnych serwerów oferujących usługi innym podsystemom.
- zbiór klientów korzystających z usług oferowanych przez serwery.
- sieć, która służy do komunikacji między serwerami, a klientami; nie zawsze jest konieczna.

Model klient-serwer — przykład



Wady i zalety

- + Model klient-serwer jest architekturą rozproszoną.
- Brak wspólnego modelu danych.

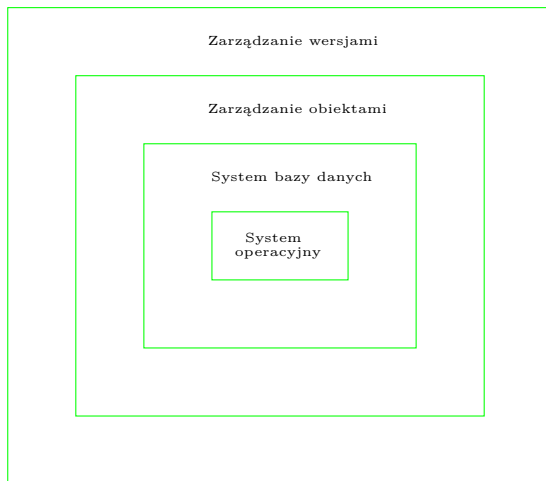
Wady i zalety

- + Model klient-serwer jest architekturą rozproszoną.
- Brak wspólnego modelu danych.

Model maszyny abstrakcyjnej

Model maszyny abstrakcyjnej (model warstwowy) opisuje sprzężanie podsystemów. System jest ułożony w stos warstw, z których każda oferuje pewne usługi. Każda warstwa jest maszyną wirtualną (abstrakcyjną), której język maszynowy (usługi oferowane przez warstwę) służy do implementacji następnego poziomu maszyny abstrakcyjnej.

Model maszyny abstrakcyjnej — przykład



Wady i zalety

- + Model warstwowy ułatwia przyrostowe tworzenie oprogramowania.
- + Architektura warstwowa jest łatwa do przenoszenia i modyfikowania.
- Podejście warstwowe jest trudne w zastosowaniu.
- Systemy oparte na „czystym” modelu warstwowym mogą być niewydajne.

Wady i zalety

- + Model warstwowy ułatwia przyrostowe tworzenie oprogramowania.
- + Architektura warstwowa jest łatwa do przenoszenia i modyfikowania.
- Podejście warstwowe jest trudne w zastosowaniu.
- Systemy oparte na „czystym” modelu warstwowym mogą być niewydajne.

Wady i zalety

- + Model warstwowy ułatwia przyrostowe tworzenie oprogramowania.
- + Architektura warstwowa jest łatwa do przenoszenia i modyfikowania.
- Podejście warstwowe jest trudne w zastosowaniu.
- Systemy oparte na „czystym” modelu warstwowym mogą być niewydajne.

Wady i zalety

- + Model warstwowy ułatwia przyrostowe tworzenie oprogramowania.
- + Architektura warstwowa jest łatwa do przenoszenia i modyfikowania.
- Podejście warstwowe jest trudne w zastosowaniu.
- Systemy oparte na „czystym” modelu warstwowym mogą być niewydajne.

Modele sterowania

Aby podsystemy pracowały jako jeden system, należy nimi sterować tak, żeby ich usługi były dostarczane we właściwe miejsce i we właściwym czasie. Wyróżnia się dwa podejścia do sterowania:

- 1 Sterowanie scentralizowane — za sterowanie odpowiada całkowicie jeden z podsystemów.
- 2 Sterowanie zdarzeniami — informacja o sterowaniu nie jest wbudowana w system, każdy z podsystemów może reagować na zdarzenia pochodzące z zewnątrz.

Modele sterowania

Aby podsystemy pracowały jako jeden system, należy nimi sterować tak, żeby ich usługi były dostarczane we właściwe miejsce i we właściwym czasie. Wyróżnia się dwa podejścia do sterowania:

- 1 Sterowanie scentralizowane — za sterowanie odpowiada całkowicie jeden z podsystemów.
- 2 Sterowanie zdarzeniami — informacja o sterowaniu nie jest wbudowana w system, każdy z podsystemów może reagować na zdarzenia pochodzące z zewnątrz.

Sterowanie scentralizowane

Rozróżniamy dwie klasy systemów sterowania scentralizowanego, w zależności od tego czy podsystemy działają współbieżnie, czy sekwencyjnie:

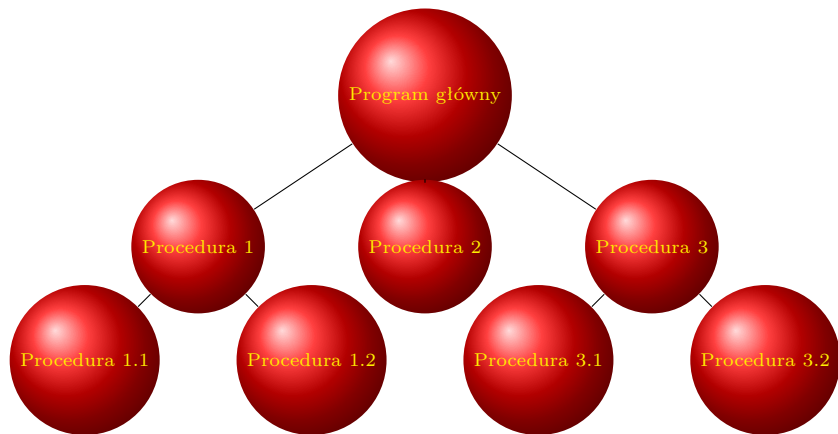
- 1 Model wywołanie-powrót — stosowany jedynie w przypadku systemów sekwencyjnych.
- 2 Model menedżera — można stosować w przypadku systemów współbieżnych. Jeden z komponentów jest wybierany do roli menadżera, systemu, który zarządza innymi procesami. Inaczej nazywany modelem pętli zdarzeń.

Sterowanie scentralizowane

Rozróżniamy dwie klasy systemów sterowania scentralizowanego, w zależności od tego czy podsystemy działają współbieżnie, czy sekwencyjnie:

- 1 Model wywołanie-powrót — stosowany jedynie w przypadku systemów sekwencyjnych.
- 2 Model menedżera — można stosować w przypadku systemów współbieżnych. Jeden z komponentów jest wybierany do roli menadżera, systemu, który zarządza innymi procesami. Inaczej nazywany modelem pętli zdarzeń.

Model sterowania wywołanie-powrót



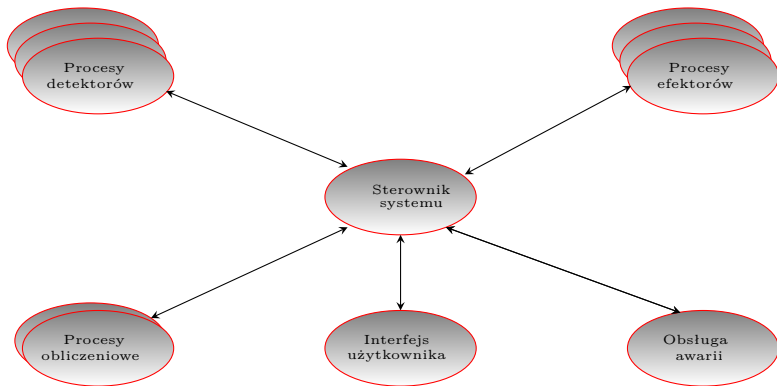
Wady i zalety

- + Łatwa analiza przepływu sterowania i deterministyczne działanie systemu.
- Utrudniona obsługa wyjątków.

Wady i zalety

- + Łatwa analiza przepływu sterowania i deterministyczne działanie systemu.
- Utrudniona obsługa wyjątków.

Scentralizowane sterowanie z menedżerem — przykład



Sterowanie zdarzeniami

Dwa podstawowe modele sterowania zdarzeniami to:

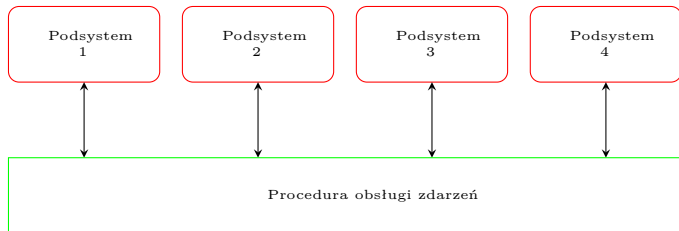
- 1 Model rozgłaszania — zdarzenie jest ogłoszeniem dla wszystkich podsystemów.
- 2 Model z przerwaniem — zewnętrzne przerwanie są wykrywane przez obsługę przerw i przekazywane do odpowiedniego komponentu, gdzie są przetwarzane.

Sterowanie zdarzeniami

Dwa podstawowe modele sterowania zdarzeniami to:

- 1 Model rozgłaszania — zdarzenie jest ogłoszeniem dla wszystkich podsystemów.
- 2 Model z przerwaniem — zewnętrzne przerwanie są wykrywane przez obsługę przerw i przekazywane do odpowiedniego komponentu, gdzie są przetwarzane.

Model rozgłaszania



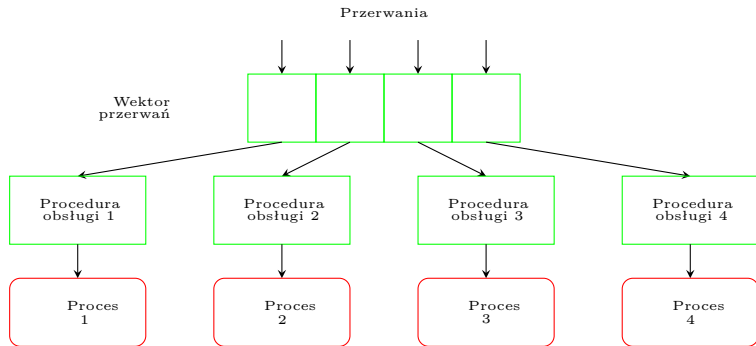
Wady i zalety

- + Prostota ewolucji.
- Brak informacji zwrotnej, co do tego, czy zdarzenie zostało obsłużone.

Wady i zalety

- + Prostota ewolucji.
- Brak informacji zwrotnej, co do tego, czy zdarzenie zostało obsłużone.

Model sterowania z przerwaniem



Wady i zalety

- + Szybkie odpowiedzi na zdarzenia.
- Złożoność programowania i trudności z zatwierdzeniem.

Wady i zalety

- + Szybkie odpowiedzi na zdarzenia.
- Złożoność programowania i trudności z zatwierdzeniem.

Rozkład na moduły

Rozkład na moduły dotyczy podsystemów. Można tego dokonać w oparciu między innymi o następujące modele:

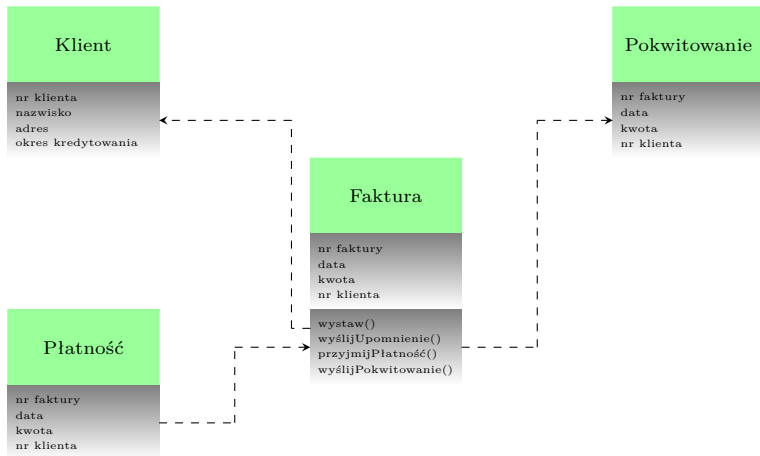
- 1 Model obiektowy — system jest dzielony na zbiór komunikujących się obiektów.
- 2 Model przepływu danych — system jest dzielony na moduły funkcjonalne, które pobierają dane wejściowe i przetwarzają je na dane wyjściowe. Model ten nosi również nazwę modelu potokowego.

Rozkład na moduły

Rozkład na moduły dotyczy podsystemów. Można tego dokonać w oparciu między innymi o następujące modele:

- 1 Model obiektowy — system jest dzielony na zbiór komunikujących się obiektów.
- 2 Model przepływu danych — system jest dzielony na moduły funkcjonalne, które pobierają dane wejściowe i przetwarzają je na dane wyjściowe. Model ten nosi również nazwę modelu potokowego.

Model obiektowy — przykład



Wady i zalety

- + Obiekty są luźno od siebie uzależnione, więc można zmieniać ich implementacje nie wpływając na pozostałe obiekty.
- + Obiekty są często reprezentantami bytów świata rzeczywistego, co ułatwia zrozumienie systemu.
- + Opracowano języki programowania, które umożliwiają bezpośrednią implementację komponentów obiektowych.
- + Model obiektowy może być zastosowany zarówno w systemach współbieżnych, jak i sekwencyjnych.
 - Konieczność jawnego używania nazw i interfejsów obiektów, które dostarczają usług.
 - Trudno ocenić wpływ zmiany interfejsu pojedynczego obiektu na pozostałe obiekty.
 - Trudno reprezentować w postaci obiektów złożone byty.

Wady i zalety

- + Obiekty są luźno od siebie uzależnione, więc można zmieniać ich implementacje nie wpływając na pozostałe obiekty.
- + Obiekty są często reprezentantami bytów świata rzeczywistego, co ułatwia zrozumienie systemu.
- + Opracowano języki programowania, które umożliwiają bezpośrednią implementację komponentów obiektowych.
- + Model obiektowy może być zastosowany zarówno w systemach współbieżnych, jak i sekwencyjnych.
 - Konieczność jawnego używania nazw i interfejsów obiektów, które dostarczają usług.
 - Trudno ocenić wpływ zmiany interfejsu pojedynczego obiektu na pozostałe obiekty.
 - Trudno reprezentować w postaci obiektów złożone byty.

Wady i zalety

- + Obiekty są luźno od siebie uzależnione, więc można zmieniać ich implementacje nie wpływając na pozostałe obiekty.
- + Obiekty są często reprezentantami bytów świata rzeczywistego, co ułatwia zrozumienie systemu.
- + Opracowano języki programowania, które umożliwiają bezpośrednią implementację komponentów obiektowych.
- + Model obiektowy może być zastosowany zarówno w systemach współbieżnych, jak i sekwencyjnych.
 - Konieczność jawnego używania nazw i interfejsów obiektów, które dostarczają usług.
 - Trudno ocenić wpływ zmiany interfejsu pojedynczego obiektu na pozostałe obiekty.
 - Trudno reprezentować w postaci obiektów złożone byty.

Wady i zalety

- + Obiekty są luźno od siebie uzależnione, więc można zmieniać ich implementacje nie wpływając na pozostałe obiekty.
- + Obiekty są często reprezentantami bytów świata rzeczywistego, co ułatwia zrozumienie systemu.
- + Opracowano języki programowania, które umożliwiają bezpośrednią implementację komponentów obiektowych.
- + Model obiektowy może być zastosowany zarówno w systemach współbieżnych, jak i sekwencyjnych.
 - Konieczność jawnego używania nazw i interfejsów obiektów, które dostarczają usług.
 - Trudno ocenić wpływ zmiany interfejsu pojedynczego obiektu na pozostałe obiekty.
 - Trudno reprezentować w postaci obiektów złożone byty.

Wady i zalety

- + Obiekty są luźno od siebie uzależnione, więc można zmieniać ich implementacje nie wpływając na pozostałe obiekty.
- + Obiekty są często reprezentantami bytów świata rzeczywistego, co ułatwia zrozumienie systemu.
- + Opracowano języki programowania, które umożliwiają bezpośrednią implementację komponentów obiektowych.
- + Model obiektowy może być zastosowany zarówno w systemach współbieżnych, jak i sekwencyjnych.
 - Konieczność jawnego używania nazw i interfejsów obiektów, które dostarczają usług.
 - Trudno ocenić wpływ zmiany interfejsu pojedynczego obiektu na pozostałe obiekty.
 - Trudno reprezentować w postaci obiektów złożone byty.

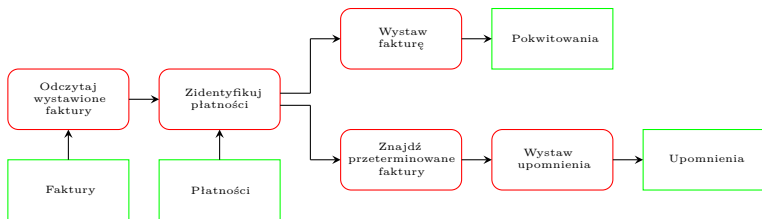
Wady i zalety

- + Obiekty są luźno od siebie uzależnione, więc można zmieniać ich implementacje nie wpływając na pozostałe obiekty.
- + Obiekty są często reprezentantami bytów świata rzeczywistego, co ułatwia zrozumienie systemu.
- + Opracowano języki programowania, które umożliwiają bezpośrednią implementację komponentów obiektowych.
- + Model obiektowy może być zastosowany zarówno w systemach współbieżnych, jak i sekwencyjnych.
 - Konieczność jawnego używania nazw i interfejsów obiektów, które dostarczają usług.
 - Trudno ocenić wpływ zmiany interfejsu pojedynczego obiektu na pozostałe obiekty.
 - Trudno reprezentować w postaci obiektów złożone byty.

Wady i zalety

- + Obiekty są luźno od siebie uzależnione, więc można zmieniać ich implementacje nie wpływając na pozostałe obiekty.
- + Obiekty są często reprezentantami bytów świata rzeczywistego, co ułatwia zrozumienie systemu.
- + Opracowano języki programowania, które umożliwiają bezpośrednią implementację komponentów obiektowych.
- + Model obiektowy może być zastosowany zarówno w systemach współbieżnych, jak i sekwencyjnych.
 - Konieczność jawnego używania nazw i interfejsów obiektów, które dostarczają usług.
 - Trudno ocenić wpływ zmiany interfejsu pojedynczego obiektu na pozostałe obiekty.
 - Trudno reprezentować w postaci obiektów złożone byty.

Model przepływu danych — przykład



Wady i zalety

- + Architektura potokowa umożliwia wielokrotne użycie przekształceń.
- + Jest intuicyjna dla wielu ludzi.
- + Ewolucja systemu polega na dodaniu nowych przekształceń i jest bardzo łatwa.
- + Jest łatwa do zaimplementowania zarówno w systemach sekwencyjnych, jak i współbieżnych.
- Konieczne jest wprowadzenie wspólnego formatu danych zrozumiałego dla wszystkich przekształceń.
- Nie nadaje się do systemów interaktywnych.

Wady i zalety

- + Architektura potokowa umożliwia wielokrotne użycie przekształceń.
- + Jest intuicyjna dla wielu ludzi.
- + Ewolucja systemu polega na dodaniu nowych przekształceń i jest bardzo łatwa.
- + Jest łatwa do zaimplementowania zarówno w systemach sekwencyjnych, jak i współbieżnych.
 - Konieczne jest wprowadzenie wspólnego formatu danych zrozumiałego dla wszystkich przekształceń.
 - Nie nadaje się do systemów interaktywnych.

Wady i zalety

- + Architektura potokowa umożliwia wielokrotne użycie przekształceń.
- + Jest intuicyjna dla wielu ludzi.
- + Ewolucja systemu polega na dodaniu nowych przekształceń i jest bardzo łatwa.
- + Jest łatwa do zaimplementowania zarówno w systemach sekwencyjnych, jak i współbieżnych.
 - Konieczne jest wprowadzenie wspólnego formatu danych zrozumiałego dla wszystkich przekształceń.
 - Nie nadaje się do systemów interaktywnych.

Wady i zalety

- + Architektura potokowa umożliwia wielokrotne użycie przekształceń.
- + Jest intuicyjna dla wielu ludzi.
- + Ewolucja systemu polega na dodaniu nowych przekształceń i jest bardzo łatwa.
- + Jest łatwa do zaimplementowania zarówno w systemach sekwencyjnych, jak i współbieżnych.
 - Konieczne jest wprowadzenie wspólnego formatu danych zrozumiałego dla wszystkich przekształceń.
 - Nie nadaje się do systemów interaktywnych.

Wady i zalety

- + Architektura potokowa umożliwia wielokrotne użycie przekształceń.
- + Jest intuicyjna dla wielu ludzi.
- + Ewolucja systemu polega na dodaniu nowych przekształceń i jest bardzo łatwa.
- + Jest łatwa do zaimplementowania zarówno w systemach sekwencyjnych, jak i współbieżnych.
- Konieczne jest wprowadzenie wspólnego formatu danych zrozumiałego dla wszystkich przekształceń.
- Nie nadaje się do systemów interaktywnych.

Wady i zalety

- + Architektura potokowa umożliwia wielokrotne użycie przekształceń.
- + Jest intuicyjna dla wielu ludzi.
- + Ewolucja systemu polega na dodaniu nowych przekształceń i jest bardzo łatwa.
- + Jest łatwa do zaimplementowania zarówno w systemach sekwencyjnych, jak i współbieżnych.
 - Konieczne jest wprowadzenie wspólnego formatu danych zrozumiałego dla wszystkich przekształceń.
 - Nie nadaje się do systemów interaktywnych.

Architektury charakterystyczne dla różnych dziedzin

Istnieją architektury wspólne dla pewnych konkretnych dziedzin zastosowań. Ich egzemplarze różnią się w szczegółach, ale można używać wielokrotnie wspólnej struktury architektonicznej do budowy nowych systemów. Te architektury można podzielić na:

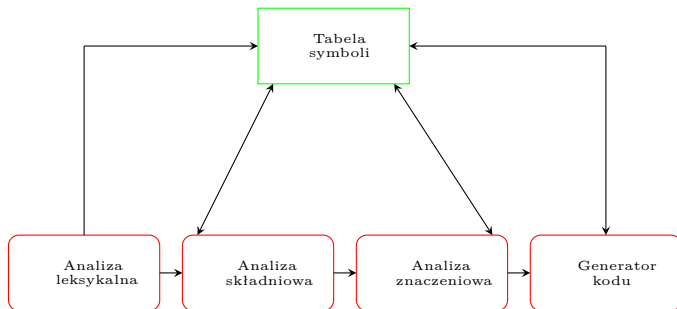
- 1 Modele ogólne — budowane metodą wstępującą, obejmują zasadnicze charakterystyki rzeczywistych systemów.
- 2 Modele odniesienia — budowane metodą zstępującą, są jeszcze bardziej abstrakcyjne niż modele ogólne. Są sposobem informowania projektantów o ogólnej strukturze systemów danej klasy.

Architektury charakterystyczne dla różnych dziedzin

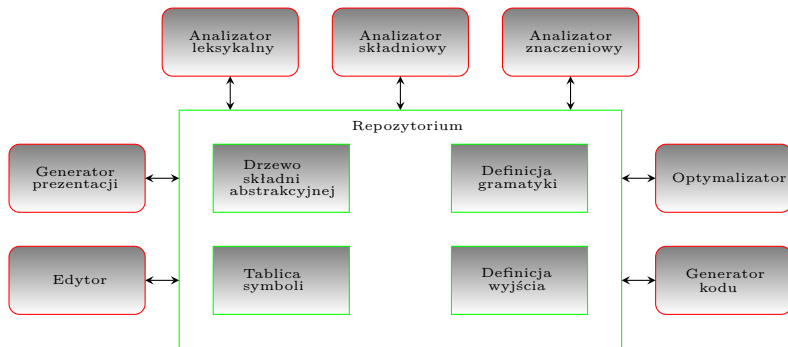
Istnieją architektury wspólne dla pewnych konkretnych dziedzin zastosowań. Ich egzemplarze różnią się w szczegółach, ale można używać wielokrotnie wspólnej struktury architektonicznej do budowy nowych systemów. Te architektury można podzielić na:

- 1 Modele ogólne — budowane metodą wstępującą, obejmują zasadnicze charakterystyki rzeczywistych systemów.
- 2 Modele odniesienia — budowane metodą zstępującą, są jeszcze bardziej abstrakcyjne niż modele ogólne. Są sposobem informowania projektantów o ogólnej strukturze systemów danej klasy.

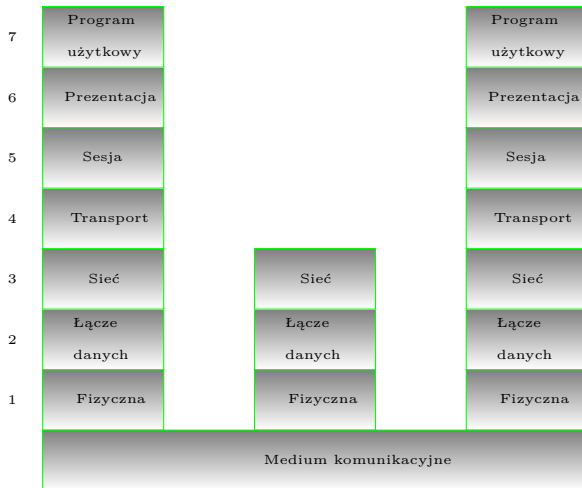
Model ogólny kompilatora — przepływ danych



Model ogólny kompilatora — repozytorium



Model odniesienia — przykład



Pytania

?

Koniec

Dziękuję Państwu za uwagę.