Język modelowania danych UML

Ewa Stemposz

{ewag@ipipan.waw.pl, ewag@pjwstk.waw.pl}

Instytut Podstaw Informatyki PAN
Polsko-Japońska Wyższa Szkoła Technik Komputerowych

Zagadnienia

Krótka charakterystyka UML
Diagramy przypadków użycia
Diagramy klas
Diagramy dynamiczne

- **interakcji**
- **stanu**
- **aktywności**

Diagramy implementacyjne

- **diagramy komponentów**
- **diagramy wdrożeniowe**

Diagramy pakietów
Mechanizmy rozszerzalności
Podsumowanie

Unified Modeling Language (UML)

“UML jest językiem ogólnego przeznaczenia do specyfikacji, konstrukcji, wizualizacji i dokumentowania wytworów związanych z systemami intensywnie wykorzystującymi oprogramowanie”.

UML 0.8-0.9 styczeń 1995 - wrzesień 1996
UML 1.0 styczeń 1997, przesłany do OMG
UML 1.1 koniec 1997, zatwierdzony jako składnik standardu OMG
UML 1.3 kwiecień 1999 (mówi się o wersji 1.4, ale brak danych)

Połączone siły trzech znanych metodologów oprogramowania:

Grady Booch **Ivar Jacobson** **James Rumbaugh**

Zalety i wady poprzedników UML

Każda z metodyk, poprzedników UML, posiada swoje zalety i wady.

- **OMT (Rumbaugh)**: dobry do modelowania dziedziny przedmiotowej. Nie przykrywa dostatecznie dokładnie zarówno aspektu użytkowników systemu, jak i aspektu implementacji.

- **OOSE (Jacobson)**: dobrze podchodzi do kwestii modelowania użytkowników i cyklu życiowego systemu. Nie przykrywa dokładnie modelowania dziedziny przedmiotowej, jak i aspektu implementacji.

- **OOAD (Booch):** dobrze podchodzi do kwestii projektowania, implementacji oraz związków ze środowiskiem implementacji. Nie przykrywa dostatecznie dobrze fazy rozpoznawania i analizy wymagań użytkowników.

Istnieje wiele aspektów systemów, które pozostały właściwie nie przykryte przez żadne z wyżej wymienionych podejść, np. przystosowanie notacji do preferencji projektantów. Celem UML było przykrycie również tych aspektów.

Nowe elementy wprowadzone w UML

- **Jasne odróżnienie klasy, typu** i **wystąpienia klasy**
- **Uszczegółowienia** (*refinements*) dla objęcia związków pomiędzy elementami o tej samej semantyce, a różnym poziomie abstrakcji
- **Odpowiedzialności** (*responsibilities*)
- **Kompozycja** jako silniejsza forma agregacji
- **Interfejsy** i **zależności (***dependencies)* typu: dostawca i klient pewnej informacji występujące między elementami modeli
- **Współbieżność**
- **Wzorce / współpraca**
- **Diagramy aktywności** (dla reinżynierii procesów biznesowych)
- **Komponenty**
- **Pakiety**
- **Mechanizmy rozszerzalności**: stereotypy (stereotypes), wartości etykietowane (*tagged values*), ograniczenia (*constraints*)

Diagramy definiowane w UML

Diagramy przypadków użycia

Diagramy klas

Diagramy dynamiczne:

- Diagramy interakcji:
  - Diagramy sekwencji
  - Diagramy współpracy (kolaboracji)
- Diagramy stanów
- Diagramy aktywności

Diagramy implementacyjne:

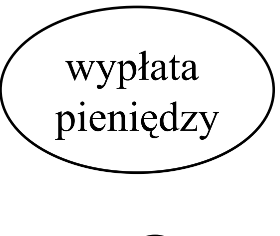
- Diagramy komponentów
- Diagramy wdrożeniowe

Diagramy pakietów

Diagramy te zapewniają uzyskanie wielu perspektyw projektowanego systemu w trakcie jego budowy.

Diagramy przypadków użycia (1)

wypłata
pieniędzy



klient

weryfikacja
klienta

«include»

System obsługi klienta

wnętrze systemu

Przypadek użycia: Powinien mieć unikalną nazwę, opisującą przypadek użycia z punktu widzenia jego zasadniczych celów. Czy lepiej jest stosować nazwę opisującą czynność (“wypłata pieniędzy”) czy polecenie (“wypłać pieniądze”) - zdania są podzielone.

Aktor: Powinien mieć unikalną nazwę.

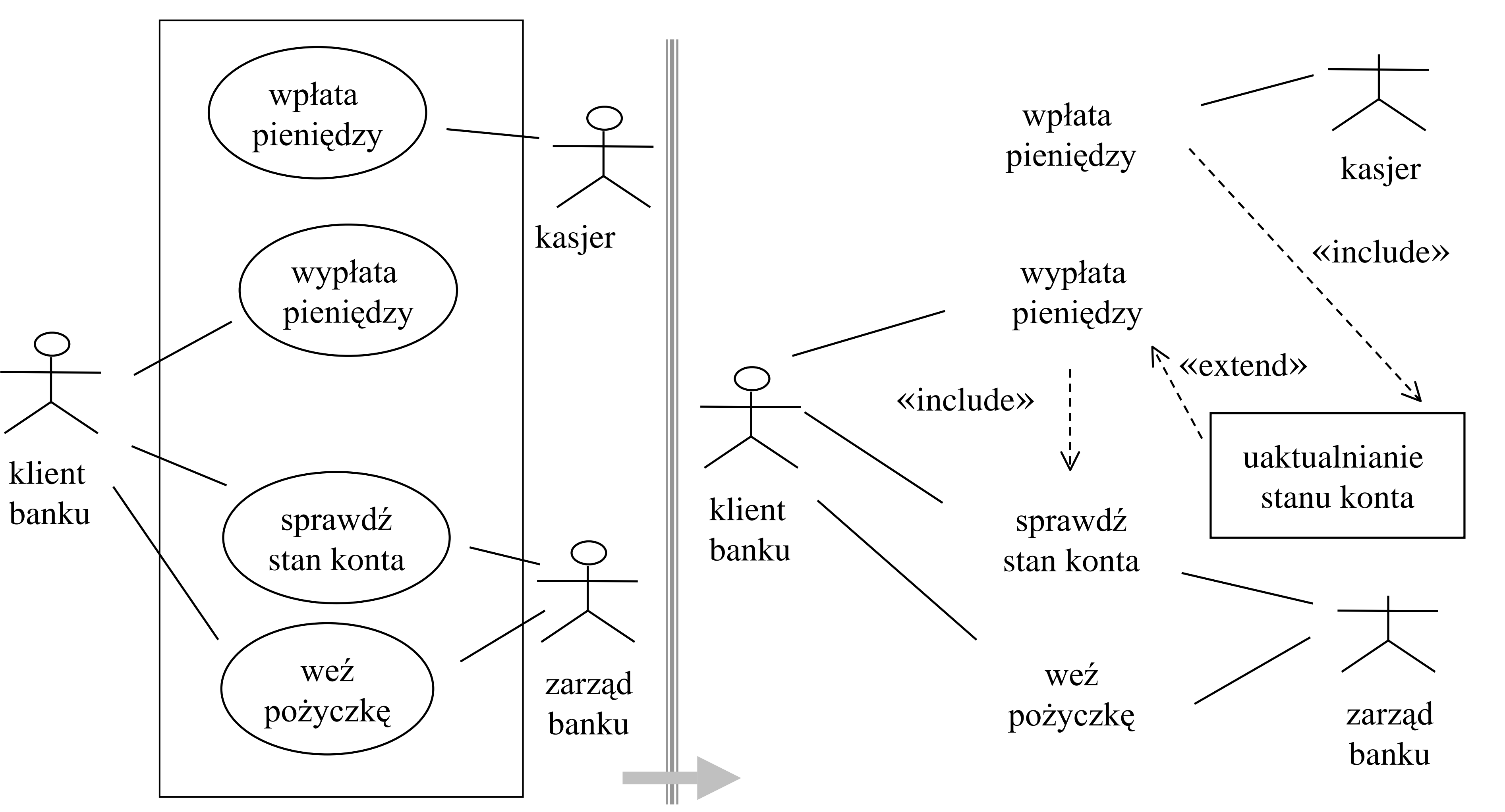
Interakcja: Pokazuje interakcję pomiędzy przypadkiem użycia a aktorem.

Blok ponownego użycia: Pokazuje fragment systemu, który jest używany przez kilka przypadków użycia; może być oznaczony jako samodzielny przypadek użycia.

Relacja typu «**include**» lub «**extend**»**:** Pokazuje związek zachodzący między dwoma przypadkami użycia lub przypadkiem użycia a blokiem ponownego użycia.

Nazwa systemu wraz z otoczeniem systemu: Pokazuje granicę pomiędzy systemem a jego otoczeniem.

Diagramy przypadków użycia (2)



Główne zadanie modelu przypadków użycia to **prawidłowe określenie wymagań funkcjonalnych** na projektowany system.

Klasa; oznaczenia (1)

Cztery pola: nazwy, atrybutów, metod i czwarte pole, którego zawartość zależy od użytkownika, np. może to być lista odpowiedzialności danej klasy.

Możliwe są różne poziomy szczegółowości.

Okno

Okno
rozmiar czy_widoczne

Okno
rozmiar czy_widoczne
wyświetl() schowaj()

Okno
rozmiar: Obszar czy_widoczne: Boolean
wyświetl() schowaj()

Pole nazwy klasy:

stereotyp nazwa_ klasy lista_wart_etyk

Pole atrybutów:

stereotyp dostępność nazwa_atrybutu : typ = wart_początkowa lista_wart_etyk

Pole metod:

stereotyp dostępność nazwa_metody (lista_arg) : typ_wart_zwracanej lista_wart_etykt

Klasa; oznaczenia (2)

gdzie:

dostępność jest określana przez trzy symbole**:**

- \+ publiczna
- \- prywatna
- \# chroniona

lista_arg: rodzaj nazwa_arg : typ = wart_początkowa

rodzaj definiuje sposób, w jaki metoda korzysta z danego argumentu**:**

in: metoda może czytać argument, ale nie może go zmieniać
out: może zmieniać, nie może czytać
inout: może czytać i zmieniać

Wszystkie elementy specyfikacji klasy za wyjątkiem nazwy klasy są opcjonalne. Nazwa klasy to z reguły rzeczownik w liczbie pojedynczej.

Przykłady klas

Okno
{*abstrakcyjna,*
autor=Kowalski
status=przetestowane}

+rozmiar: Obszar = (100,100)
#czy_widoczne: Boolean = false
+rozmiar_domyślny: Prostokąt
#rozmiar_maksymalny: Prostokąt
-xwskaźnik: XWindow*

+wyświetl()
+schowaj()
+<u>utwórz()</u>
-dołączXWindow(xwin: XWindow*)

«trwała» **Prostokąt**

punkt1: Punkt
punkt2: Punkt

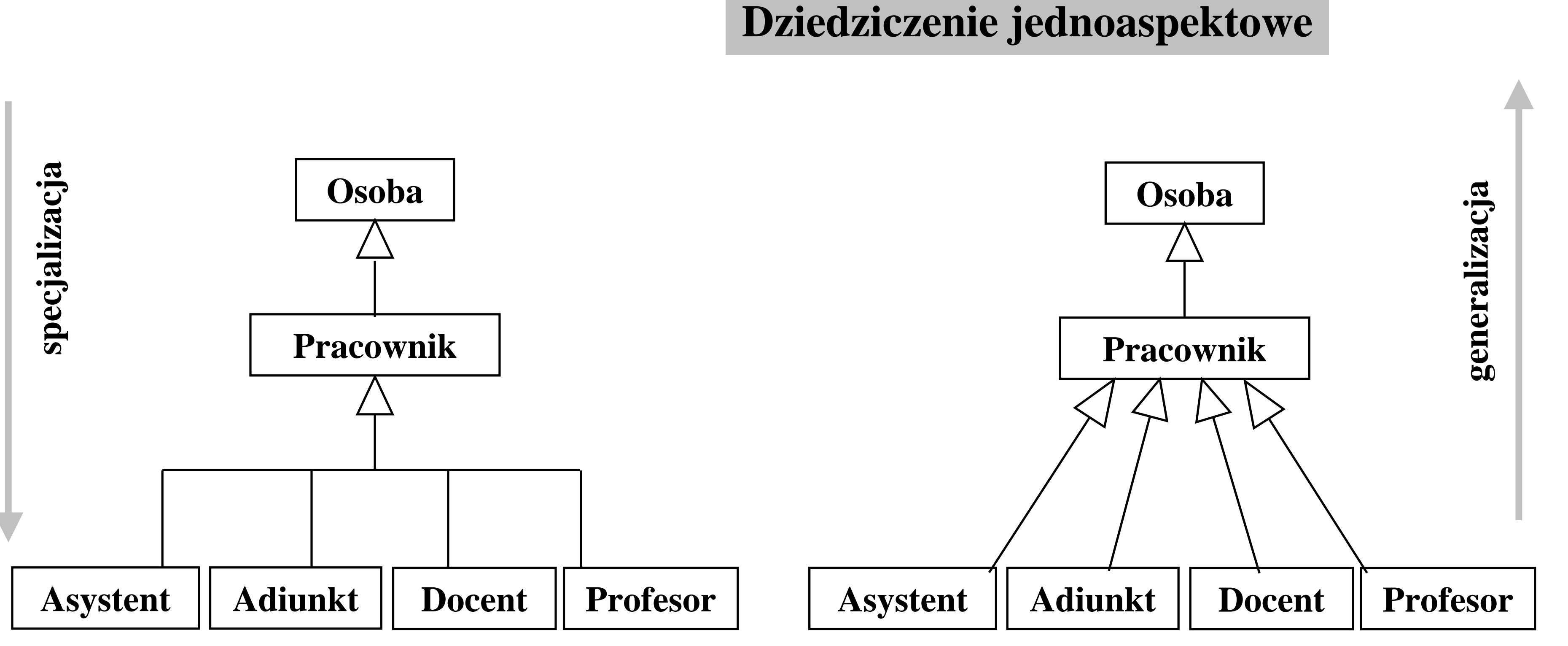
«konstruktor»
Prostokąt (p1: Punkt, p2: Punkt)

«zapytania»
obszar (): Real
aspekt(): Real
.
.
.

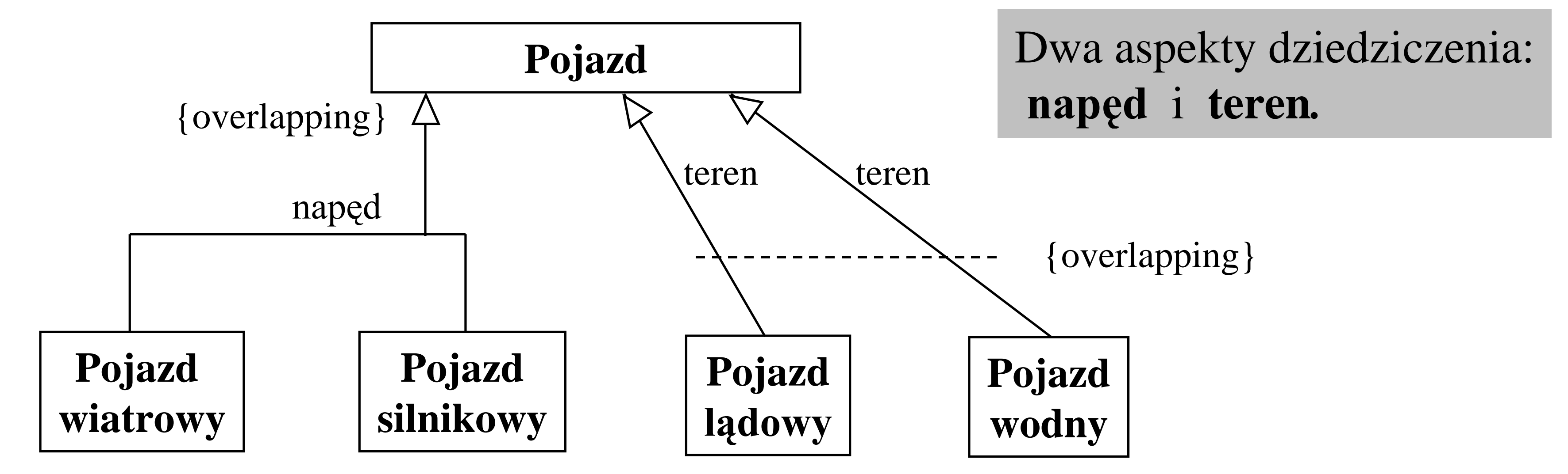
«aktualizacje»
przesuń (delta: Punkt)
przeskaluj(współczynnik: Real)

Dziedziczenie

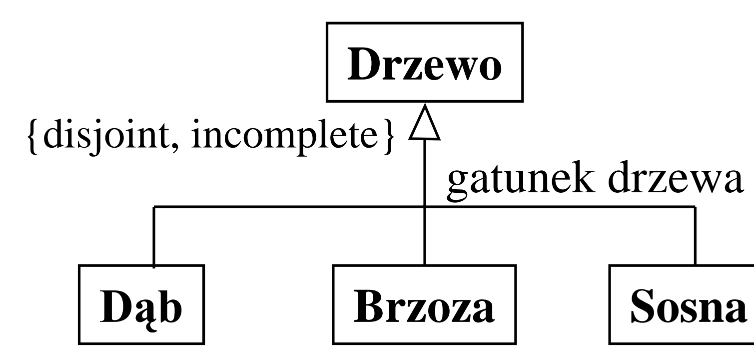
Dziedziczenie pozwala na tworzenie drzewa klas lub innych struktur bez pętli.



Dziedziczenie wieloaspektowe



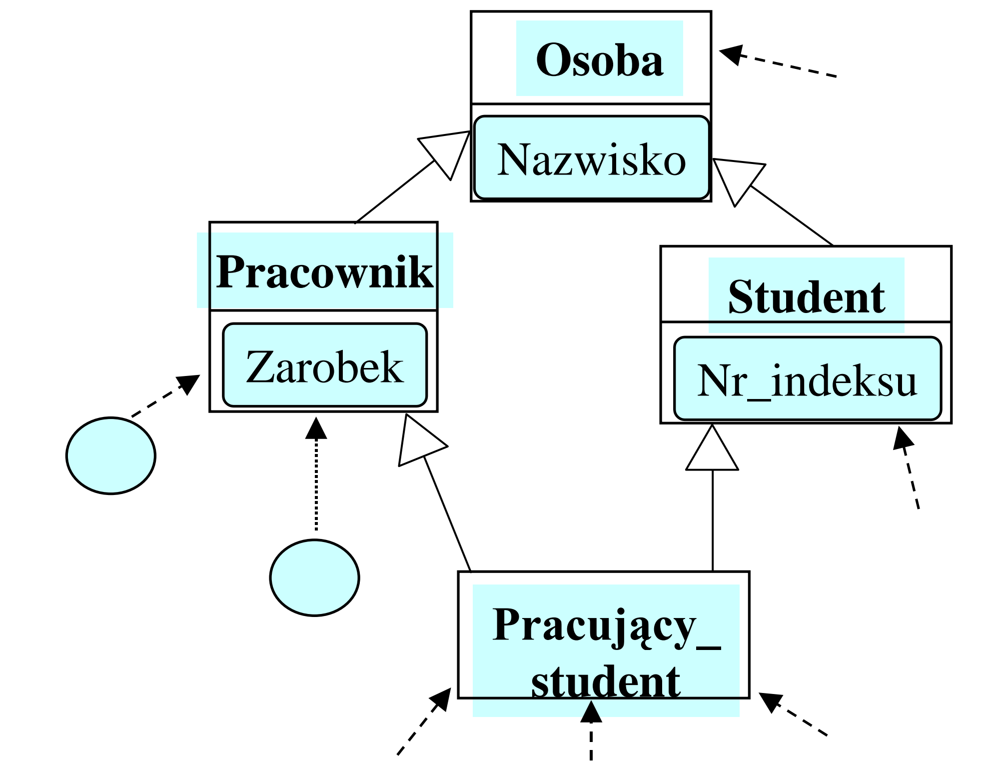
Dwa aspekty dziedziczenia: **napęd** i **teren.**



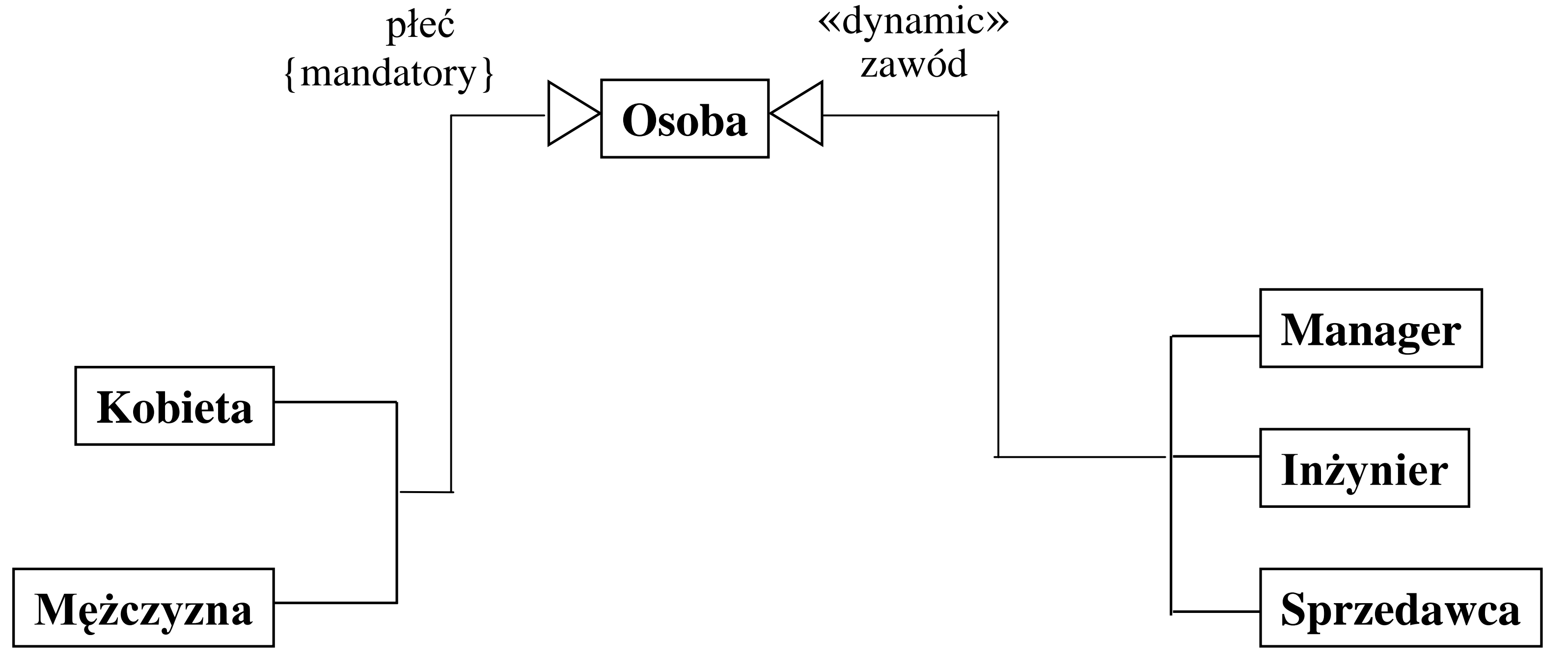
disjont (domyślne) - podział rozłączny
overlapping - podział nierozłączny; przecięcie zbiorów obiektów klas, np. *Pojazd lądowy* i *Pojazd wodny,* nie jest zbiorem pustym;
complete (domyślne) - podział całkowity
incomplete - niektóre klasy, np. nieistotne dla rozważanego problemu, zostały pominięte

Dziedziczenie wielokrotne

Dziedziczenie wielokrotne (wielodziedziczenie) ma miejsce, gdy klasa dziedziczy inwarianty z więcej niż jednej bezpośredniej nadklasy.



Dziedziczenie dynamiczne

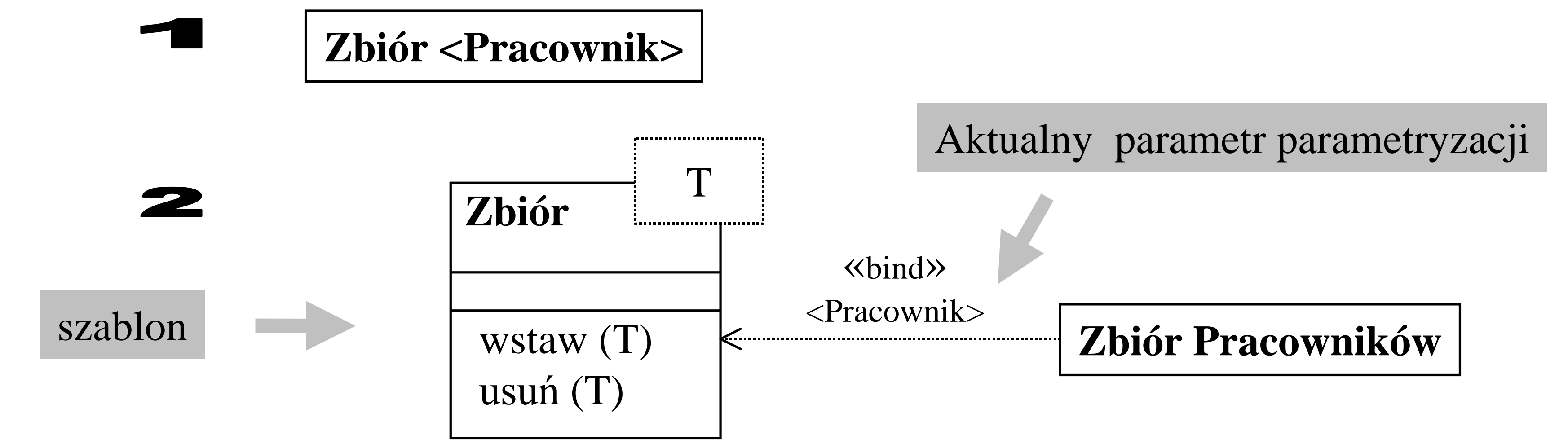


Dyskryminator (aspekt specjalizacji) ***zawód*** został tu opatrzony stereotypem «dynamic».

Klasa parametryzowana

Klasy parametryzowane są użyteczne z dwóch zasadniczych powodów: podnoszą poziom abstrakcji i wpływają na zmniejszenie długości kodu źródłowego programu.

Klasa parametryzowana może być wstawiana do diagramów UML na dwa sposoby:



Podstawowe zastosowanie klas parametryzowanych polega na wykorzystaniu ich do definiowania zbiorów (szerzej - kolekcji). Każdy obiekt klasy ***Zbiór <Pracownik>***, czy analogicznie ***Zbiór Pracowników***, jest zbiorem.

Rozszerzenia i ograniczenia w podklasie

- Podklasa nie może omijać lub zmieniać atrybutów nadklasy.
- Podklasa może zmienić ciało metody z nadklasy, ale bez zmiany jej specyfikacji.
- Podklasa może dowolnie dodawać nowe atrybuty i metody (rozszerzać zbiór własności nadklasy).
- Podklasa może ograniczać wartości atrybutów. Np. ***Koło*** jest podklasą klasy ***Elipsa***, gdzie obie średnice elipsy są sobie równe. Ograniczenia mogą spowodować, że część metod przestanie być poprawna. Np. zmiana jednej ze średnic obiektu - dozwolona dla obiektu klasy ***Elipsa*** - jest niedopuszczalna w obiekcie podklasy ***Koło***, gdyż muszą tam być zmieniane obie średnice jednocześnie.

Wystąpienie klasy

Pojęcie **wystąpienie klasy** (**instancja klasy**) oznacza obiekt, który jest "podłączony" do danej klasy, jest jej członkiem.

Wystąpienia mogą być: **bezpośrednie** i **pośrednie**.

Obiekt jest wystąpieniem bezpośrednim swojej klasy i wystąpieniem pośrednim wszystkich jej nadklas.

Możliwe oznaczenia na wystąpienie klasy

nazwa_obiektu : nazwa_klasy
nazwa_atrybutu = wart_atrybutu
...

: nazwa_klasy
nazwa_atrybutu = wart_atrybutu
...

nazwa_obiektu : nazwa_klasy

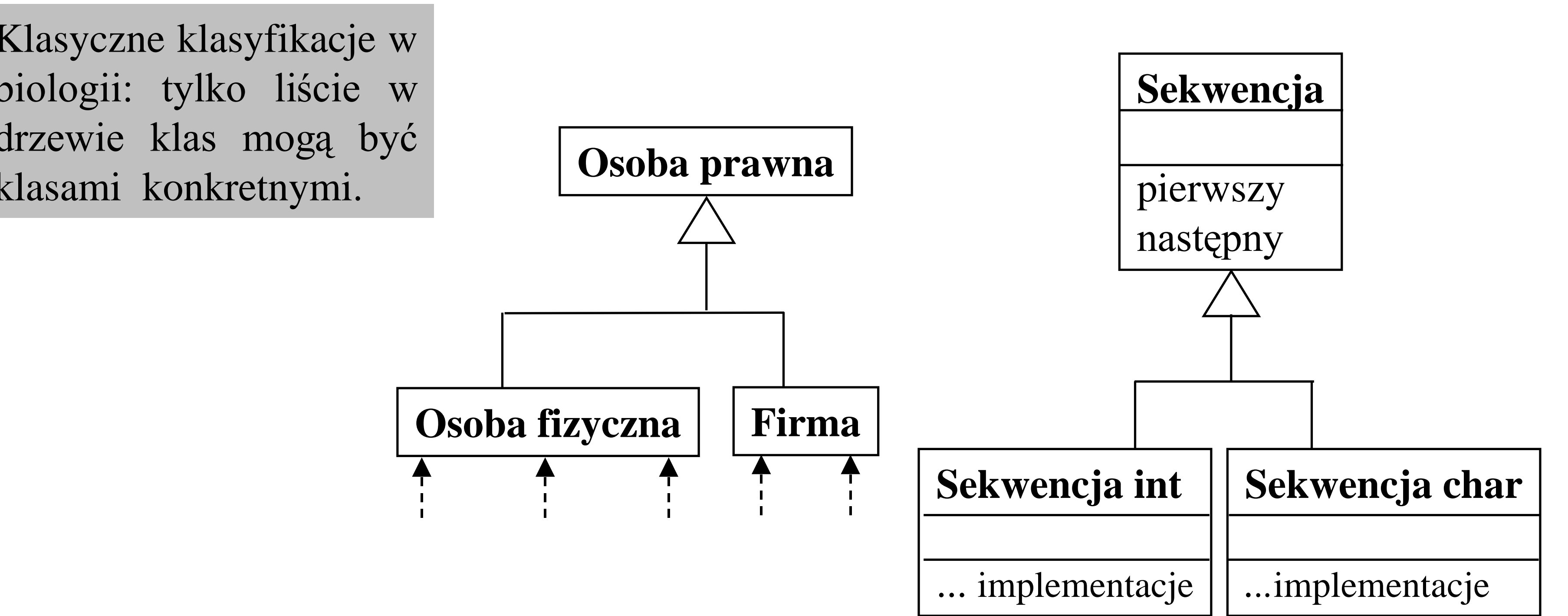
: nazwa_klasy

Klasa abstrakcyjna a klasa konkretna

Klasa abstrakcyjna nie ma (nie może mieć) bezpośrednich wystąpień i służy wyłącznie jako nadklasa dla innych klas. Stanowi jakby wspólną część definicji grupy klas o podobnej semantyce.

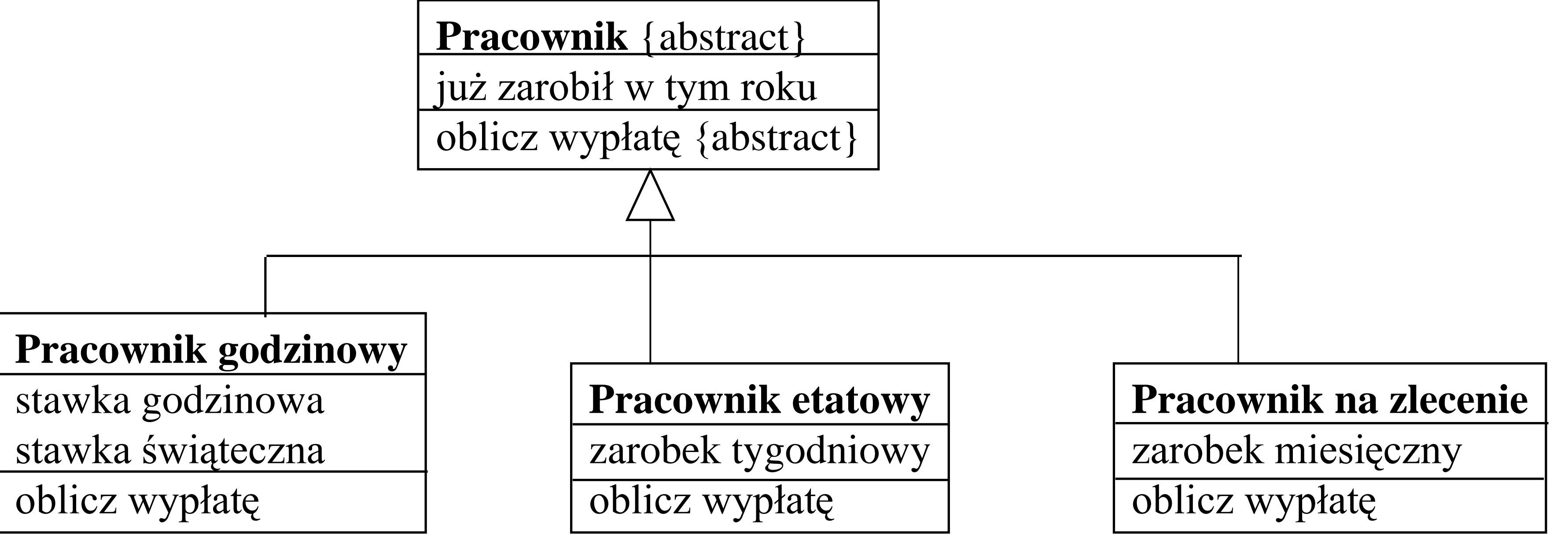
Klasa konkretna może mieć (ma prawo mieć) wystąpienia bezpośrednie.

Klasyczne klasyfikacje w biologii: tylko liście w drzewie klas mogą być klasami konkretnymi.



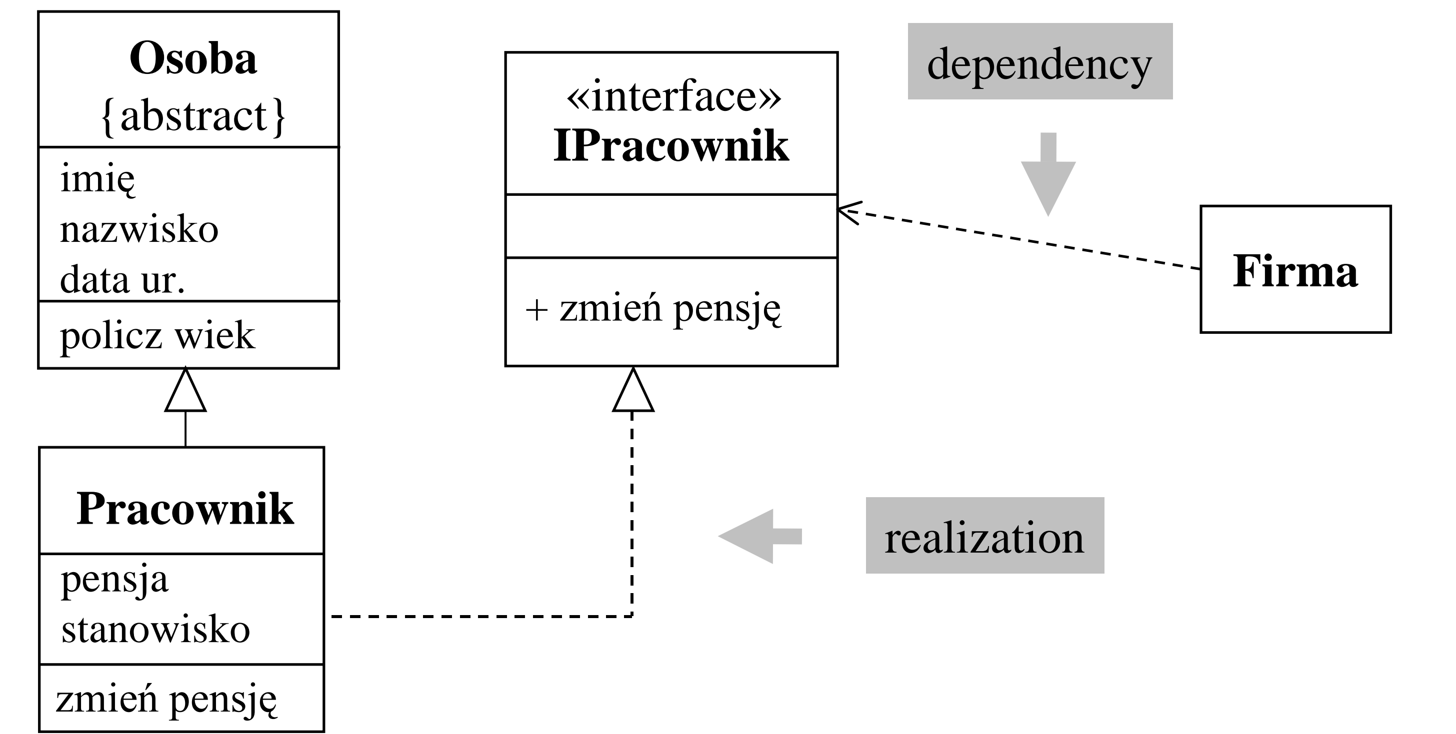
Metoda abstrakcyjna

Metoda abstrakcyjna jest to metoda wyspecyfikowana w nadklasie, której implementacja musi znaleźć się w którejś z podklas.



Klasa abstrakcyjna **może** zawierać abstrakcyjne metody, ale **nie musi** . Klasa konkretna **musi** zawierać implementacje tych metod abstrakcyjnych, które nie zostały zaimplementowane w żadnej z nadklas danej klasy konkretnej.

Interfejs, realizacja, zależność (1)

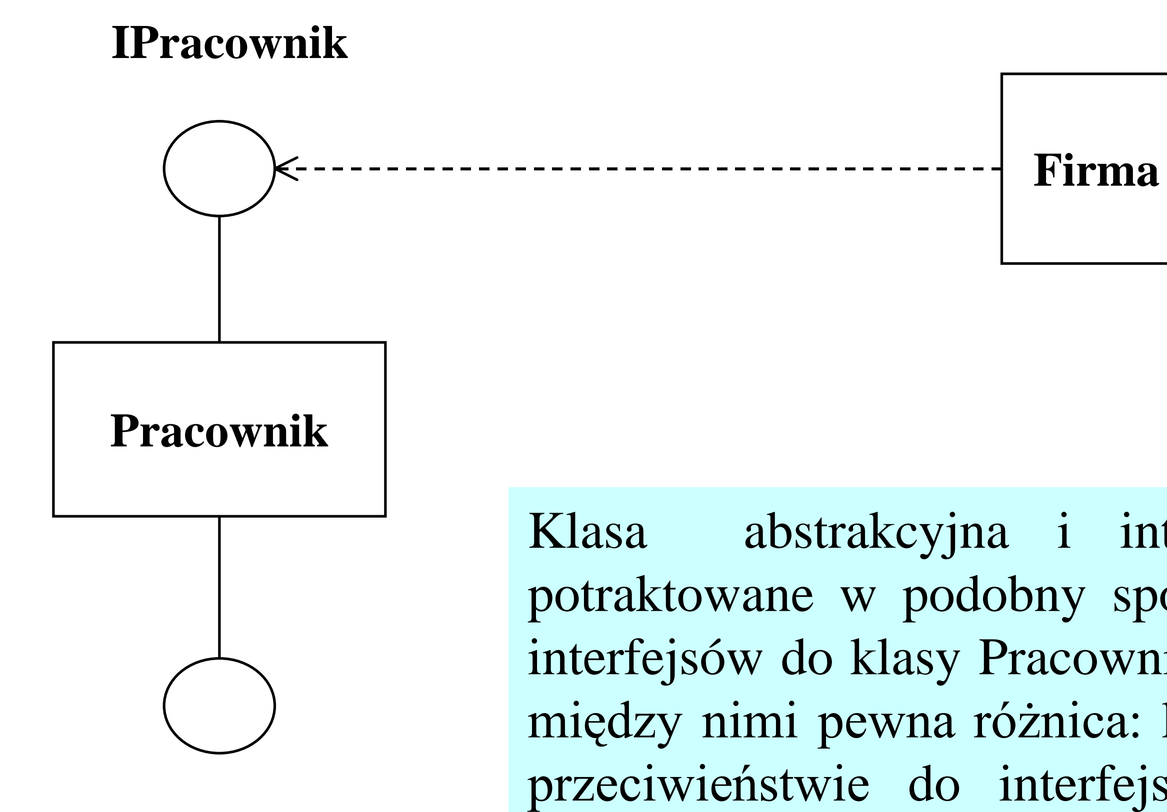


Stereotyp «interface» poprzedza nazwę klasy, która zawiera jedynie specyfikacje metod, bez implementacji. W UML interfejs nie zawiera atrybutów. Wszystkie metody są tu publiczne. Implementacje metod wyspecyfikowanych w interfejsie ***IPracownik*** zawiera klasa ***Pracownik,*** co oznaczane jest za pomocą relacji typu **realizacja (realization),** o notacji podobnej do dziedziczenia.

Zależność (**dependency**) opisuje relację typu klient-dostawca pewnej informacji.

Interfejs, zależność (2)

Dla poprzedniego diagramu można zastosować inne, bardziej zwięzłe oznaczenie.



Klasa abstrakcyjna i interfejs zostały tu potraktowane w podobny sposób - jako definicje interfejsów do klasy Pracownik. Jednakże, istnieje między nimi pewna różnica: klasa abstrakcyjna, w przeciwieństwie do interfejsu, może zawierać atrybuty i implementacje metod.

Ekstensja klasy

Ekstensja klasy (*class extent*) = aktualny (zmienny w czasie) zbiór wszystkich wystąpień tej klasy. Ekstensja klasy w implementacji oznacza specjalną strukturę danych, konkretny byt programistyczny dołączony do klasy. Ta struktura przechowuje wszystkie obiekty będące członkami danej klasy.

Niektóre metody wyspecyfikowane w danej klasie odnoszą się do jej **wystąpień**:

pracownik.*wiek*   ***pracownik.****zwolnij*   ***KONTO****.Oblicz_procent*

Niektóre metody wyspecyfikowane w danej klasie odnoszą się do jej **ekstensji**:

KL_pracownik.*nowy*   ***KL_pracownik.****zlicz*   ***KL_KONTO.******Oblicz_sumę***

Klasa może mieć nie jedną lecz wiele ekstensji.

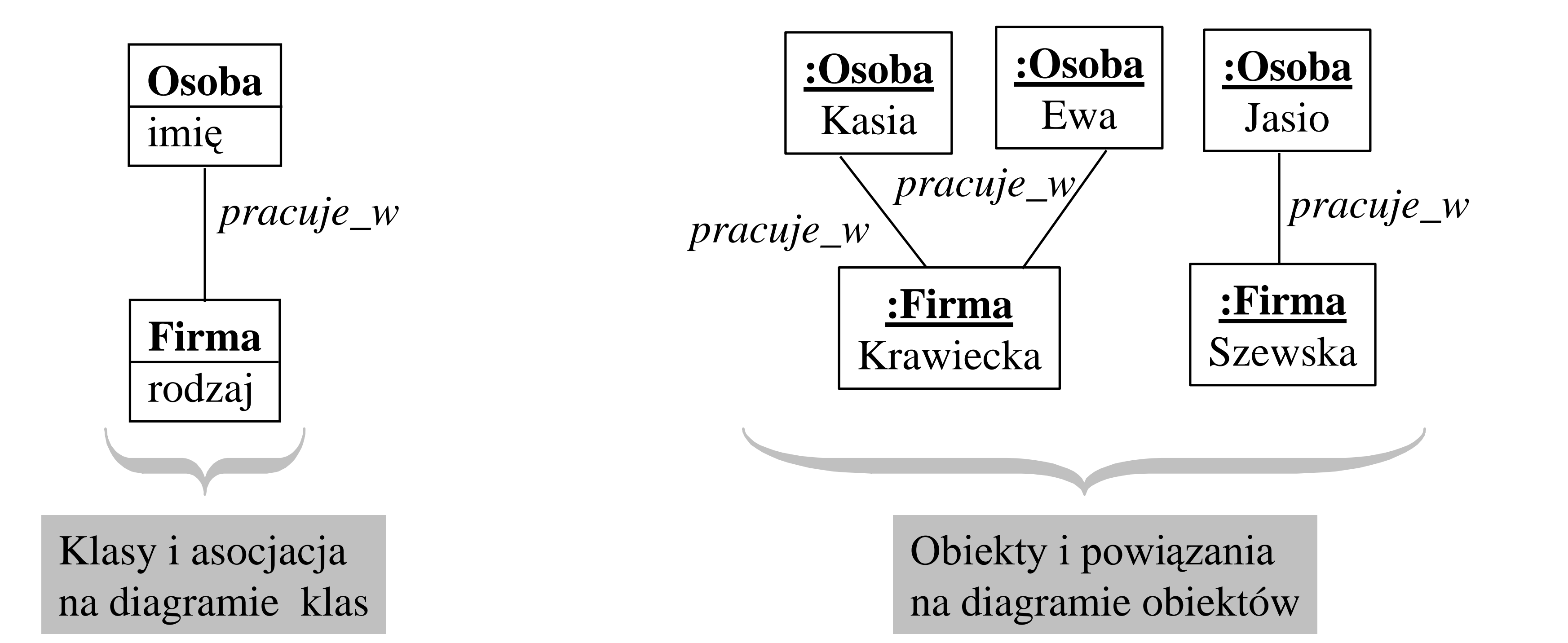
Powiązanie a asocjacja (binarne)

Powiązanie binarne

Fizyczny lub **pojęciowy związek** między dwoma obiektami, odwzorowywujący związek istniejący między odpowiednimi bytami w analizowanej dziedzinie przedmiotowej.

Asocjacja binarna

Grupa powiązań posiadających wspólną strukturę i semantykę. Powiązanie jest **wystąpieniem** asocjacji.



Liczność asocjacji

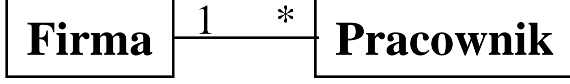
Liczność jest oznaczana na obu końcach asocjacji.

znaczenie

1	1
1..*	1, 2, 3, ...
2..*	2, 3, 4, ...
3-5	3, 4, 5
2,4,18	2, 4, 18
——	1, ?
0..1	0, 1
0..*	0, 1, 2, ...
*	0, 1, 2, ...

Przykłady:

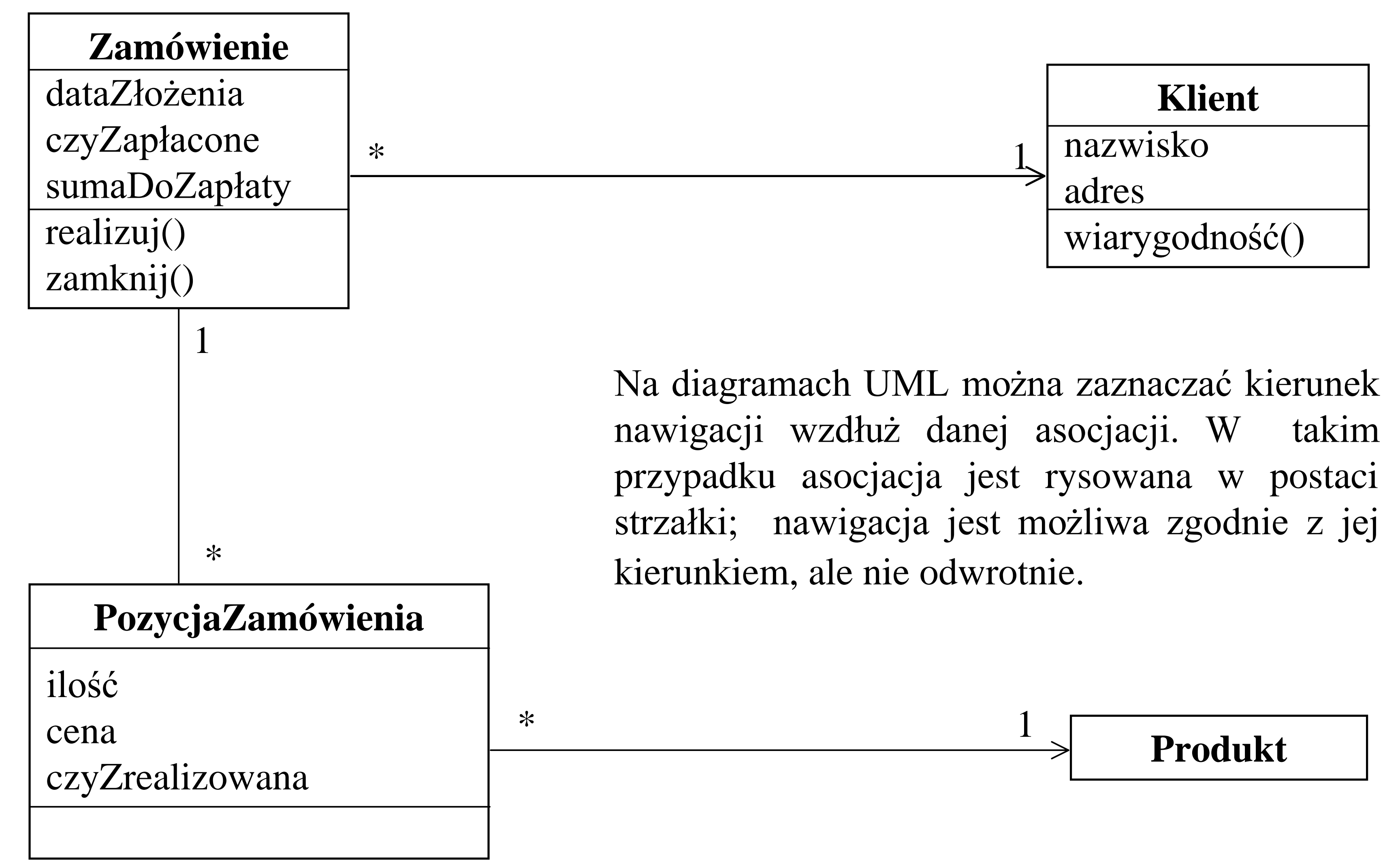
Państwo — Stolica



Osoba 0..* — 0..1 Adres

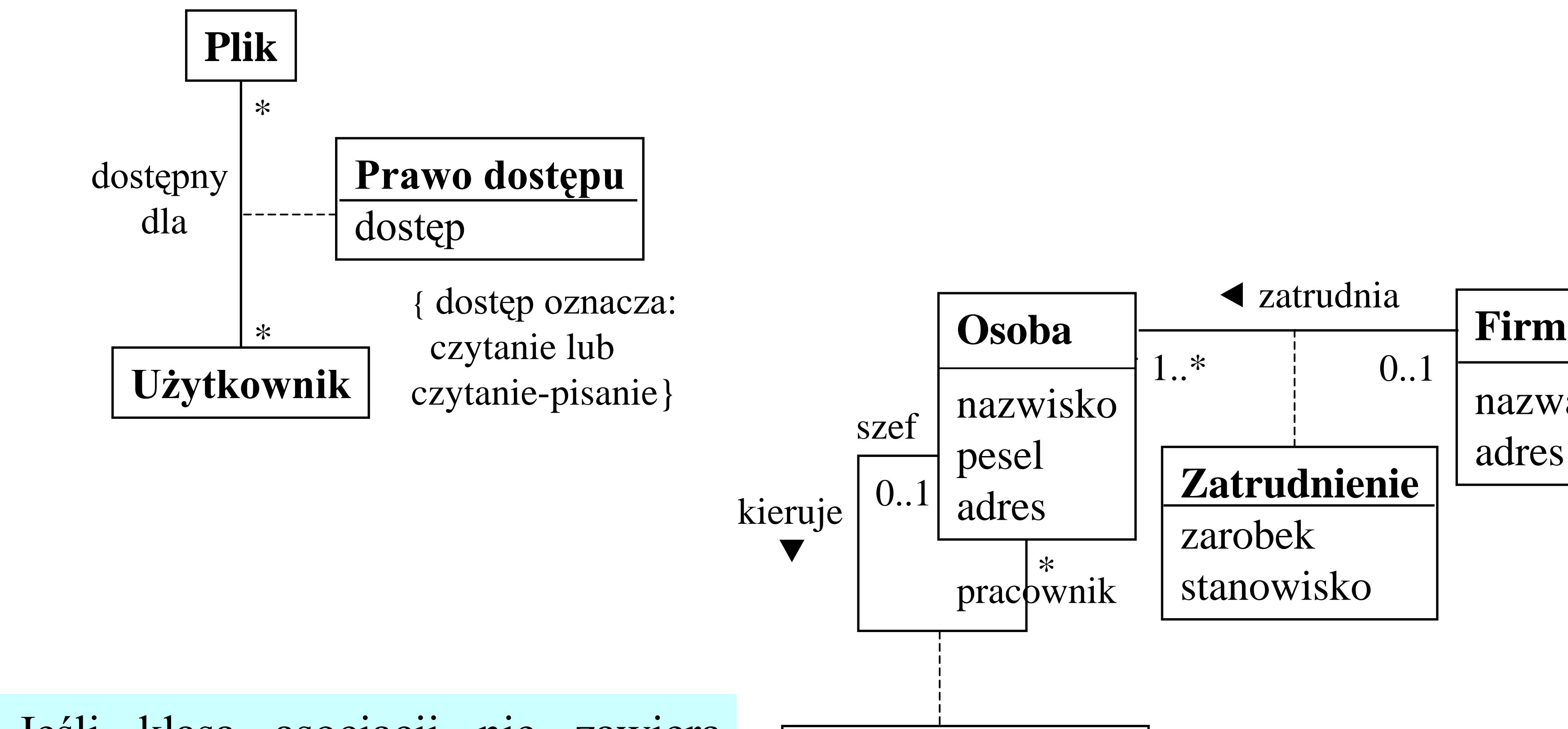
Oznaczać czy nie oznaczać liczność 1?

Asocjacje skierowane



Na diagramach UML można zaznaczać kierunek nawigacji wzdłuż danej asocjacji. W takim przypadku asocjacja jest rysowana w postaci strzałki; nawigacja jest możliwa zgodnie z jej kierunkiem, ale nie odwrotnie.

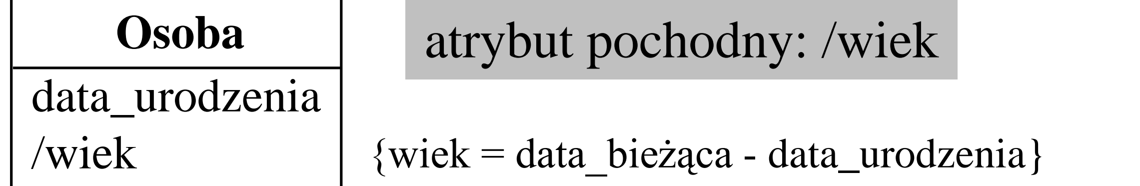
Atrybuty asocjacji

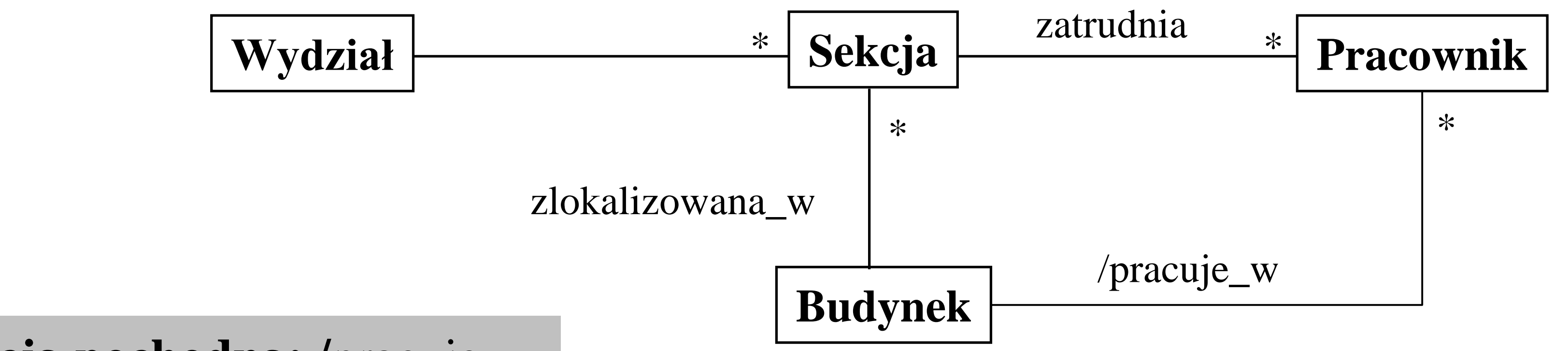


Jeśli klasa asocjacji nie zawiera metod ani asocjacji do innych klas, to jej nazwa może być opuszczona dla podkreślenia faktu, że chodzi tu wyłącznie o atrybuty asocjacji.

Atrybuty i asocjacje pochodne

Cecha pochodna jest zdefiniowana poprzez funkcję działającą na jednym lub więcej bytach modelu, które też mogą być pochodne. Cecha pochodna oznaczana jest ukośnikiem /.

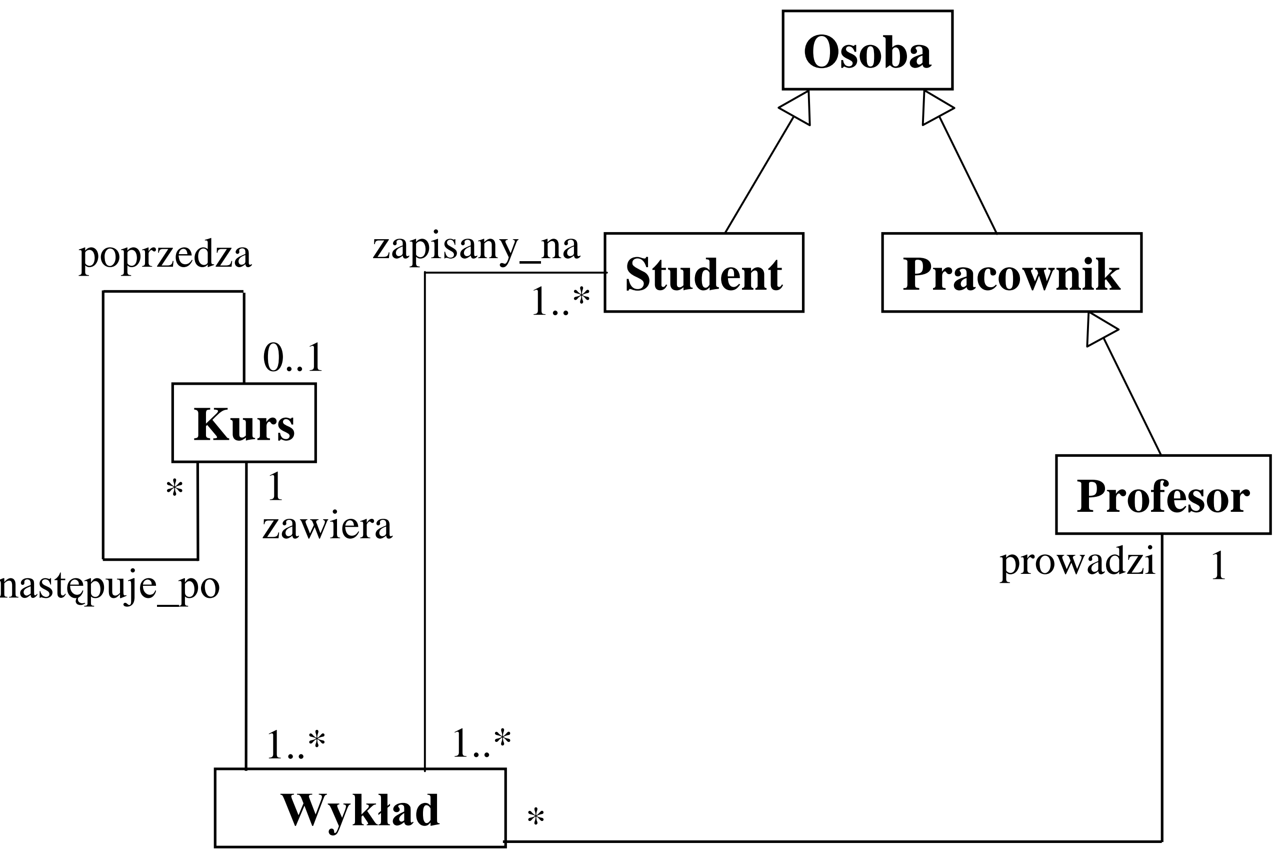




asocjacja pochodna: /pracuje_w

Asocjacja ***pracuje_w*** jest **asocjacją pochodną**, którą można wyznaczyć poprzez asocjacje ***zatrudnia*** i ***zlokalizowana_w***. Asocjację pochodną można oznaczyć poprzedzając ukośnikiem nazwę lub rolę asocjacji.

Przykładowy diagram klas



Agregacja

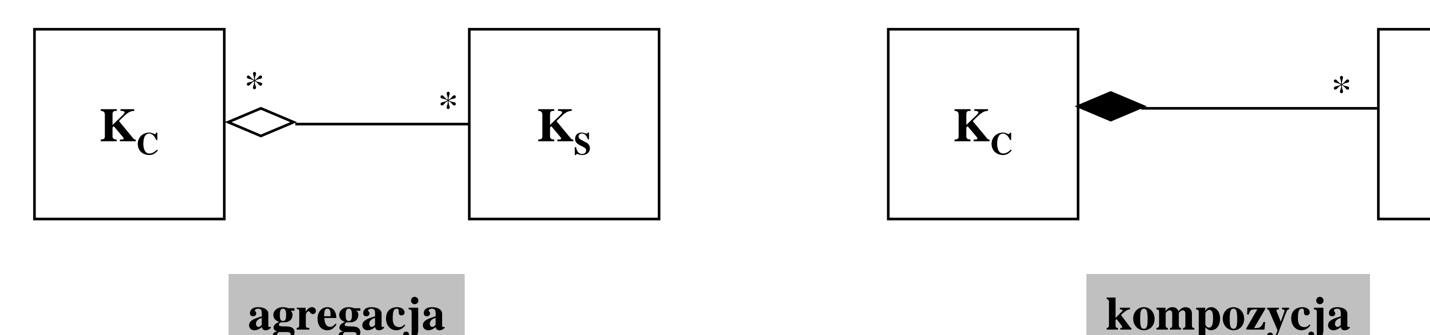
- **Agregacja jest szczególnym przypadkiem asocjacji wyrażającym związek część-całość.** Np. silnik jest częścią samochodu.
- Nie istnieje jednak powszechnie akceptowana definicja agregacji. P. Coad podaje przykład agregacji jako związek pomiędzy organizacją i jej pracownikami; dla odmiany J. Rumbaugh twierdzi, że firma nie jest agregacją jej pracowników.
- W wielu przypadkach związki agregacji są oczywiste. Jednakże wątpliwości powstają nawet w przypadku samochodu i silnika, bo np. silnik może być towarem w sklepie nie związanym z żadnym samochodem. Mętlik dookoła pojęcia agregacji wynika również z tego, że jest ona nadużywana w celu usprawiedliwienia pewnych ograniczeń modelu obiektowego.
- Np. popularne wyjaśnienie powodów braku, np. dziedziczenia wielokrotnego to, że można je „obejść przez agregację”, co jest nonsensem z punktu widzenia celów modelowania pojęciowego, tak samo jak zdanie: „asocjacje są zbędne, bo można je obejść przez atrybuty”. **Wszysto można obejść .... w assemblerze!**

Kompozycja jako mocna postać agregacji

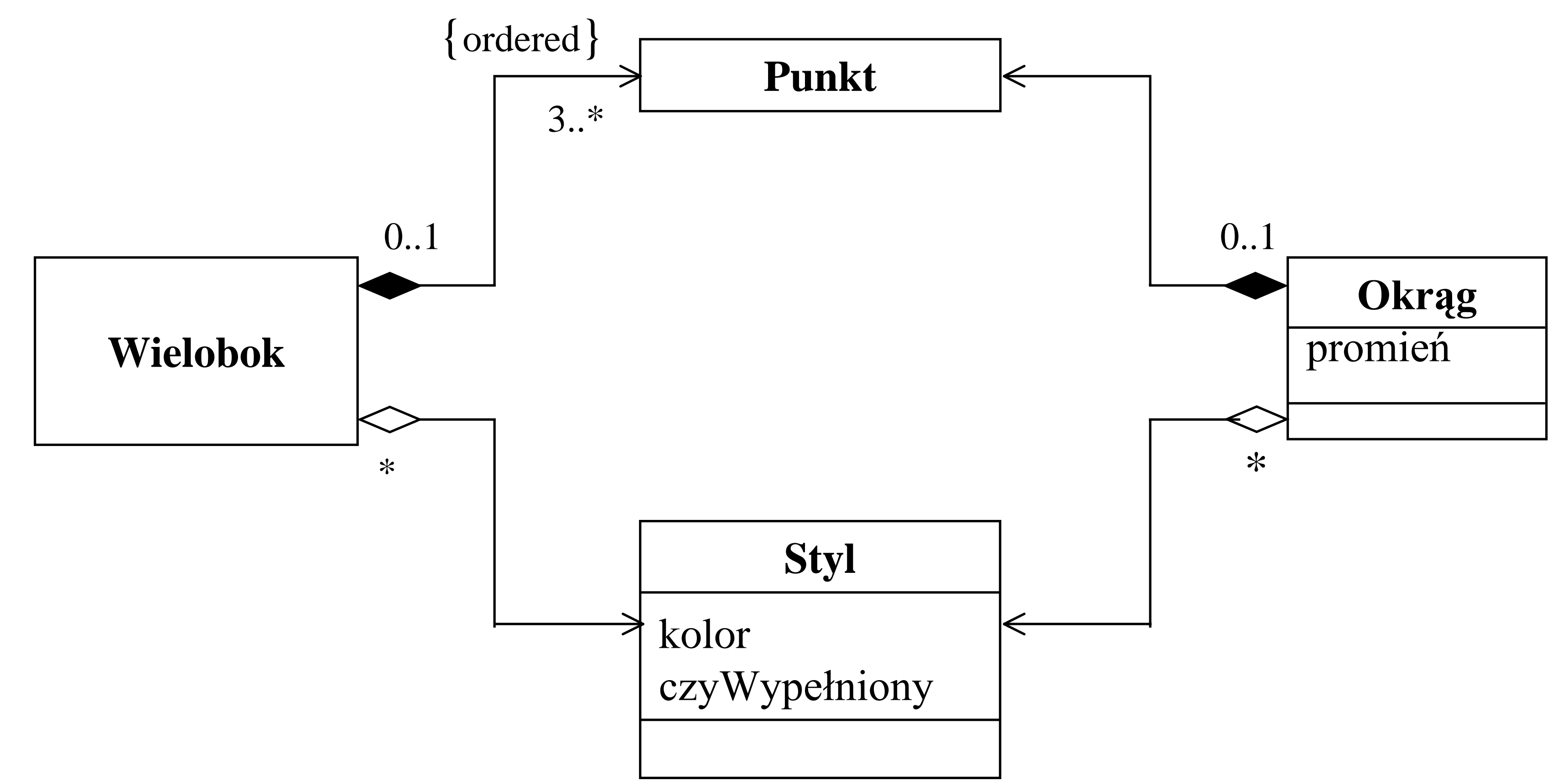
Pojęcie agregacji jest rozumiane na dwa sposoby:

- Jako związek część-całość pomiędzy obiektami świata rzeczywistego; np. silnik jest częścią samochodu.
- Jako pomocniczy środek do modelowania dowolnej innej sytuacji, kiedy trzeba wydzielić podobiekty w pewnych obiektach. np. informacja o ubezpieczeniach wewnątrz obiektów pracowników.

W UML te dwie sytuacje zostały rozdzielone. Pierwszą formę nazwano **kompozycją**. Kompozycja oznacza, że **cykl życiowy składowej zawiera się w cyklu życiowym całości, oraz że składowa nie może być współdzielona**.

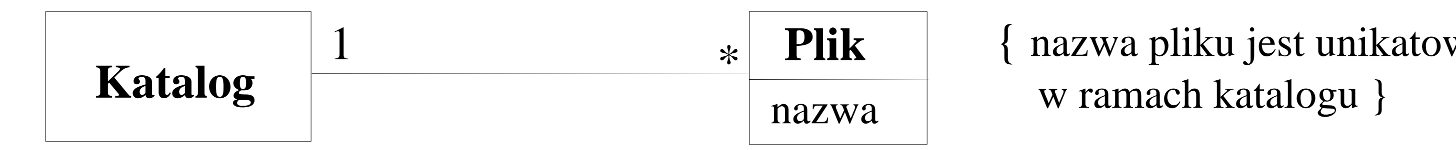


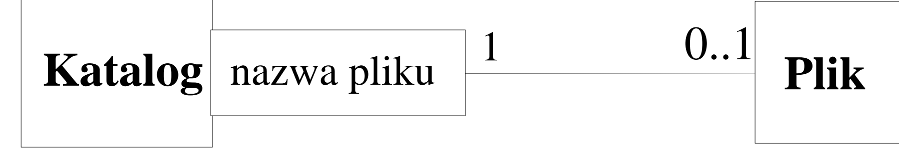
Agregacja a kompozycja; przykład



W przedstawionym rozwiązaniu, punkt na płaszczyźnie, w którym przecinają się okrąg i wielobok, jest odwzorowywany w dwa (?) obiekty klasy Punkt.

Asocjacja kwalifikowana





1 **Perspektywa pojęciowa** - plik jes t w ramach katalogu jednoznacznie identyfikowany przez nazwę.

2 **Perspektywa projektowa** - wskazanie na to, że katalog plików można zorganizować jako tablicę asocjacyjną lub słownik (przeeszukiwanie za pomocą nazwy pliku).

Asocjacja n-arna

Asocjacja n-arna to asocjacja, której wystąpienia łączą **n** obiektów, będących instancjami co najwyżej n klas. Dana klasa może pojawić się na więcej niż jednej pozycji w asocjacji.

Asocjacja binarna ze swoją uproszczoną notacją i pewnymi dodatkowymi własnościami (takimi jak możliwość ustalania kierunku nawigowania, kwalifikatorów, związków agregacji czy kompozycji) jest specjalnym rodzajem asocjacji n-arnej (dla n=2).

Diagramy interakcji

- Diagramy interakcji (współpracy i sekwencji) jako główne zadanie mają wspomożenie projektanta w procesie konstruowania modelu obiektowego (diagramu klas). Pomoc polega na analizie zachowania systemu w trakcie realizacji jego zadań i identyfikowaniu nowych czy też korekcie już istniejących elementów modelu, np.: klas, ich atrybutów czy metod oraz asocjacji między klasami.

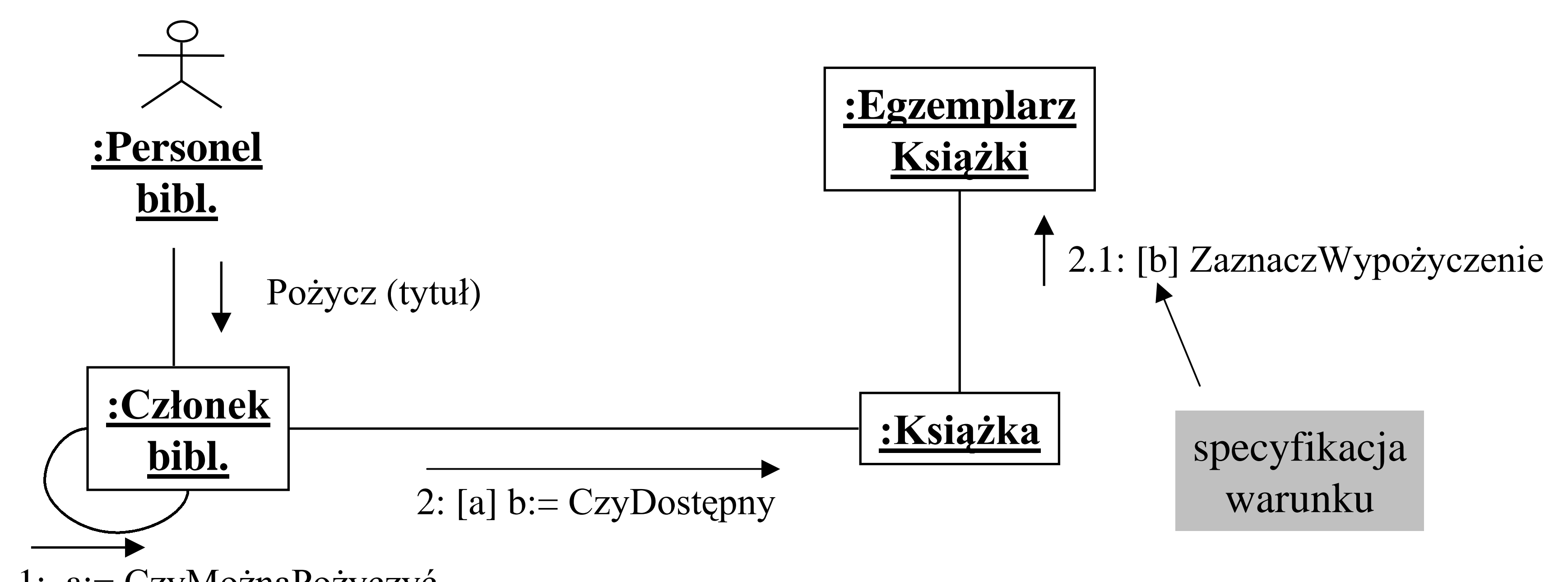
- Oba rodzaje diagramów przedstawiają bardzo podobną informację, w nieco inny sposób.

- Diagramy współpracy, stanowiące w pewnym sensie wystąpienia fragmentu diagramu klas, lepiej przedstawiają związki między obiektami biorącymi udział w realizacji danego przypadku użycia. Łatwiej też można tu odwzorować efekty oddziaływania na pojedynczy obiekt.

- Diagramy sekwencji lepiej przedstawiają zależności czasowe, bardziej niż diagramy kolaboracji nadają się do modelowania systemów czasu rzeczywistego i złożonych scenariuszy.

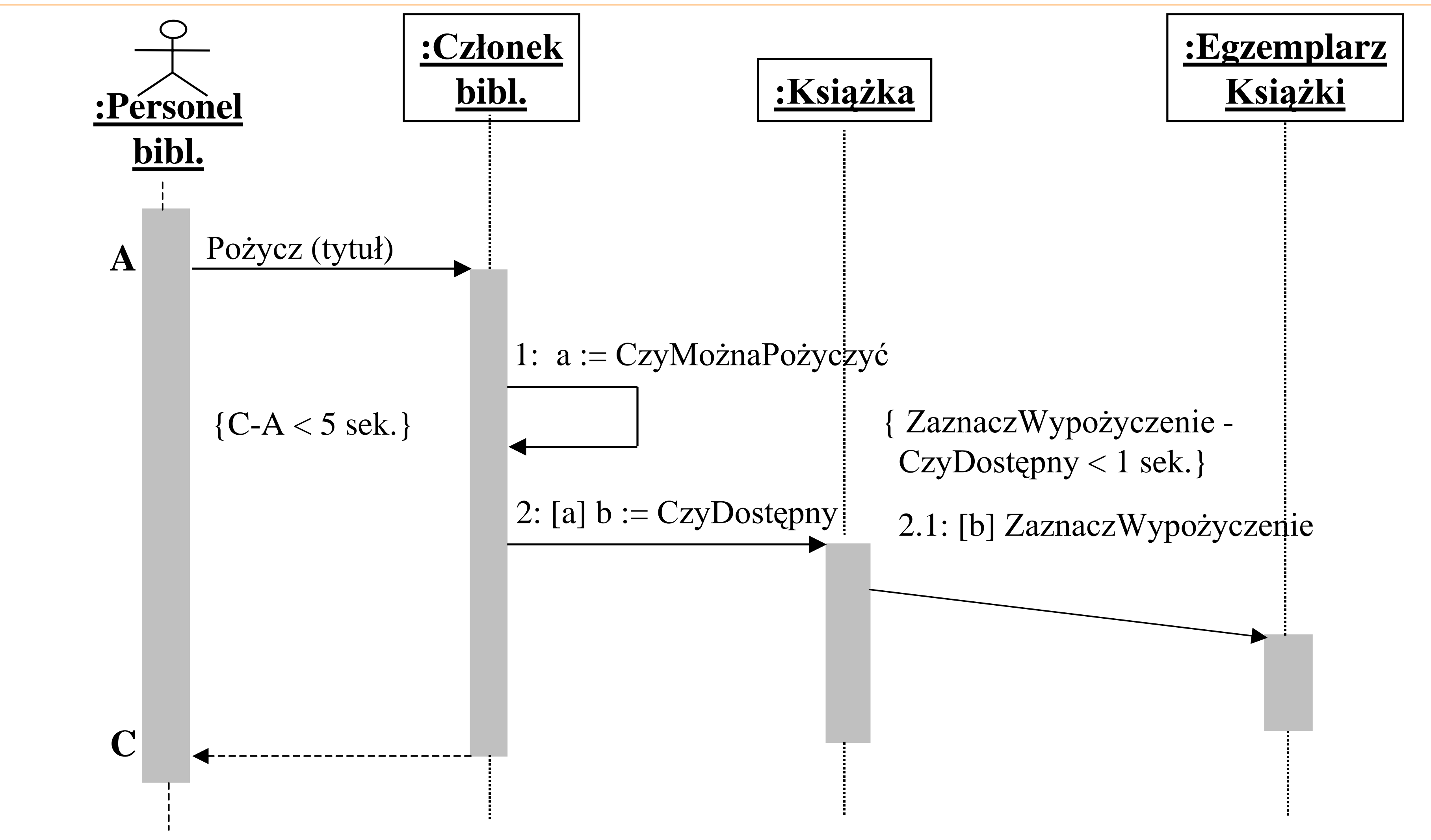
- Diagramy sekwencji dostarczają środków do modelowania przetwarzania współbieżnego.

Interakcja na diagramach współpracy



Komunikaty mogą być numerowane, albo kolejnymi liczbami naturalnymi , albo stosując numerację zagnieżdżoną. Numeracja zagnieżdżona oznacza: jeśli obiekt ***O*** otrzyma komunikat o numerze np. 7.3 to ten numer będzie dołączany jako prefix do każdego komunikatu wysyłanego w trakcie realizacji komunikatu 7.3 przez ***O***.

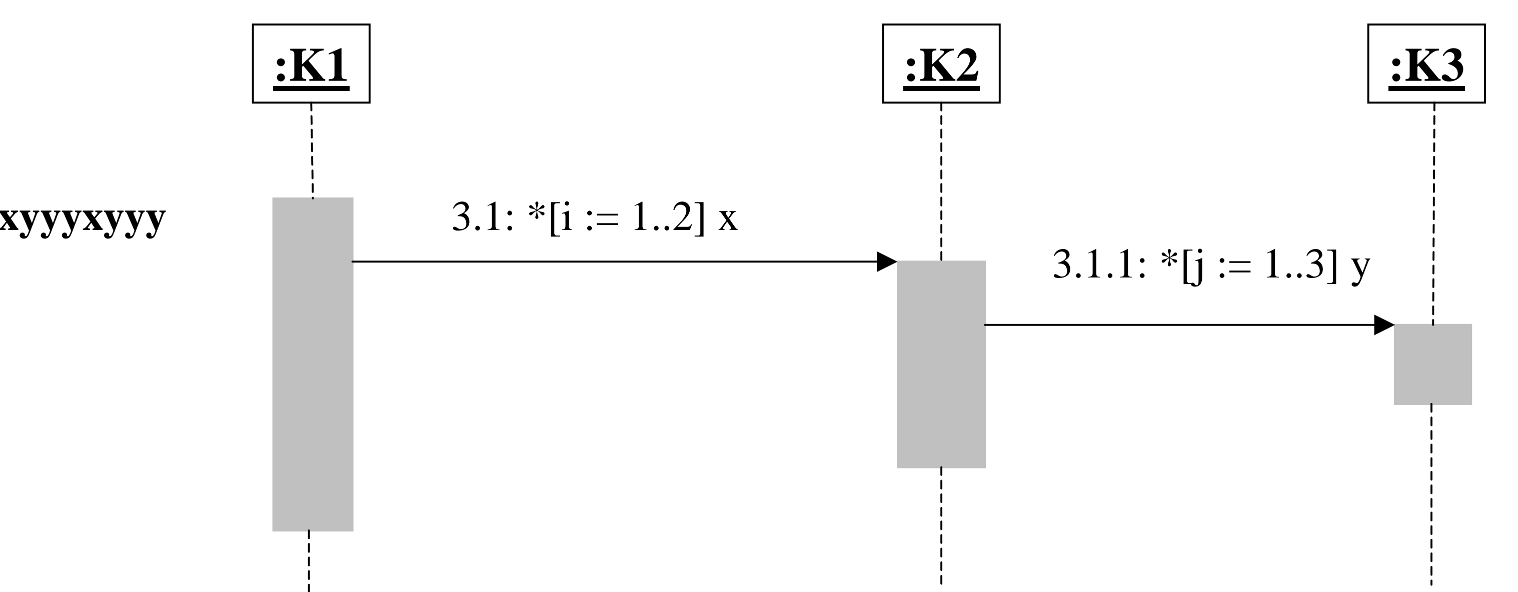
Interakcja na diagramach sekwencji



Dwa sposoby opisywania czasu: oznaczanie skali czasowej lub nakładanie ograniczeń na upływ czasu.

Generyczne diagramy interakcji

Generyczny diagram interakcji ma z założenia specyfikować **wszystkie sekwencje interakcji** dla danego przypadku użycia, a nie tylko dla jednego z możliwych scenariuszy. UML dostarcza środki zarówno do modelowania zachowań warunkowych, jak i iteracji.



Komunikat, który ma być wysłany wiele razy, musi być poprzedzony symbolem *****. Oczywiście musi być też wyspecyfikowany warunek, określający zakończenie iteracji.

Oznaczanie współbieżności

Rodzaj interakcji	Symbol	Znaczenie
synchroniczna	——▶	“Normalna” proceduralna sytuacja. Nadawca zawiesza działanie, dopóki odbiorca nie przekaże sterowania, co można oznaczyć wykorzystując symbol powrotu.
powrót	------>	Powrót nie jest komunikatem. Oznacza zakończenie komunikatu i przekazanie sterowania do nadawcy.
płaska (flat)	——>	Nadawca komunikatu przekazuje sterowanie do odbiorcy, kończąc jednocześnie własną działalność.
asynchroniczna	——⇁	Nadawca komunikatu odbiorcy nie oczekuje na odpowiedź odbiorcy, ale też i nie kończy własnej aktywności, co oznacza, że nadal przetwarza i może wysyłać komunikaty.

Diagramy stanu (1)

Maszyna stanu jest grafem skierowanym,, którego wierzchołki stanowią stany obiektu, a łuki opisują przejścia między stanami. Przejście między stanami jest odpowiedzią na zdarzenie. Zwykle, maszyna stanu jest przypisana do klasy stanowiąc model historii życia dla obiektu tej klasy. Można przypisać maszynę stanu do przypadku (ów) użycia, operacji, kolaboracji, ale dla opisu przepływu sterowania częściej wykorzystuje się inne środki, np. diagramy aktywności.

Stan obiektu może być charakteryzowany na kilka sposobów: jako okres czasu, w którym obiekt oczekuje na zdarzenie albo jako okres czasu, w którym obiekt przetwarza albo jako zbiór wartości obiektu (atrybutów i powiązań) - w pewnym aspekcie podobnych.

Nazwa stanu
entry/akcja1/akcja2/… do/aktywność1/aktywność2/… exit/akcja1/akcja2/...

akcja - operacja atomowa
lista akcji - akcja1/akcja2/… - traktowana jest, jak pojedyncza operacja,
aktywność - operacja, którą można przerwać,
lista aktywności - podobnie, jak lista akcji,

entry, **do**, **exit** - słowa kluczowe do specyfikowania operacji wykonywanych na wejściu, w trakcie i na wyjściu ze stanu

Diagramy stanu (2)

Typ zdarzenia	Opis	Składnia
wołanie	otrzymanie przez obiekt synchronicznego żądania wykonania operacji - najbardziej podstawowy rodzaj zdarzenia	**op** (a : T)
zmiana	zdarzenie typu **zmiana** jest użyteczne np. do modelowania sytuacji, gdy obiekt zmienia stan po otrzymaniu odpowiedzi na wysłany przez siebie komunikat	**when**(wyrażenie)
sygnał	otrzymania przez obiekt asynchronicznego żądania wykonania operacji; użyteczne do modelowania zdarzeń przychodzących z zewnątrz systemu	**nazwa_syg** (a : T)
czas	upłynięcie czasu określonego w sposób bezwzględny lub względny, np. after (5 sec.)	**after** (czas)

Diagramy stanu (3)

1. **przejście zewnętrzne** (external transition)

2. **przejście wewnętrzne** (internal transition)

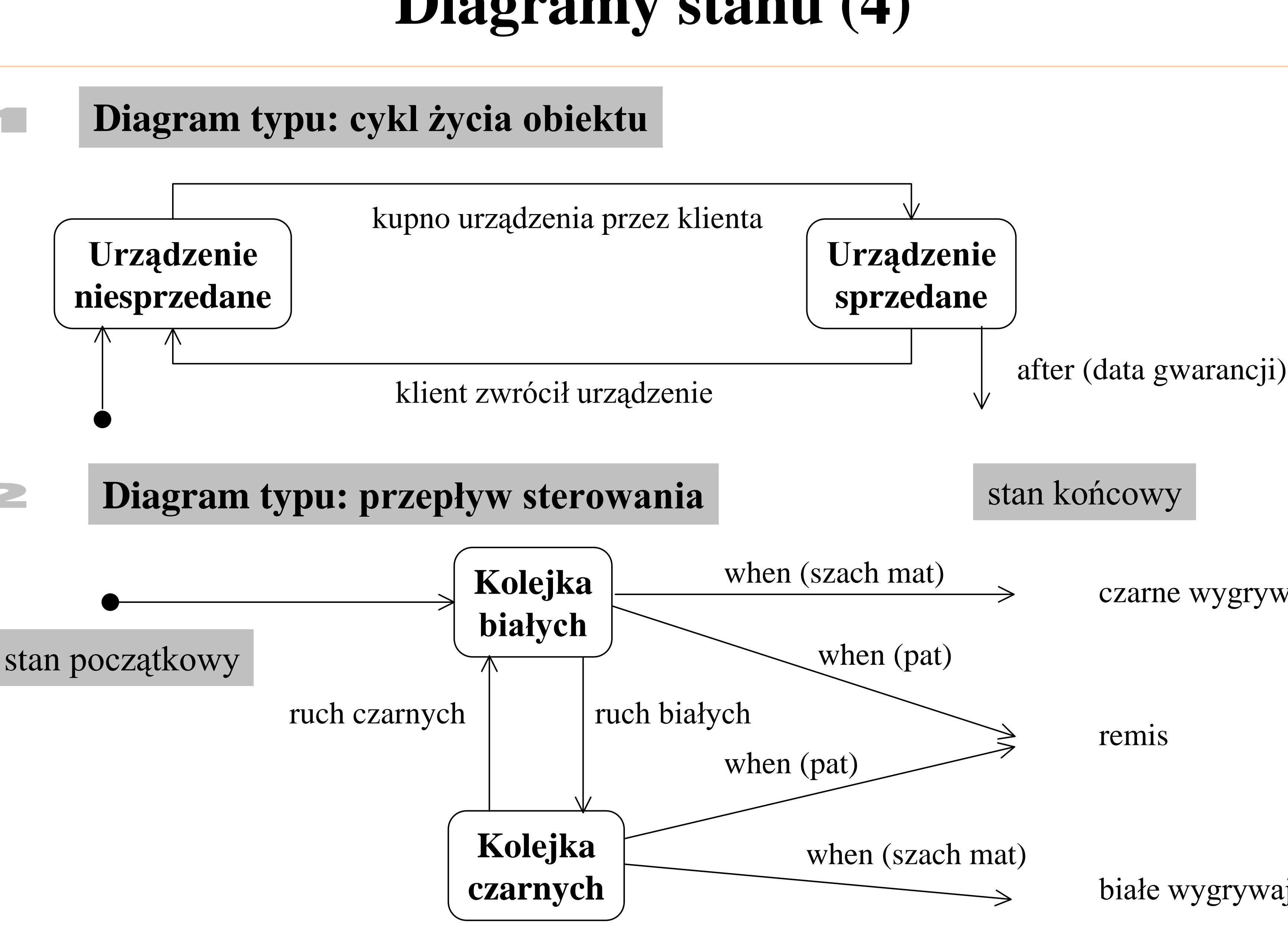
bez zmiany stanu

3. **samo-przejście** (selftransition)

4. **przejście automatyczne** (completion transition)

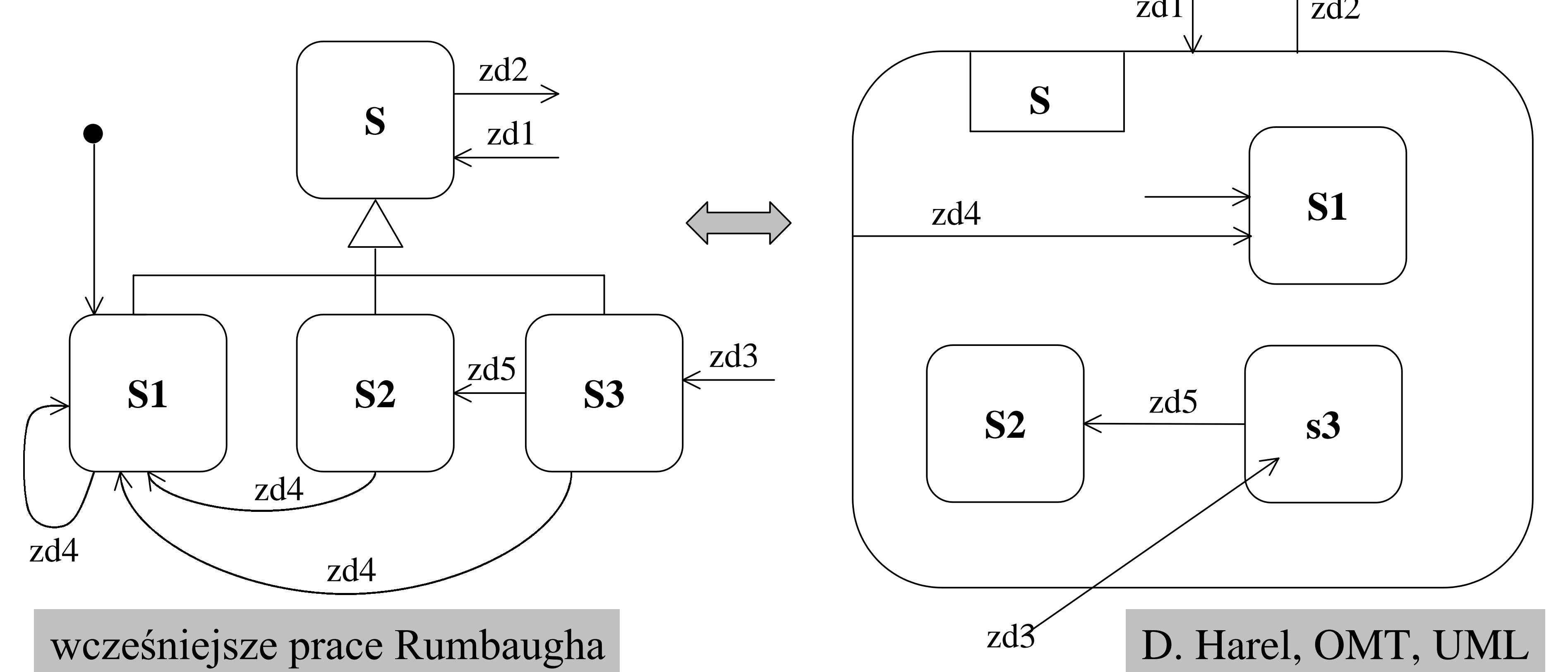
wszystkie operacje wyspecyfikowane po słowach kluczowych entry, exit i do zostały ukończone

Diagramy stanu (4)



Diagramy stanu (5)

Stan prosty nie posiada substruktury, jest specyfikowany przez zbiór operacji oraz przejść. **Stan złożony**, powstały w efekcie zagnieżdżania stanów, może być zdekomponowany na stany bardziej proste, a dekompozycja jest tu rodzajem specjalizacji. Każdy z podstanów dziedziczy przejścia nadstanu. Tylko jeden z podstanów może być aktywny w danym momencie.



Diagramy aktywności (1)

Graf aktywności to maszyna stanu, której podstawowym zadaniem nie jest analiza stanów obiektu, ale modelowanie przetwarzania (przepływów operacji). Stany grafu aktywności, zwane **aktywnościami**, odpowiadają stanom wyróżnialnym w trakcie przetwarzania, a nie stanom obiektu. Aktywność może być interpretowana różnie, w zależności od perspektywy: jako zadanie do wykonania i to zarówno przez człowieka, jak i przez komputer (z perspektywy pojęciowej) czy też np. jako pojedyncza metoda (z perspektywy projektowej). Podobnie, przejścia między stanami raczej nie są tu związane z nadejściem zdarzenia, ale z zakończeniem przetwarzania wyspecyfikowanego dla danego stanu.

Dla skompletowania projektu każda aktywność powinna być rozpisana na szereg operacji, z których każdą trzeba będzie na późniejszym etapie przydzielić do odpowiedniej klasy. Diagramy aktywności są szczególnie użyteczne przy modelowaniu przepływów operacji czy też w opisie zachowań z przewagą przetwarzania współbieżnego.

Diagramy aktywności (2)

Oznaczenia:

nazwa aktywności — **aktywność**

przejście, z zasady nie opisywane; może być opatrzone warunkiem, może też być oznaczone symbolem iteracji; akcje opisujące przejścia powinny być raczej dołączone do którejś z aktywności;

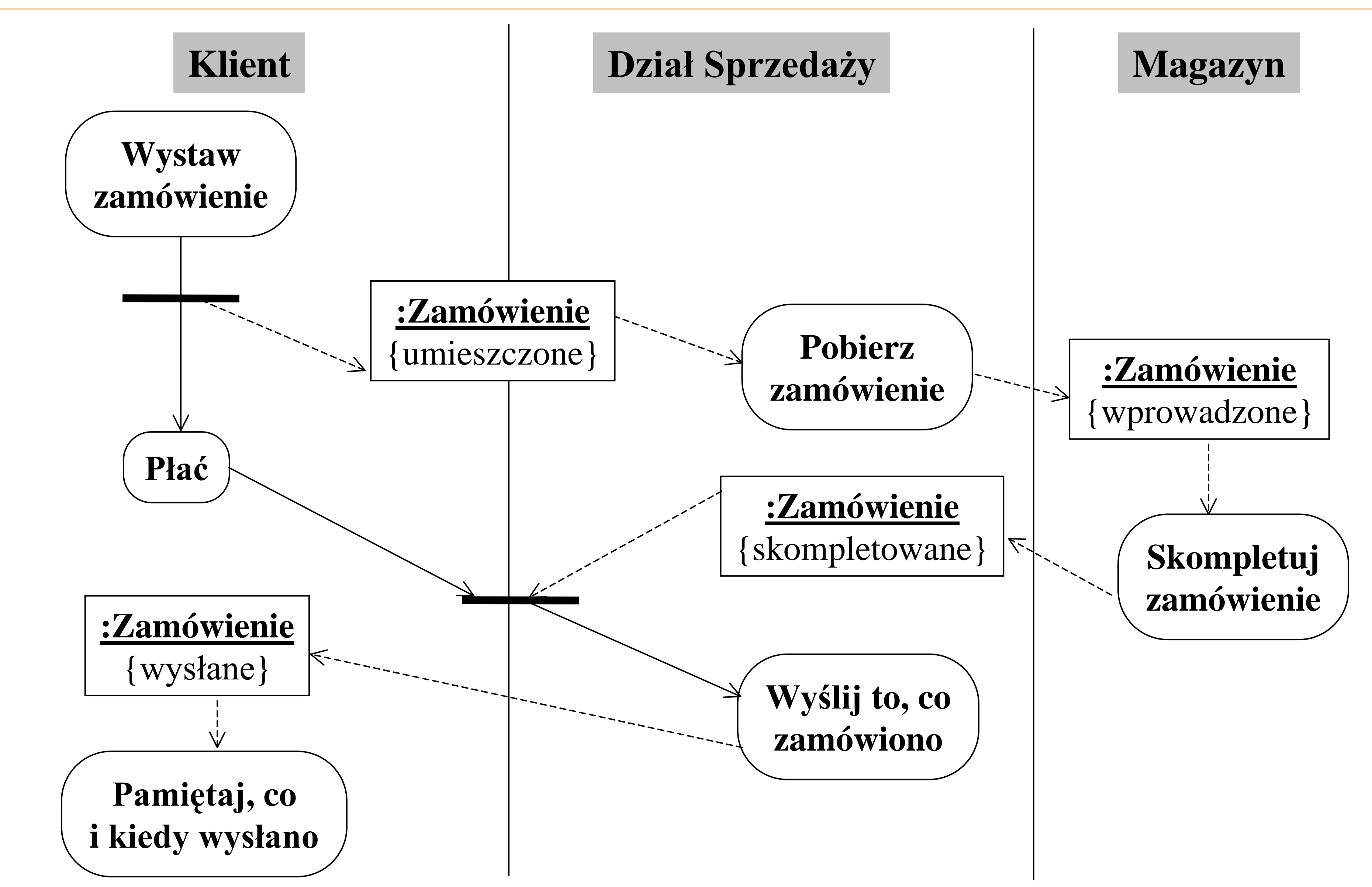
romb, który może rozdzielać jedno przejście na kilka innych (opatrzonych warunkami) lub łączyć kilka alternatywnych przejść w jedno

sztabka synchronizujaca (synchronization bar); może być typu "**fork**" (rozdzielenie jednej operacji na kilka przebiegających równolegle) lub typu "**join**" (synchronizacja kilku operacji równoległych w jedną)

aktywność początkowa

aktywność końcowa

Diagramy aktywności (3)



Diagramy implementacyjne

Diagramy implementacyjne pokazują niektóre aspekty implementacji SI, włączając w to strukturę kodu źródłowego oraz strukturę kodu czasu wykonania (*run-time*). Konstruowanie takich diagramów jest użyteczne zarówno ze względu na ponowne użycie, jak i ze względu na możliwość osiągania z ich pomocą odpowiednich parametrów wydajnościowych.

UML wprowadza dwa rodzaje takich diagramów:

- **Diagramy komponentów** pokazujące zarówno implementację elementów projektu (np. klas) przez komponenty, jak i interfejsy oraz zależności między komponentami, innymi słowy pokazujące strukturę kodu konstruowanego SI.

- **Diagramy wdrożeniowe** pokazujące konfigurację systemu czasu wykonania, czyli rozmieszczenie komponentów i obiektów na obliczeniowych zasobach czasu wykonania, zwanych tu węzłami. Taka konfiguracja może być zarówno statyczna, jak i dynamiczna - i komponenty i obiekty mogą migrować między węzłami w czasie wykonania.

Diagramy komponentów

Komponent jest jednostką implementacji z dobrze zdefiniowanym interfejsem, dobrze wyizolowany z kontekstu, nadający się do wielokrotnego wykorzystania.

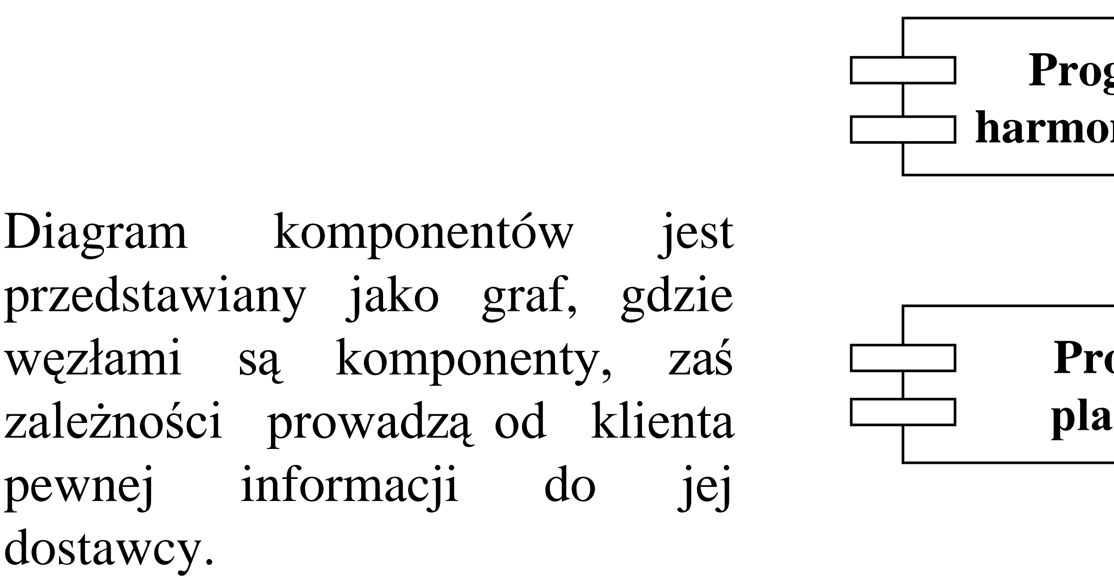
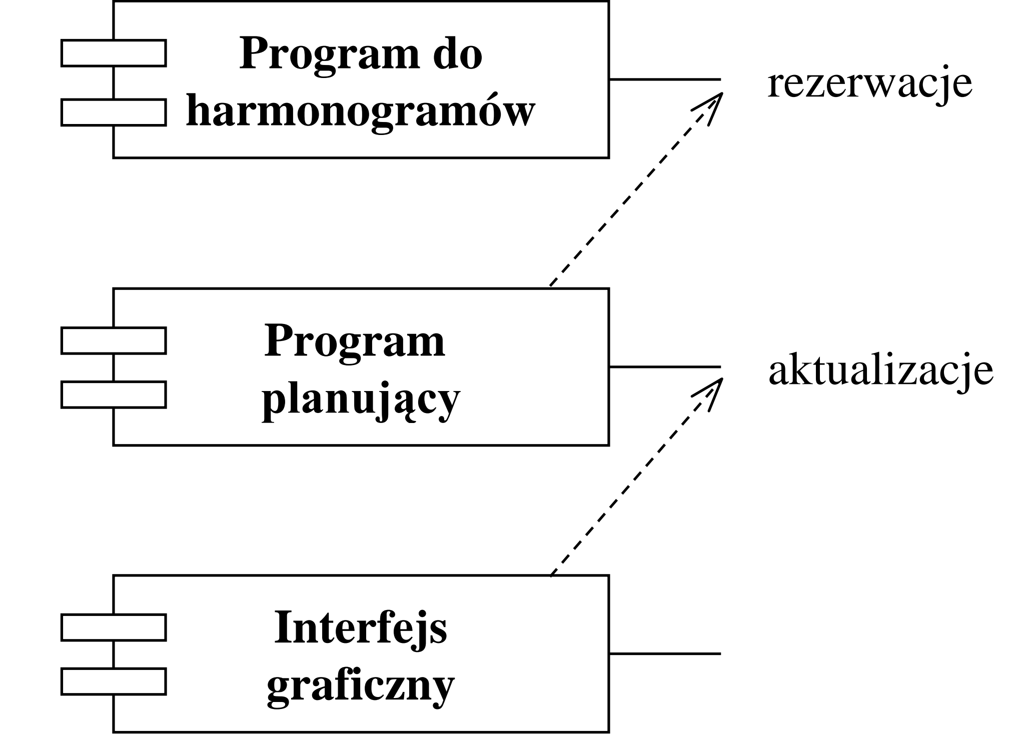


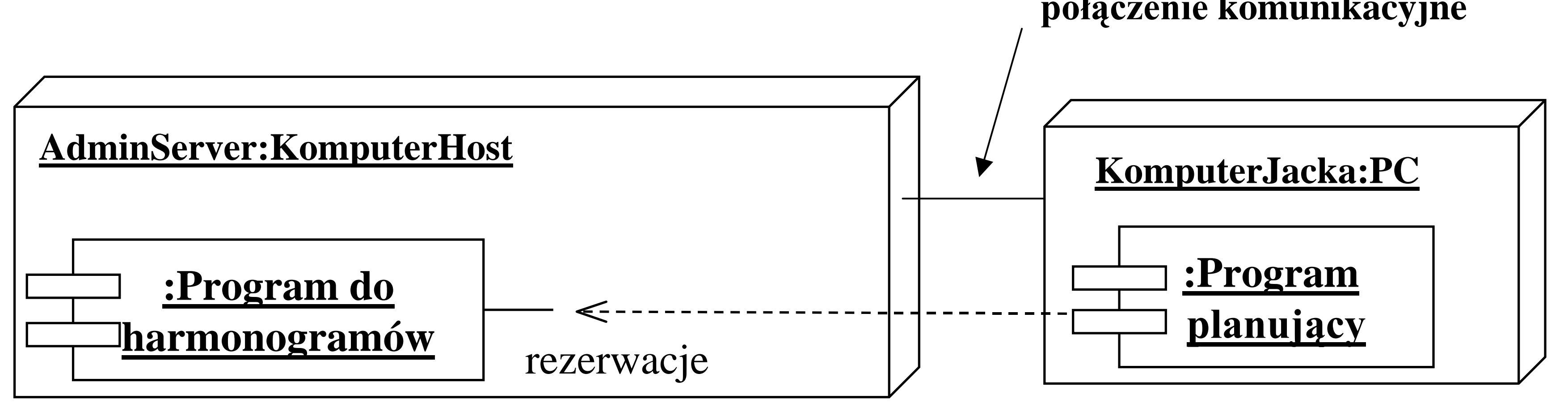
Diagram komponentów jest przedstawiany jako graf, gdzie węzłami są komponenty, zaś zależności prowadzą od klienta pewnej informacji do jej dostawcy.



Diagramy wdrożeniowe

Diagramy wdrożeniowe pokazują konfigurację elementów czasu wykonania: komponentów sprzętowych (fizycznych jednostek posiadających co najmniej pamięć, a często i możliwości obliczeniowe), komponentów oprogramowania oraz związanych z nimi obiektów.

Diagram wdrożeniowy jest grafem, gdzie wierzchołki, zwane węzłami, połączone są przez linie odwzorowywujące połączenia komunikacyjne komponentów sprzętowych. Węzły, podobnie jak połączenia komunikacyjne, mogą być opatrzone stereotypami, np.: «CPU», «pamięć», «urządzenie jakieś». Węzły przechowują wystąpienia obiektów i komponentów.



Diagramy pakietów (1)

- Pakiety stanowią zgrupowanie elementów modelu. Są środkiem ogólnego zastosowania przeznaczonym do budowania struktur hierarchicznych.

- Każdy element modelu musi być przypisany do jednego pakietu (home package). Model może być opisany przez zbiór pakietów.

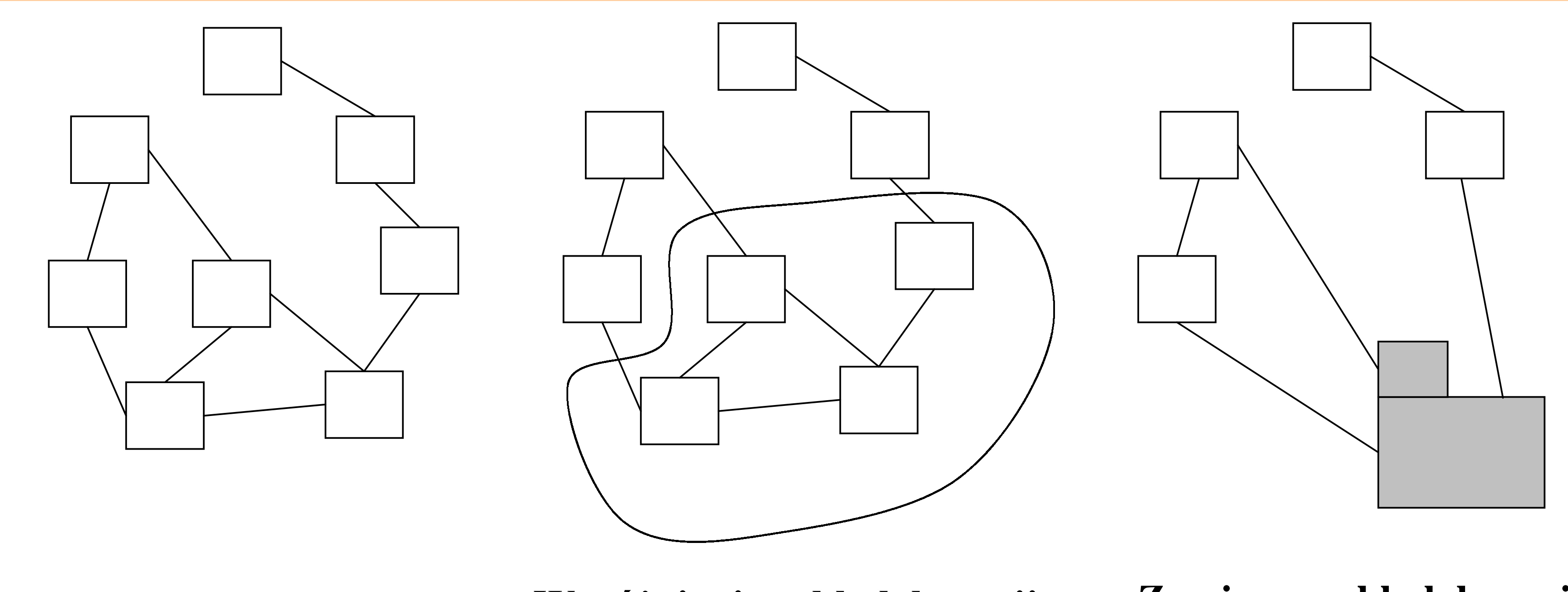
- Pakiet oprócz elementów modelu może też zawierać inne pakiety (zagnieżdżanie).

- Są pakiety specjalnego rodzaju: fasada (facade), model, podsystem, system.

- Stosowanie pakietów znacząco ułatwia zarządzanie przechowywaniem, konfiguracjami czy modyfikowaniem elementów systemu.

- Dobrze przeprowadzony podział na pakiety może znacząco ułatwić zarządzanie procesem konstrukcji produktu programistycznego.

Diagramy pakietów (2)

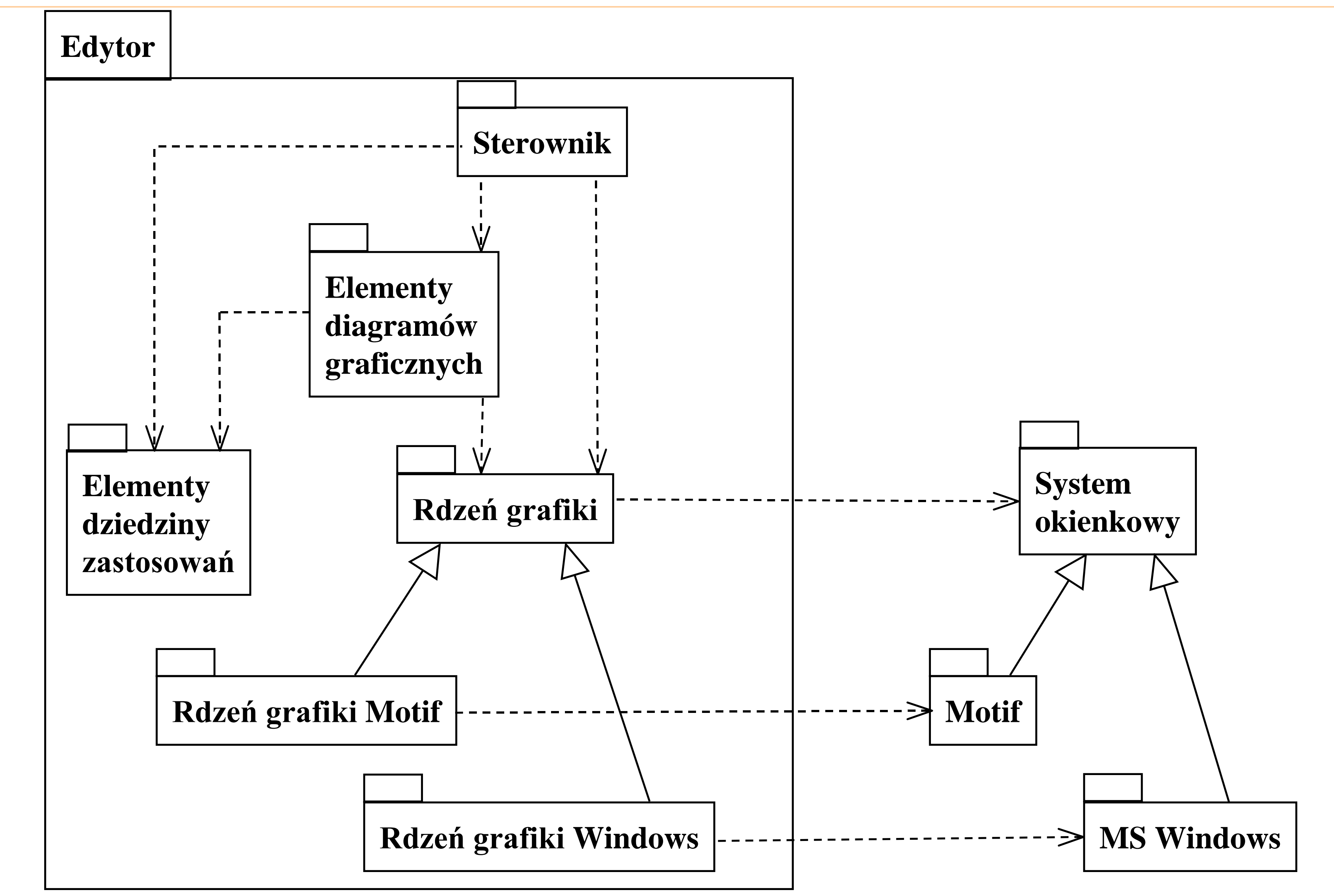


Złożona kolaboracja **Wyróżnianie subkolaboracji** **Zamiana subkolaboracji na pakiet**

Ukrywanie detali, jest użyteczne, jak każda abstrakcja. Temu celowi może np. służyć wyodrębnienie subkolaboracji, a następnie zamiana jej na pakiet. Pakiet nie posiada własnego interfejsu, w tym sensie, że przesłanie komunikatu do pakietu, oznacza przesłanie komunikatu do obiektu wewnątrz pakietu.

W UML, **sparametryzowana kolaboracja** jest traktowana jako **wzorzec projektowy (design pattern)**.

Diagramy pakietów (3)



Diagramy pakietów (4)

«**facade**» («fasada») Zawiera wyłącznie odwołania do elementów zdefiniowanych w innych pakietach.

«**model**» Stanowi mniej lub więcej kompletną abstrakcję systemu (na pewnym poziomie szczegółowości), widzianą z pewnej perspektywy. Zwykle zawiera drzewo pakietów.

«**subsytem**» (« podsystem») Jest rodzajem pakietu, który reprezentuje pewien spójny, logiczny, dobrze wyizolowany fragment systemu. Posiada dobrze wyspecyfikowany zbiór interfejsów, do interakcji z otoczeniem. Podsystem podzielony jest na dwie części: specyfikacyjną i realizacyjną. Część specyfikacyjna zawiera opis interakcji z otoczeniem, z reguły za pomocą przypadków użycia. Część realizacyjna, posługując się kolaboracjami, podaje sposoby realizacji przypadków przez podsystem. Podsytemy mogą być zbudowane z innych podsystemów, wtedy te najniższego poziomu muszą już zawierać elementy modelu.

Podsytem stanowi zgrupowanie elementów modelu logicznego. Komponent jest zgrupowaniem elementów modelu implementacyjnego. Zwykle, podsystemy są implementowane jako komponenty. Takie podejście upraszcza mapowanie modelu logicznego na implementacyjny.

Mechanizmy rozszerzalności w UML

UML posiada trzy rodzaje mechanizmów rozszerzalności:

- stereotypy,
- wartości etykietowane,
- ograniczenia.

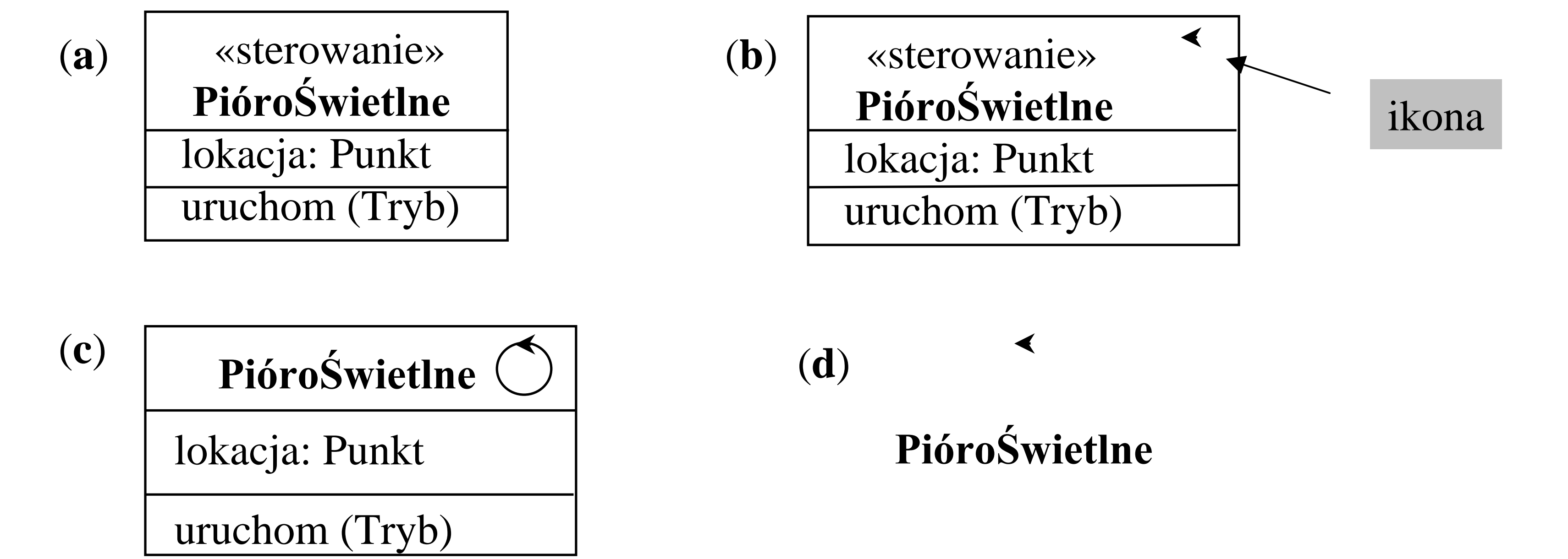
Stereotypy

- Stereotypy umożliwiają metaklasyfikację elementów modelu.
- Istnieje **lista** stereotypów dla każdego **rodzaju** elementów modelu (elementu metamodelu UML), np. relacji między przypadkami użycia, klas czy metod.
- Dany element modelu (np. konkretna klasa czy metoda) może być oznaczona co **najwyżej jednym** stereotypem.
- Są stereotypy predefiniowane, ale użytkownicy mogą też definiować własne.
- Stereotypy rozszerzają semantykę metamodelu.

Stereotypy; notacja

Notacja: zwykle «**nazwa stereotypu**» lub **ikona**, ale można też używać **koloru** czy **tekstury**, choć z różnych względów nie jest to polecane (ograniczenia ludzkie lub sprzętu).

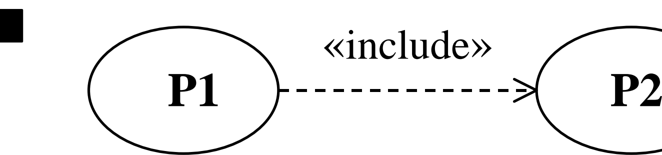
« » guillemets - jeden znak - używany w charakterze cudzysłowia w jęz. francuskim

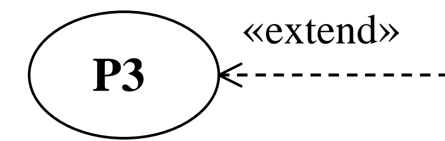


Ikona może być używana na 2 sposoby: zamiast symbolu stereotypu (**c**, **d**) lub razem z nim (**b**). W przypadkach **a**, **b**, **c** zawartość elementu modelu opatrzonego stereotypem (tu: klasy **Pióro Świetlne**) jest widoczna. W przypadku **d** została opuszczona.

Stereotypy; przykłady

1





rodzaj elementów modelu: relacje między przypadkami użycia
lista stereotypów dla tego rodzaju: «include» i «extend»

Każda relacja między przypadkami użycia (element modelu) jest opatrzona jednym z dwóch stereotypów z powyższej listy.

2

«trwała» **Prostokąt**
punkt1: Punkt punkt2: Punkt
«konstruktory» Prostokąt (p1: Punkt, p2: Punkt) «zapytania» obszar () : Real aspekt() : Real ... «aktualizacje» przesuń (delta: Punkt) przeskaluj (współczynnik: Real)

«trwała» **Prostokąt**
punkt1: Punkt punkt2: Punkt
«konstruktory» Prostokąt (p1: Punkt, p2: Punkt) «zapytania» obszar () : Real aspekt () : Real ... «» przesuń (delta: Punkt) przeskaluj (współczynnik: Real)

Jednym stereotypem można opatrzyć całą listę elementów modelu. Koniec listy może być oznaczony przez «».

Wartości etykietowane

Wartości etykietowane są używane do skojarzenia **arbitralnej informacji z pojedynczym elementem modelu.**

- Wartość etykietowaną stanowi ciąg znaków o postaci: **słowo kluczowe = wartość**.

  Dowolny łańcuch znaków może być użyty jako słowo kluczowe.

- Są słowa kluczowe predefiniowane, ale użytkownik może też definiować własne.
- Listę wartości etykietowanych (oddzielonych przecinkami) umieszcza się w **{}**.
- Dowolny element modelu może być skojarzony nie tylko z listą wartości

  etykietowanych, ale w bardziej ogólnym sensie z łańcuchem własności, w postaci:

  {dowolny łańcuch znaków}.

Przykład:

{autor = "Jan Nowak", termin zakończenia = "31 Maja 1999", status = analiza}

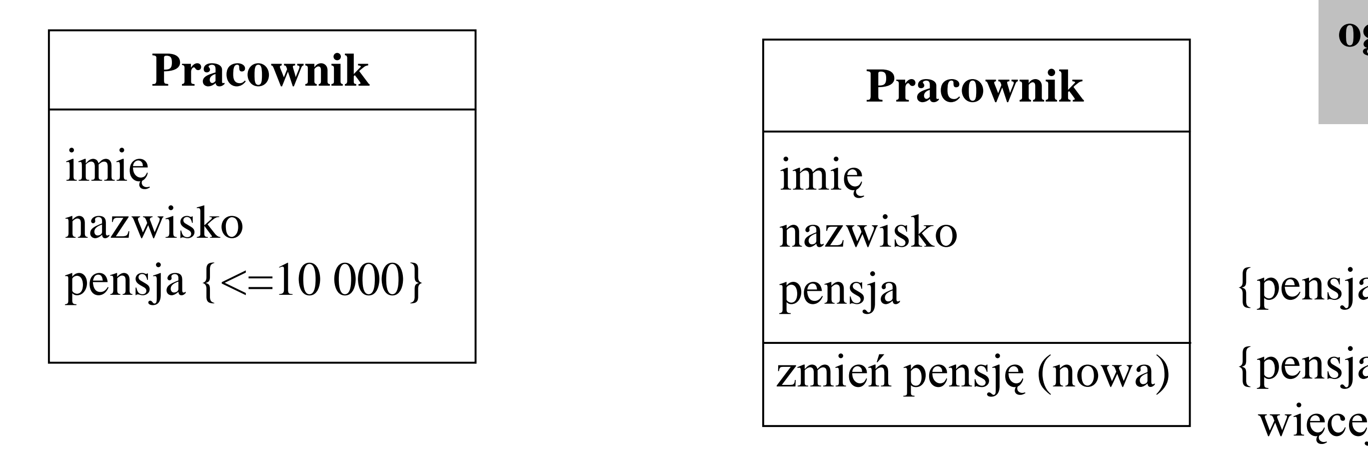
Wartości etykietowane są szczególnie przydatne do przechowywania zarówno informacji związanych z zarządzaniem projektem (jak w przykładzie powyżej), jak i do przechowywania szczegółów implementacyjnych.

Ograniczenia

Ograniczenia specyfikują restrykcje nakładane na elementy modelu. Mogą stanowić wyrażenia języka naturalnego czy języka formalnego (np. OCL w UML), mogą też przyjmować postać formuły matematycznej lub fragmentu kodu (czy też pseudokodu).

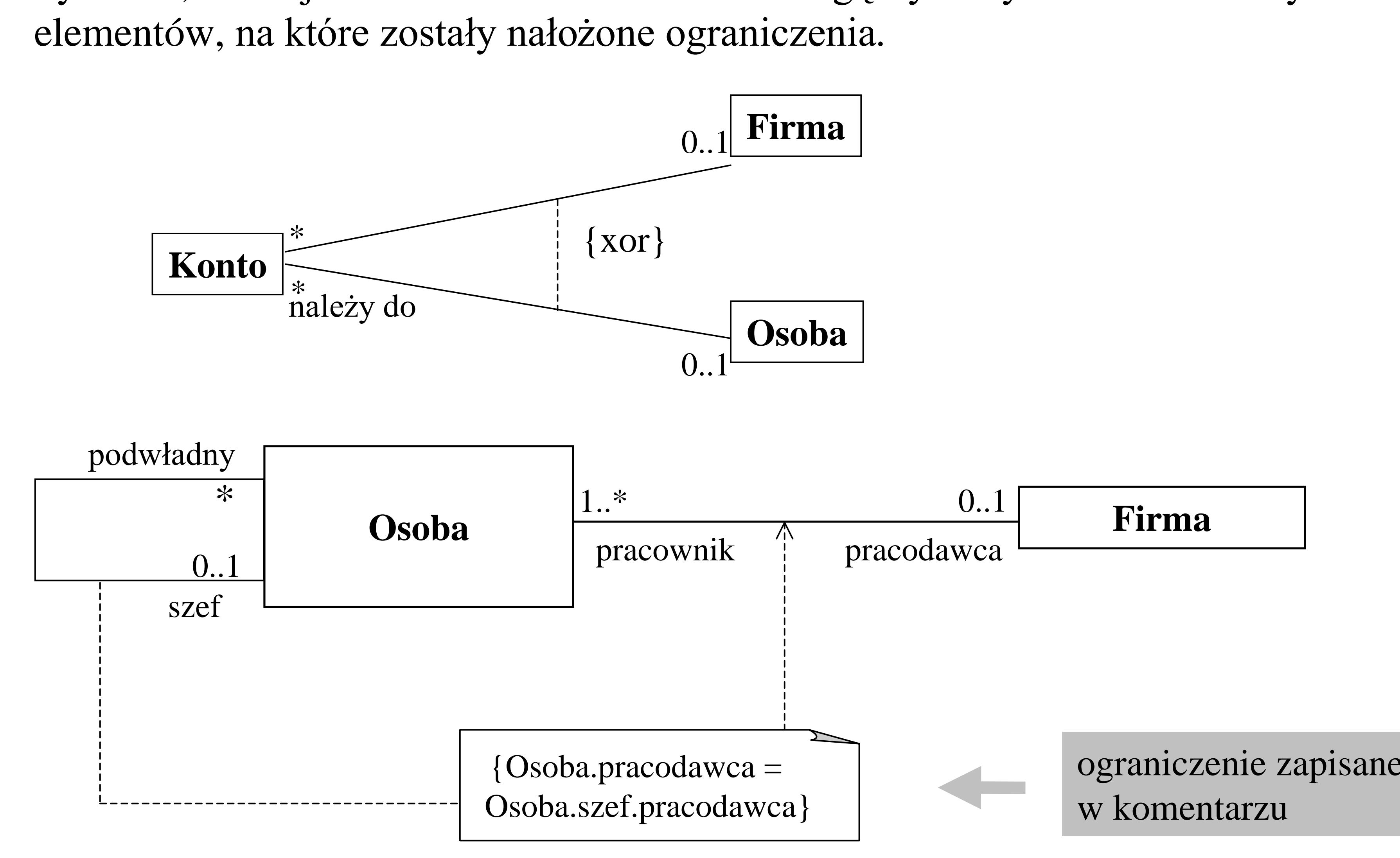
Notacja: są zawarte wewnątrz **{}** i umieszczane za elementem w klasie, lub poza klasą. Mogą też być umieszczane w komentarzu.

Pracownik
imię nazwisko pensja {<=10 000}



Ograniczenia; przykłady

Symbole, takie jak - - - - oraz - - - - > mogą być używane do wskazywania elementów, na które zostały nałożone ograniczenia.



Czy korzystać z mechanizmów rozszerzalności?

- UML dostarczyła kilku mechanizmów rozszerzalności, aby umożliwić projektantom wprowadzanie modyfikacji bez konieczności zmiany samego języka modelowania. Twórcy UML starali się w ten sposób (chociażby w pewnym stopniu) zaspokoić potrzeby specyficznych dziedzin problemowych czy środowisk programowych.
- Narzędzia mogą przechowywać wprowadzone modyfikacje oraz manipulować nimi bez konieczności wnikania w ich semantykę - modyfikacje z reguły są przechowywane w postaci łańcuchów znakowych.
- Narzędzia mogą ustanowić własną składnię i semantykę dla obsługi mechanizmów rozszerzalności.
- Należy pamiętać, że rozszerzenia stanowią z definicji odstępstwo od standardów UML, i że w naturalny sposób prowadzą do utworzenia pewnego dialektu UML, a to z kolei może prowadzić do problemów z przenaszalnością. Trzeba zawsze dobrze rozważyć zyski i straty, które mogą być poniesione dzięki korzystaniu z tych mechanizmów, szczególnie wtedy, gdy "stare" standardowe mechanizmy pracują wystarczająco dobrze.

Podsumowanie UML

- UML powstał w rezultacie połączonych wysiłków trzech znanych metodologów: G. Booch'a, I. Jacobson'a i J. Rumbaugh'a, których metodyki opanowały około 40% rynku zastosowań metodyk obiektowych. UML zyskuje coraz większą popularność jako składowa narzędzi CASE i prawdopodobnie będzie dominował przez wiele najbliższych lat w obszarze analizy i projektowania SI.
- UML, zgodnie z deklaracją twórców, nie ma ambicji być metodyką projektowania. Jest zestawem pojęć, oznaczeń i diagramów, który mogą być używane w dowolnej metodyce opartych o podstawowe pojęcia obiektowości. Pojęcia UML, w założeniu twórców, mają przykryć większość istotnych aspektów modelowanych systemów.
- Kwestia semantyki i pragmatyki tej notacji pozostaje mglista, co jest nieuchronnym skutkiem sprzeczności pomiędzy naturą procesów twórczych a ich formalizacją.
- UML jest składową standardu OMG (CORBA).
- Nie wszyscy są zachwyceni UML. Niektórzy specjaliści uważają go za twór niestabilny, zbyt ciężki, przereklamowany i źle zdefiniowany. UML ma konkurentów w postaci całego zestawu innych metodyk obiektowych.