Wprowadzenie do UML'a

Jolanta Sala
Halina Tańska

2018/2019

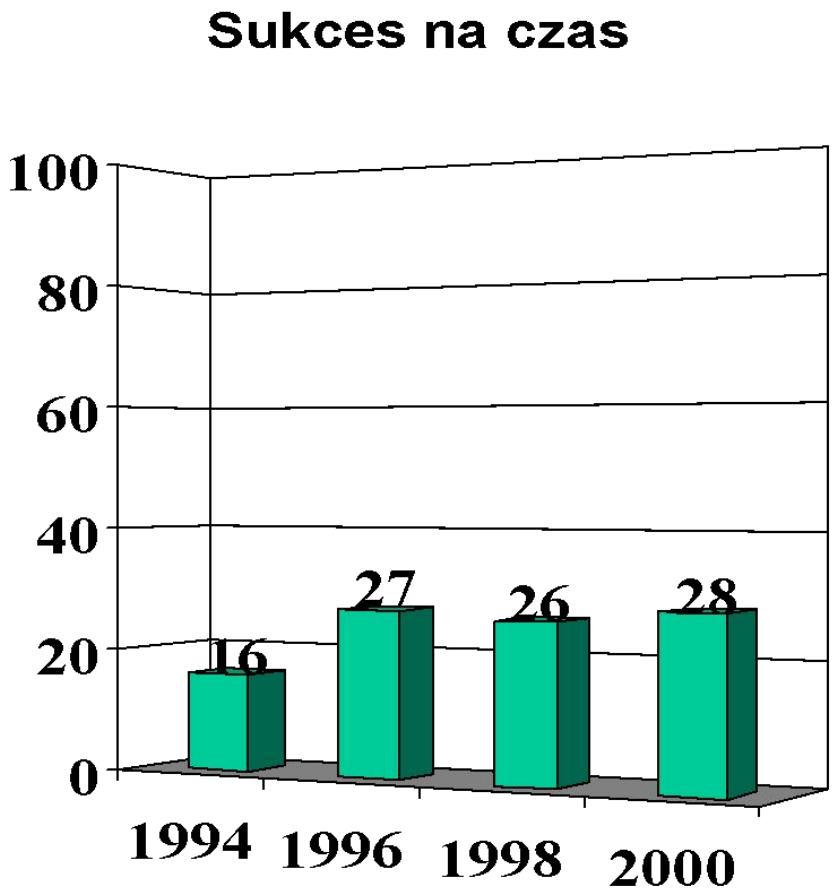
Cechy SI (aplikacji)

Producent oprogramowania powinien oferować *produkty*:

- wysokiej jakości
- spełniające wymagania użytkowników
- na czas
- zgodnie z planowanym budżetem

Uwaga: Wynikiem pracy zespołu nie jest kod godny nagrody Pulitzera. Najważniejsze są **dobre programy** spełniające zmienne wymagania użytkowników.

Kryzys inżynierii oprogramowania?



- 67% systemów spełnia wymagania użytkowników
- o 45% - przekroczenie zakładanej wielkości nakładów
- średnio o 63% jest przekraczany czas realizacji

Projektowanie systemów informatycznych

Co nam daje UML?

- ***wspólny język*** dla wszystkich grup zawodowych zaangażowanych w proces rozwoju systemu
- modelowanie systemów (nie tylko oprogramowania) z ***użyciem pojęć obiektowych***
- ***język modelowania***, użyteczny zarówno dla ludzi jak i dla maszyn
- ***notacja pośrednia***, pomost pomiędzy ludzkim rozumieniem struktury i działania programów, a kodem programów

Czym jest zunifikowany język modelowania UML

- Zunifikowany język modelowania (*Unifield Modeling Language* – UML) jest standardowym językiem modelowania graficznego, używanym do modelowania procesów biznesowych, oprogramowania oraz architektury systemów.
- UML został stworzony przez grupę *Object Management Group* (OMG), nie służy jedynie do modelowania rozwiązań zorientowanych obiektowo.
- Jest on językiem graficznym zaprojektowanym tak, aby osiągnąć jak największą elastyczność i możliwość dostosowania do konkretnych zadań.

Unified Modeling Language
korzenie

- języki programowania obiektowego zaczęły pojawiać się między 75, a 89 rokiem - w tym czasie metodycy programowania poszukiwali metod analizy i projektowania zgodnych z podejściem obiektowym.
- 1989-1994 liczba języków projektowania obiektowego wzrosła do > 50
- wielu użytkowników miało kłopoty ze znalezieniem odpowiedniej metody dla siebie

Unified Modeling Language
najpopularniejsze metody

- ***Booch*** (projektowanie i implementacja)
- ***OOSE Jacobson*** (wymagania i wysoko poziomowe projektowanie)
- ***OMT Rumbaugh*** (analiza i rozwijanie systemów przetwarzających duże ilości danych)
- ***Fusion***
- ***Shlaer - Mellor***
- ***Coad - Yourdon***

Unified Modeling Language (UML)

Prace nad UML rozpoczęto w 1994 roku
kiedy do Gradyego Boocha w Rational
dołączył James Rumbaugh

1995	Unified Method 0.8 (z połączenia Booch Method i OMT)
1996	UML 0.9 (dodano OOSE i inne metody)
1997	UML 1.0 – został przekazany OMG
	UML 1.1 – zaakceptowany przez OMG
1998	UML 1.2
1999	UML 1.3
2001	UML 1.4
2003	UML 1.5
2004	UML 2.0

Unified Modeling Language

przeznaczenie

Unified Modeling Language jest językiem do:

- obrazowania (komunikacja między członkami zespołu)
- specyfikowania
- tworzenia (inżynieria wprzód i wstecz)
- dokumentowania

artefaktów powstałych ***podczas budowania systemu informatycznego.***

Unified Modeling Language
model i modelowanie

- **Model jest uproszczeniem rzeczywistości.**
- Modelowanie prowadzi do lepszego zrozumienia systemu.
- Opanowanie ***złożonego systemu wymaga opracowania wielu wzajemnie powiązanych modeli***.
- W wypadku systemów informatycznych czynność tę mogą ułatwić ***różne spojrzenia na architekturę systemu w miarę rozwijania go***.

Czym jest model?

- Model to:
  - miniatura przedmiotu
  - wzorzec, na którym opierać się będzie produkcja czegoś, co jeszcze nie jest produkowane
  - projekt lub typ
  - przedmiot służący jako przykład do naśladowania lub przetwarzania
- Modelowanie to konstruowanie planu, zwłaszcza według pewnego wzorca

Po co modele?

Modele tworzymy dla:

- lepszego zrozumienia systemu
- specyfikacji pożądanej struktury i zachowanie systemu
- opisania architektury systemu i panowania nad nią (dekompozycja, upraszczanie, ponowne użycie)
- lepszego zarządzania ryzykiem

- Model pomaga szybciej zrozumieć projekt, wyjaśnić i rozbić na mniejsze części złożone problemy i scenariusze, jak również maksymalnie zbliżyć projekt do końcowego rezultatu przed rozpoczęciem wdrażania.

Unified Modeling Language

trzeba zrozumieć

- UML wskazuje sposób opracowywania i czytania poprawnych modeli.
- UML nie mówi nic o tym, jakie modele należy przygotować i kiedy należy to uczynić.

Czynności związane
z procesami tworzenia oprogramowania opisują
metody projektowania.

Język UML

- UML to język służący do ***specyfikowania, konstruowania, obrazowania oraz dokumentowania składowych systemów oprogramowania***. Twórcami UML są: Gary Booch, Ivar Jacobson, oraz James Rumbaugh.
- ***UML to język a nie metodyka konstruowania oprogramowania***, tzn. nie podaje wskazówek dotyczących sposobu organizacji poszczególnych faz procesu wytwórczego.

UML

- UML stanowi „wspólny język” dla
  - analityków,
  - programistów,
  - testerów,
  - architektów oprogramowania,
  - projektantów baz danych ,
  - wielu innych pracowników związanych z projektowaniem oraz produkcją oprogramowania.
- Dzięki UML twórcy oprogramowania mogą poznać zasady i warunki prowadzenia produkcji, a także sposób tworzenia oprogramowania i architektury systemów.

Co można modelować przy użyciu UML?

- UML umożliwia *modelowanie wielu różnych aspektów działalności, od procesów biznesowych i samego biznesu po funkcje powiązane z dziedzinami IT*, takimi jak projektowanie baz danych, architektury aplikacji czy sprzętu i wielu innych.
- Projektowanie oprogramowania i systemów informatycznych jest skomplikowanym zadaniem, wymagającym skoordynowanej pracy różnych grup mających różne funkcje: określenie potrzeb firm i wymagań systemu, integrację komponentów, konstruowanie baz danych, produkowanie sprzętu wspierającego działanie oprogramowania.

Typy modeli i obszary ich stosowania

Typ modelu	Obszar stosowania
Biznesowy	Procesy biznesowe, postęp prac nad projektem, organizacja
Wymagań	Określenie wymagań i komunikacja
Architektury	Projekt wysokiego poziomu dla tworzonego systemu, interakcja między różnymi systemami, wyjaśnienie projektu osobom pracującym nad systemem
Aplikacji	Architektura projektów niższych poziomów wewnątrz systemu
Baz danych	Projektowanie struktury bazy danych i określenie sposobu interakcji z aplikacją (aplikacjami) w danym zastosowaniu

Kto powinien budować modele?

- *Analitycy tworzą i projektują modele biznesowe* dotyczące teraźniejszej sytuacji i jej przyszłego rozwoju. Często budują modele aplikacji na poziomie architektury.
- *Programiści zajmujący się produkowaniem aplikacji* pisząc kod powinni go modelować przed napisaniem. Dzięki wykorzystaniu narzędzi generujących gotowy kod na podstawie opracowanego modelu można zautomatyzować nużące tworzenie typowych fragmentów kodu.
- *Testerzy oprogramowania* mogą stosować modele jako pomoc przy przeprowadzania testów.

Metody projektowania

Zbiór częściowo uporządkowanych kroków, których wykonanie prowadzi do ***osiągnięcia ustalonego celu***.

W inżynierii oprogramowania tym celem jest ***udostępnienie oprogramowania, które spełnia potrzeby przedsiębiorstwa***.

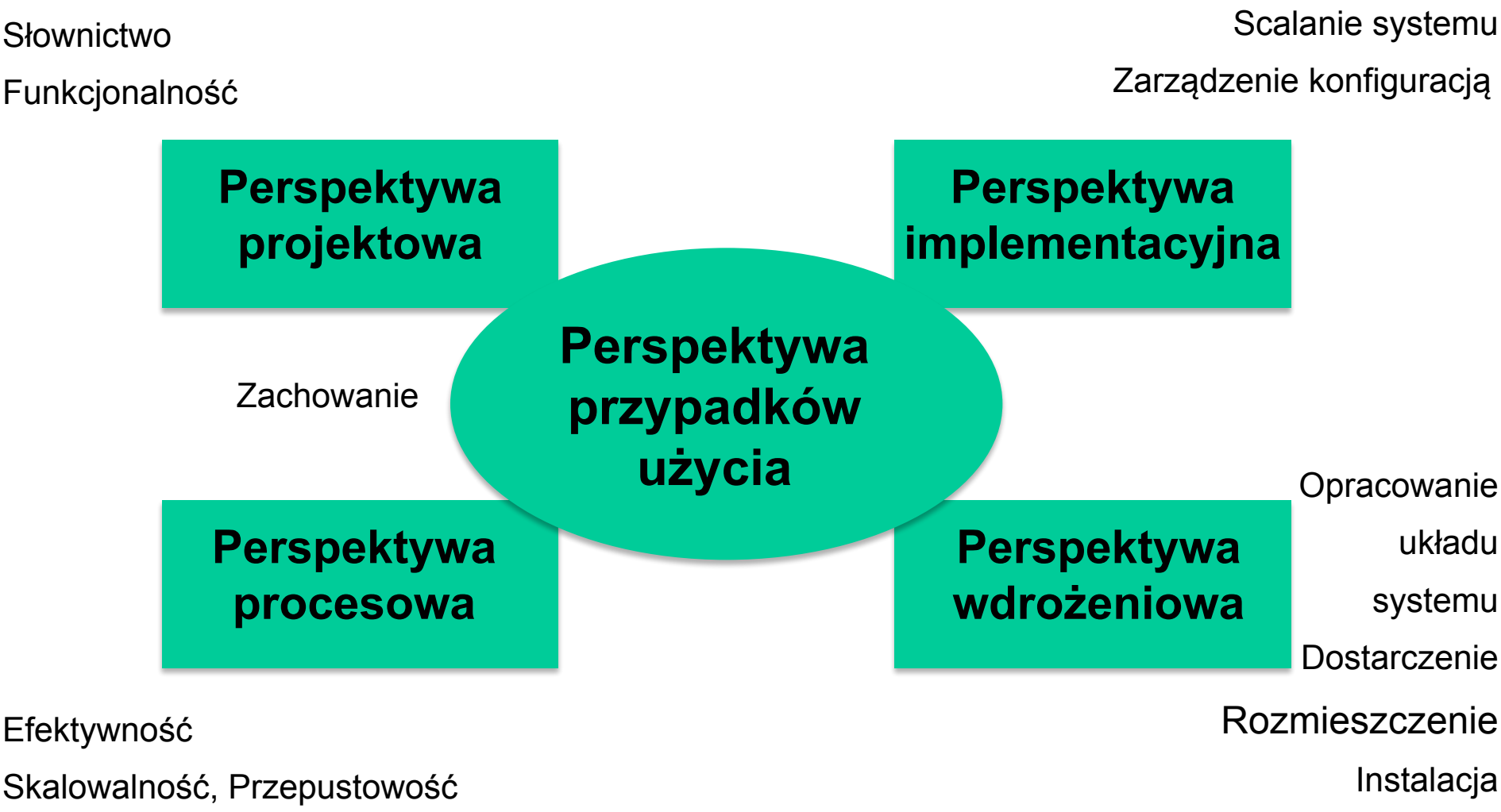
Rational Unified Process

Metodyka RUP obejmuje cały cykl życia projektu:

- analizę
- projektowanie
- zapewnianie jakości w kolejnych iteracjach rozwoju systemu

Rational Unified Process

Perspektywy – punkty widzenia *różnych grup użytkowników*



Perspektywy

- W trakcie konstruowania dowolnego modelu (diagramu) powinny być brane pod uwagę następujące trzy perspektywy:
  - ***Perspektywa pojęciowa (koncepcyjna)*** – przedstawia pojęcia funkcjonujące w dziedzinie problemowej. W szczególności analizowane są operacje wykonywane na bytach, cechy opisujące byty oraz istniejące pomiędzy bytami różnego rodzaju związki semantyczne. Perspektywa pojęciowa nie powinna odnosić się do środowiska implementacji.
  - ***Perspektywa projektowa (procesowa)*** – uwzględnia środowisko implementacji, przy czym nacisk położony jest bardziej na projektowanie interfejsów niż kodowanie.
  - ***Perspektywa implementacyjna*** – związana jest bezpośrednio z wytwarzaniem kodu.
- Zrozumienie perspektywy, która była brana pod uwagę w trakcie konstruowania danego modelu, jest ważnym czynnikiem mającym wpływ na prawidłowe zinterpretowanie modelu. Właściwe zrozumienie perspektywy jest warunkiem koniecznym poprawnego wykorzystania modelu.
- Często analitycy i projektanci lekceważą konieczność wyraźnego zakreślenia granic między perspektywami i konstruują swoje modele z perspektywy implementacyjnej.

Perspektywy

- Konstruując model powinno się *uwzględniać jedną, wyraźnie określoną perspektywę*.
- Aby poprawnie zinterpretować model, należy *wiedzieć, z jakiej perspektywy został on skonstruowany*.
- Modele, tak jak i całość projektu, zawsze powstają w *sposób iteracyjny i przyrostowy*.

Znaczenie diagramów

- Diagram – schemat przedstawiający zbiór bytów, jest swego rodzaju rzutem systemu
- Diagram przedstawia system z określonej perspektywy (z określonego punktu widzenia)
- Diagram ma najczęściej postać grafu
  - Wierzchołki grafu – elementy
  - Gałęzie grafu – związki
- Teoretycznie diagram może zawierać dowolną kombinację elementów i związków
- W praktyce wprowadza się pewne kombinacje elementów i relacji, które można umieszczać na diagramach określonego rodzaju

Faza analizy

W podejściu obiektowym w fazie analizy najczęściej wykorzystywane są:

- ***Model przypadków użycia*** – specyfikujący funkcjonalność systemu widzianą z perspektywy jego przyszłych użytkowników. Model ten jest przedstawiany w postaci diagramu przypadków użycia.
- ***Model obiektowy*** – opisujący statyczną budowę systemu. Model ten jest przedstawiany w postaci diagramu klas i diagramu obiektów. Główna różnica pomiędzy diagramem klas a diagramem obiektów polega na tym, że diagram klas przedstawia klasy oraz może przedstawiać obiekty, podczas gdy diagram obiektów przedstawia obiekty, ale nie przedstawia klas. Czynności prowadzące do powstania modelu obiektowego określa się terminem analiza statyczna.
- ***Model dynamiczny (model zachowań)*** – służący do identyfikowania operacji niezbędnych systemowi do realizacji zadań; najczęściej rozważanymi rodzajami zadań są przypadki użycia. Model ten jest przedstawiany w postaci m.in. diagramów stanów i diagramów komunikacji między obiektami. Zidentyfikowane metody nanoszone są na stworzony uprzednio wstępny diagram klas uzupełniając w ten sposób definicje jego klas. Czynności prowadzące do powstania modelu dynamicznego określa się terminem analiza dynamiczna.

Faza projektowania

- W fazie projektowania *model pojęciowy jest dostosowywany do wymagań niefunkcjonalnych* oraz do *ograniczeń środowiska implementacji*, stając się modelem logicznym.
- Podstawowym zadaniem tej fazy jest określenie najlepszej strategii dla sposobu zaimplementowania systemu.
- Wyniki powinny być szczegółowe na tyle, aby w trakcie implementacji nie powstały niejednoznaczności; stopień szczegółowości jest uzależniony od doświadczenia programistów i złożoności problemu.

Faza implementacji

- Podczas implementacji, *model logiczny jest przekształcany w model fizyczny*, czyli kod.
- Model logiczny oraz model fizyczny stanowią modele rozwiązania.

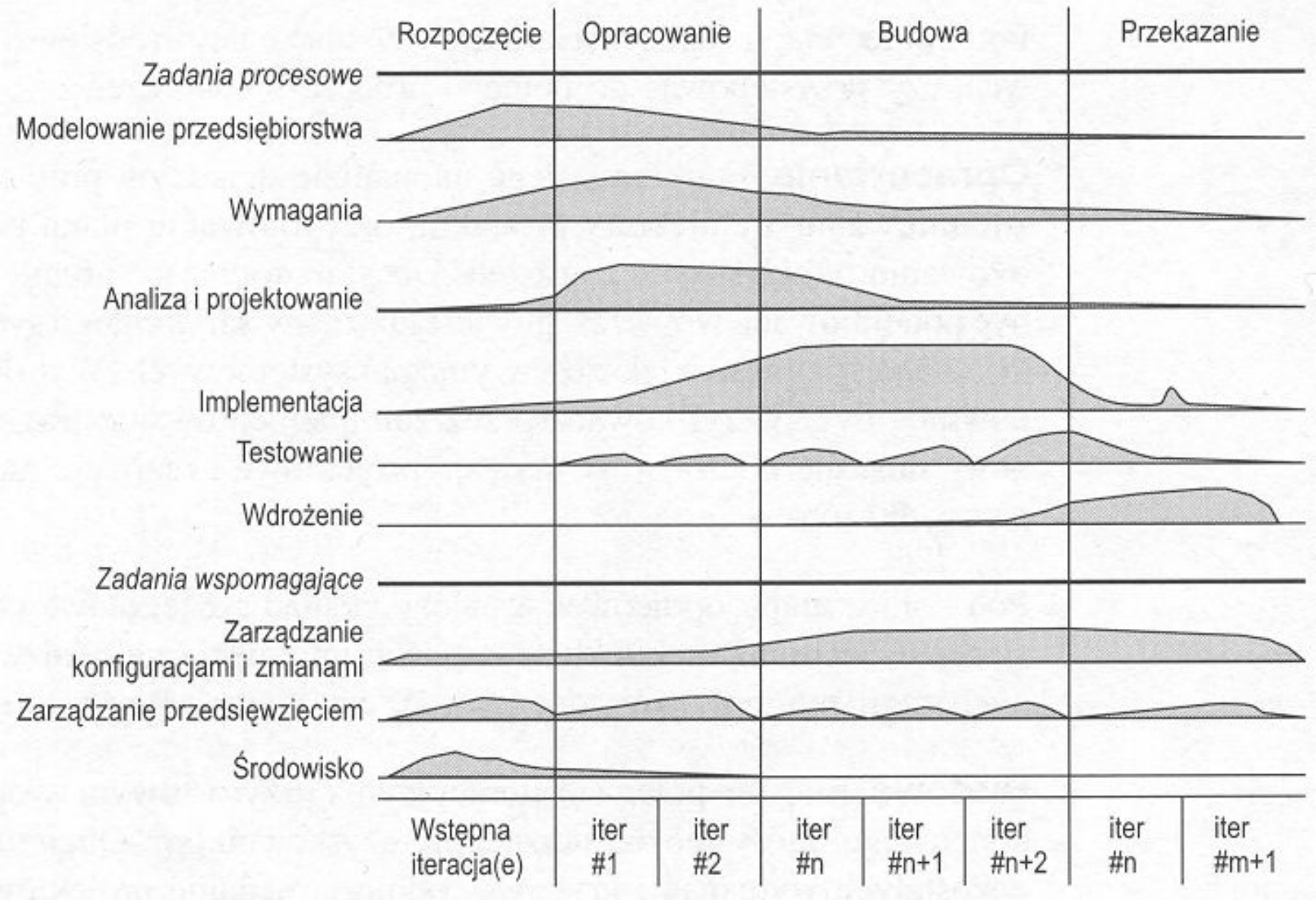
Rational Unified Process

Metoda Ad-hoc

Dla wielu programistów myślenie o implementacji i implementacja jest tym samym, lecz metoda ta posiada szereg wad:

- komunikacja często nieprecyzyjna
- nie da się opanować systemu przez analizę kodu (modele wspomagają spojrzenie na cały system)
- informacje o modelach powstałych w głowie programisty często przepadają

Rational Unified Process



Unified Modeling Language

UML ***nie jest językiem programowania graficznego***, ale modele w nim zapisane mogą w **100%** być przetworzone w język programowania:

- Java
- Visual Basic
- C++
- C#
- ... i wiele innych obiektowych języków programowania

Unified Modeling Language

Diagramy struktury:

- diagram klas (*class diagram*)
- diagram obiektów (*object diagram*)
- diagram komponentów (*component diagram*)
- diagram pakietów (*package diagram*)
- diagram wdrożenia (*deployment diagram*)
- zbiorowy diagram komponentów (*composite structure diagram*)

Diagramy czynności:

- diagram czynności (*activity diagram*)
- diagram przypadków użycia (*use case diagram*)
- diagram maszyny stanów (*state machine diagram*)
- diagram sekwencji (*sequence diagram*)
- diagram komunikacji (*communication diagram*)
- diagram przeglądu współdziałania (*interaction overview diagram*)
- diagram czasowy (*timing diagram*)

Język UML – definiuje zestaw diagramów

- Diagram przypadków użycia – służy do modelowania funkcjonalności systemu z punktu widzenia jego przyszłych użytkowników
- Diagram klas - służy do modelowania struktury danych przechowywanych w systemie, zawiera klasy i może zawierać obiekty
- Diagram obiektów - służy do modelowania struktury danych przechowywanych w systemie; zawiera wyłącznie obiekty
- Diagramy dynamiczne - służą do modelowania zachowań:
  - Diagram stanów
  - Diagram aktywności
  - Diagram interakcji: diagram sekwencji oraz diagram współpracy
- Diagramy implementacyjne:
  - Diagram komponentów
  - Diagram wdrożeniowy
- Diagram pakietów - służy do celów organizacyjnych.
- Diagramy te pozwalają opisać projektowany system z wielu perspektyw, razem składają się na jego szczegółowy opis.

Modele a diagramy

Główny obszar działania	Modele	Diagramy UML	Podstawowe pojęcia
Struktura	Model obiektowy	**Diagram klas** **Diagram obiektów**	Klasa, obiekt, asocjacja, generalizacja, zależność, realizacja, interfejs
Struktura	Model przypadków użycia	**Diagram przypadków użycia**	Aktor, przypadek użycia, inkluzja, ekstensja, generalizacja
Struktura	Model implementacji	**Diagram komponentów** **Diagram wdrożeniowy**	Komponent, interfejs, zależność, realizacja Węzeł, komponent, zależność, lokacja
Dynamika	Model dynamiczny	**Diagram stanów** **Diagram aktywności** **Diagram interakcji**	Stan, zdarzenie, przejście, akcja, aktywność Stan, aktywność, fork, join, romb decyzyjny Interakcja, współpraca, komunikat, aktywacja
Zarządzanie	Model zarządzania	**Diagram pakietów**	Pakiet, podsystem
Rozszerzalność	Wszystkie modele	Wszystkie diagramy	Stereotyp, wartość etykietowana, ograniczenie

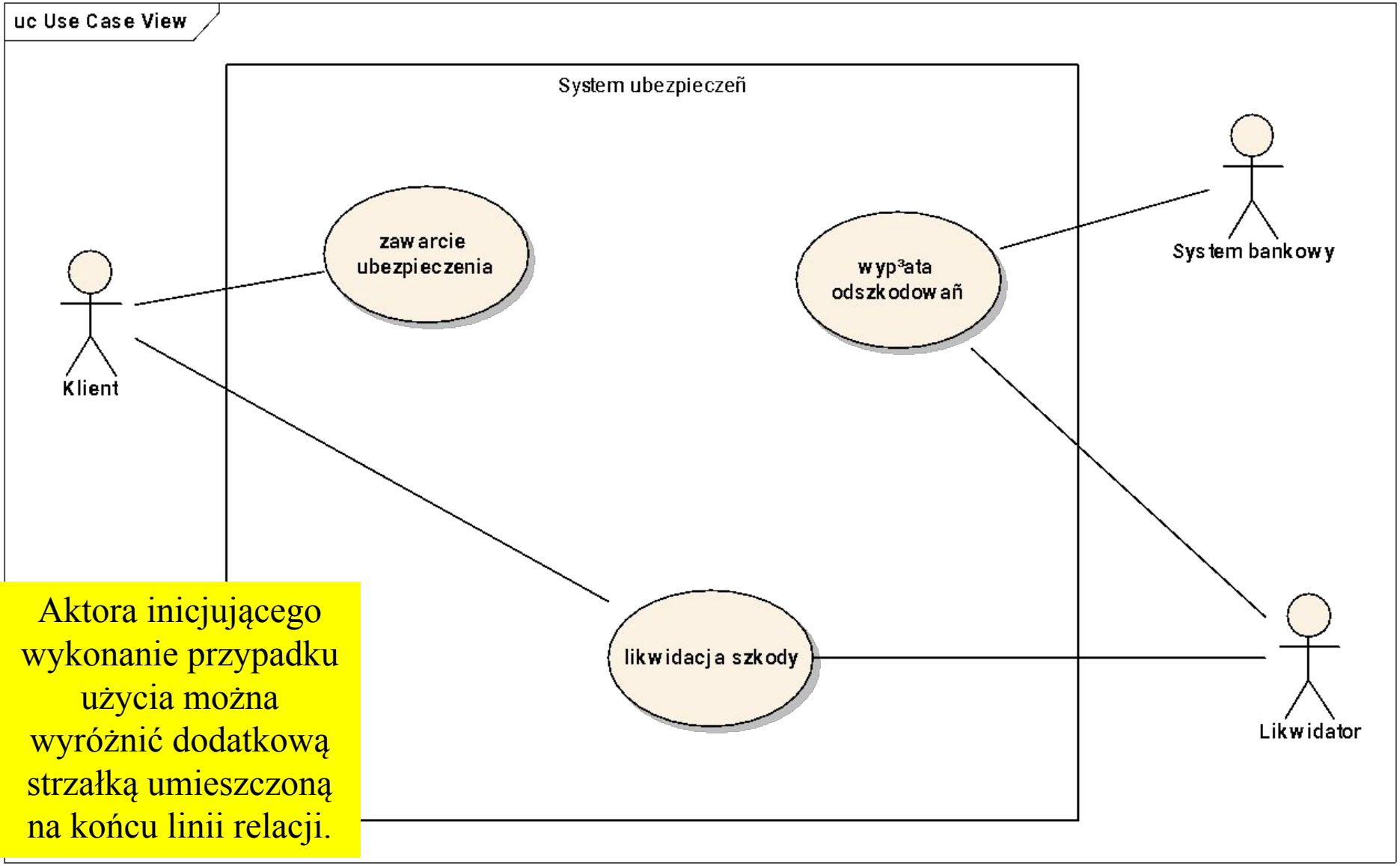
Modele obiektowe (UML 2.1)

Rodzaj diagramu	Przeznaczenie
Diagram przypadków użycia	Identyfikacja kategorii użytkowników oraz sposobów używania przez nich systemu
Diagram klas	Modelowanie klas obiektów i ich wzajemnych relacji
Diagram czynności (diagram aktywności)	Modelowanie procesów biznesowych, scenariuszy przypadków użycia lub algorytmów
Diagram maszyny stanowej	Modelowanie historii życia obiektu – jego stanów i możliwych przejść między stanami
Diagram komponentów	Modelowanie fizycznych składników oprogramowania, ich zależności i interfejsów
Diagram pakietów	Grupowanie elementów modelu w pakiety i pokazanie wzajemnych zależności pakietów
Diagram rozmieszczenia (diagram wdrożenia)	Modelowanie konfiguracji sprzętowych i programowych komponentów systemu
Diagram sekwencji (diagram przebiegu)	Modelowanie czasowej sekwencji wymiany komunikatów podczas współpracy obiektów, pakietów lub komponentów
Diagram komunikacji	Modelowanie przepływu komunikatów podczas współpracy obiektów, pakietów lub komponentów
Diagram struktury złożonej	Modelowanie wewnętrznej struktury złożonej klasy, komponentu lub przypadku użycia
Diagram przeglądu interakcji	Modelowanie przepływu sterowania w procesie biznesowym lub systemie
Diagram obiektów	Modelowanie chwilowej konfiguracji obiektów oprogramowania
Diagram czasowy	Modelowanie uzależnień czasowych

Diagram przypadków użycia

- Służy do modelowania dynamiki systemu
- Przedstawia zbiór przypadków użycia, aktorów oraz związki między nimi
- Jest szczególnie przydatny w obrazowaniu, specyfikowaniu i dokumentowaniu zachowania
- Przedstawia byty z zewnątrz (wnętrze pozostaje ukryte)

Diagram przypadków użycia firmy ubezpieczeniowej



Aktora inicjującego wykonanie przypadku użycia można wyróżnić dodatkową strzałką umieszczoną na końcu linii relacji.

Aktorzy diagramu PU modelują zewnętrzne obiekty współpracujące z budowanym systemem.

Diagram przypadków użycia

- Diagram przypadków użycia (ang. *Use Case Diagram*) jest diagramem, który ***przedstawia funkcjonalność systemu wraz z jego otoczeniem***
- Diagramy przypadków użycia pozwalają na ***graficzne zaprezentowanie własności systemu tak, jak są one widziane po stronie użytkownika***
- Diagramy przypadków użycia służą do ***zobrazowania usług, jakie są widoczne z zewnątrz systemu***

Diagramy przypadków użycia

- specyfikują ***wymagania stawiane systemowi***
- obrazują ***zachowanie systemu***
- modelują ***otoczenie systemu***
- nie definiują sposobu implementacji systemu
- opisują jedynie najważniejsze aspekty zachowania systemu
- nie są przesadnie szczegółowe
- są ***platformą do komunikacji analityka z klientem***

Diagram przypadków użycia

Kluczowymi elementami są:

- aktorzy (*actor*)
- przypadki użycia (*use case*)
- związki (*association*)

Dodatkowo diagram może zwierać:

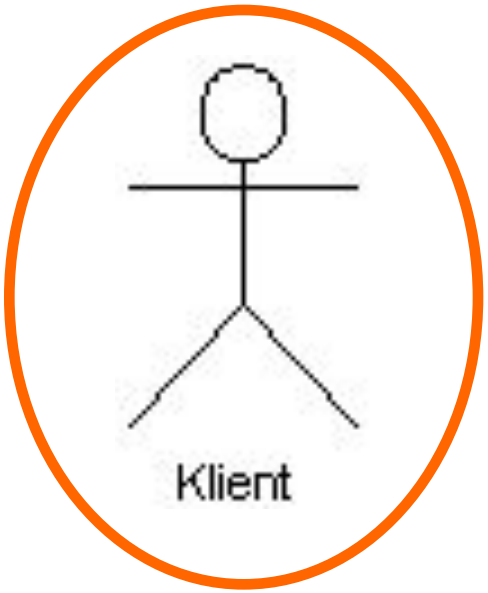
- notatki (*note*)
- ograniczenia (*constraints*)
- pakiety (*packages*)

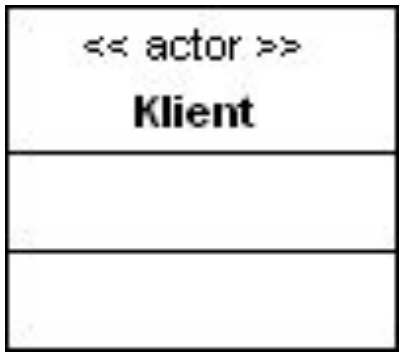
Aktor

- Aktor (ang. *actor*) jest ***funkcją, jaką pełni użytkownik w stosunku do systemu oraz przypadków użycia***.
- Aktor reprezentuje ***spójny zbiór ról, które są odgrywane przez użytkowników przypadku użycia w czasie interakcji z tym przypadkiem***.
- Aktorem może być ***człowiek, urządzenie, inny system lub czas***.
- Aktor nie musi być fizycznym obiektem. Istotne by pełnił określoną funkcję wobec systemu i przypadku użycia, którego używa.

Aktor

Aktor to ***użytkownik*** lub ***inny system***, który wchodzi w ***interakcję*** z naszym systemem.





Najczęściej używany symbol

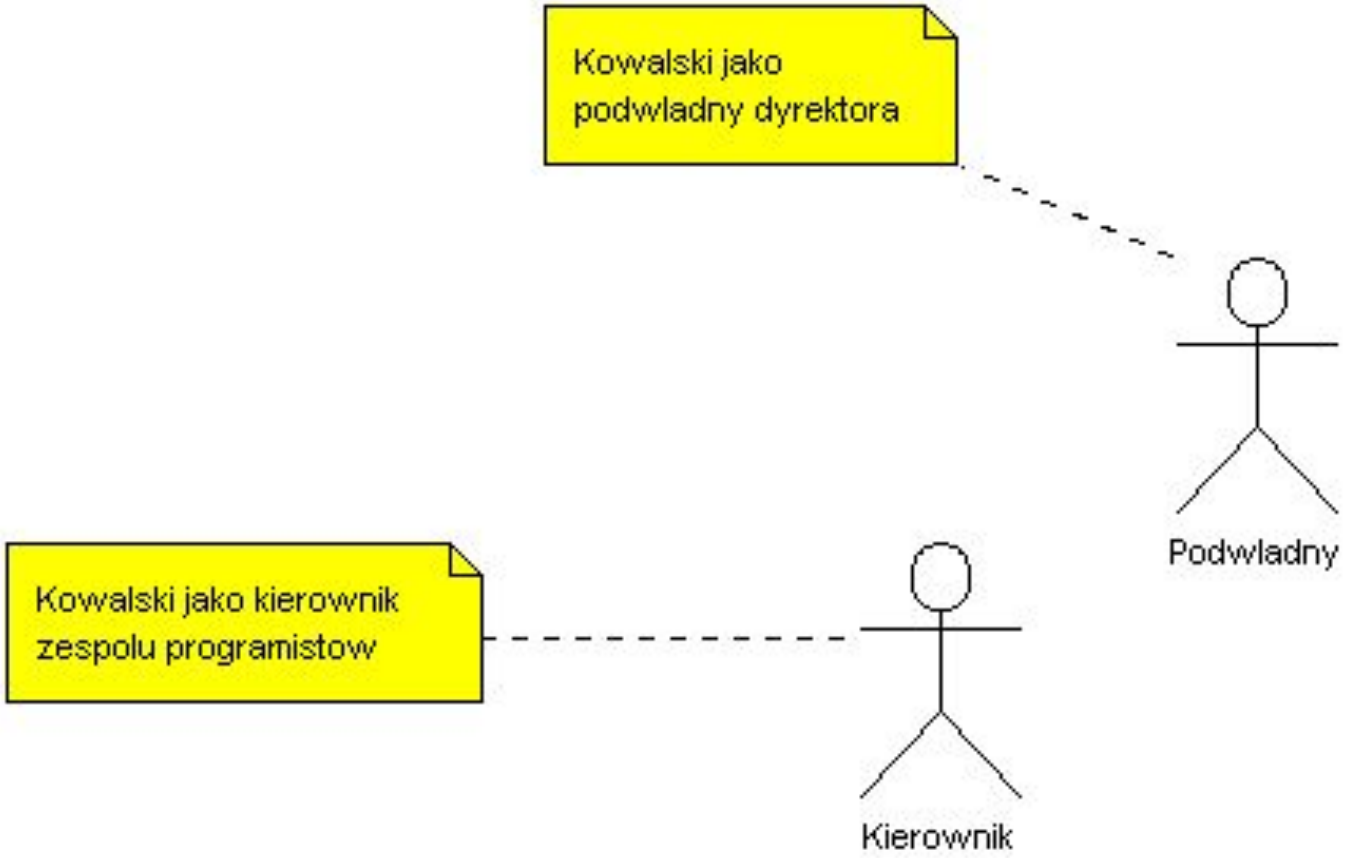
Aktor

- Aktorzy stanowią otoczenie systemu (nie są częścią systemu)
- Aktor może aktywnie wymieniać informacje z systemem (dostarczać informacje i pobierać)
- Aktor może wywoływać akcje w systemie
- **Aktorami mogą być**:
  - człowiek
  - urządzenie
  - inny system

Aktor

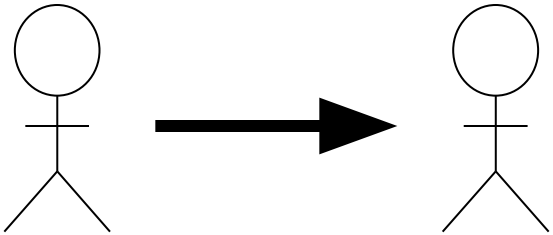
Aktor reprezentuje ***rolę*** w jakiej człowiek, inny system bądź urządzenie może się wcielić w interakcji z naszym systemem.

Jeden Kowalski w wielu rolach



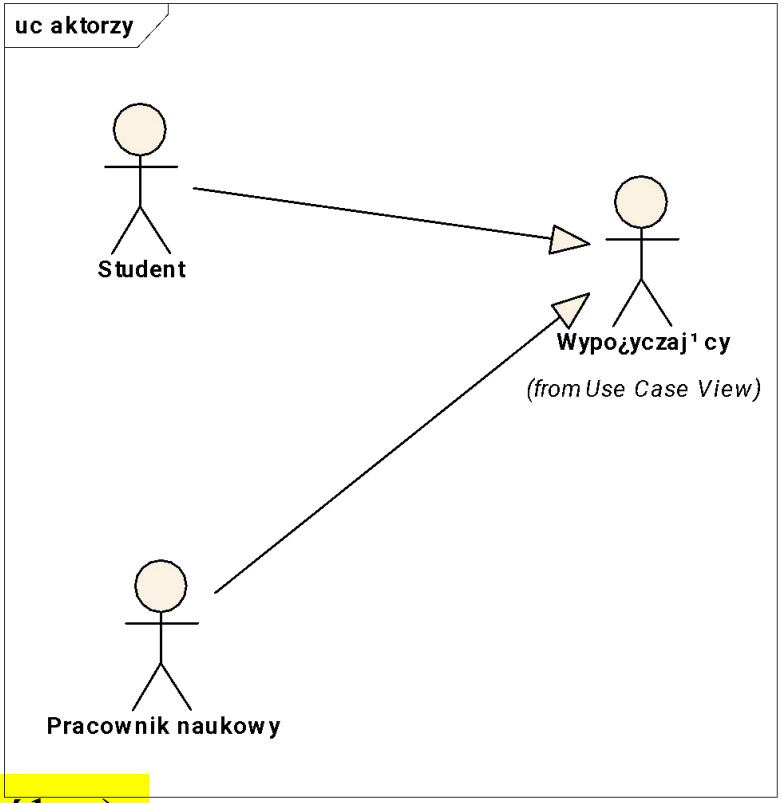
Generalizacja

Potomek zawsze może zastąpić przodka



Student Użytkownik

potomek przodek

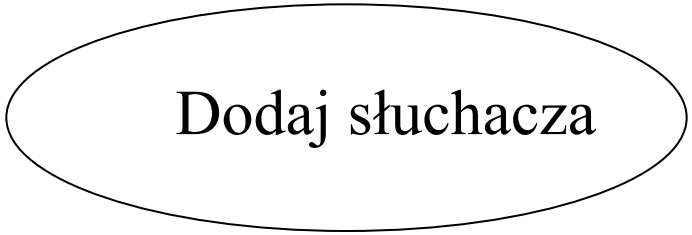


Grot strzałki wskazuje na przodka (klasę ogólną)

Związek generalizacji to związek pomiędzy elementem ogólnym (***nadklasa*** lub ***przodek***) a specyficznym jego rodzajem zwanym ***podklasą*** lub ***potomkiem***. Element specyficzny jest całkowicie zgodny z elementem ogólnym i zawiera dodatkową informację. Egzemplarz elementu specyficznego może być użyty wszędzie tam, gdzie dopuszcza się egzemplarz elementu ogólnego.

Przypadek użycia

- Przypadek użycia (PU) jest ***graficzną reprezentacją wymagań funkcjonalnych***
- Definiuje zachowanie systemu bez informowania o wewnętrznej strukturze i narzucania sposobu implementacji
- Przypadek użycia pozwala na zdefiniowanie przyszłego, spodziewanego zachowania systemu

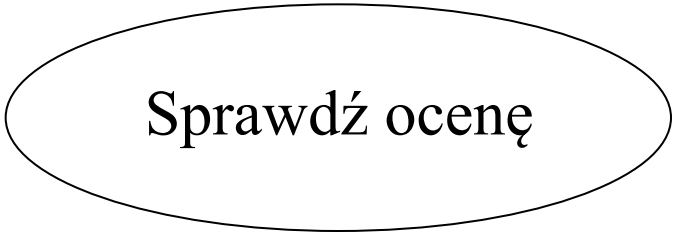


Przypadek użycia

- Kwant funkcjonalności systemu dostarczający aktorowi usług o mierzalnej wartości (*I. Jacobson*).
- Czynność, której wykonanie bezpośrednio świadczy o efektywności pracy
- Nazwana lub dobrze określona interakcja pomiędzy użytkownikiem a systemem komputerowym

Przypadek użycia

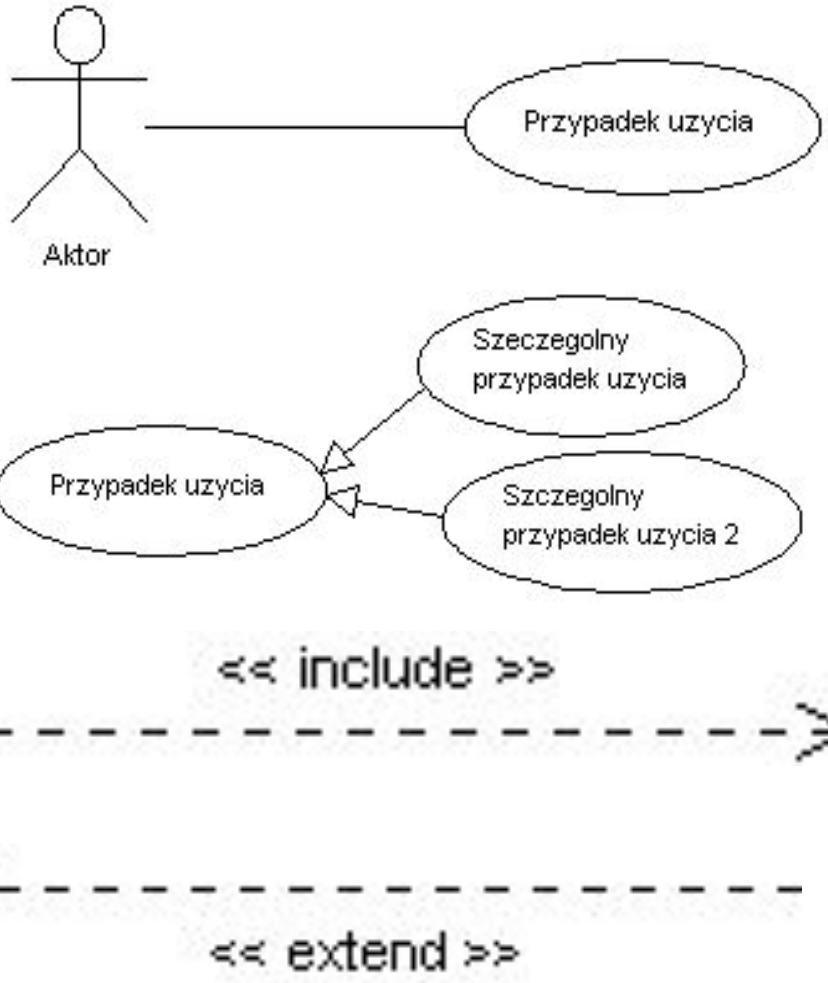
- ***Przypadek użycia musi być w interakcji, chociaż z jednym aktorem***. Wyjątek stanowi sytuacja, gdy przypadek użycia jest połączony z innym przypadkiem użycia związkiem rozszerzenia lub zawierania.
- Przypadek użycia to ***zbiór scenariuszy powiązanych ze sobą wspólnym celem użytkownika***.



Związki

w diagramach przypadków użycia

- powiązania (tylko między aktorem a przypadkiem użycia)
- uogólnienia
- zawierania – include
- rozszerzenia - extend

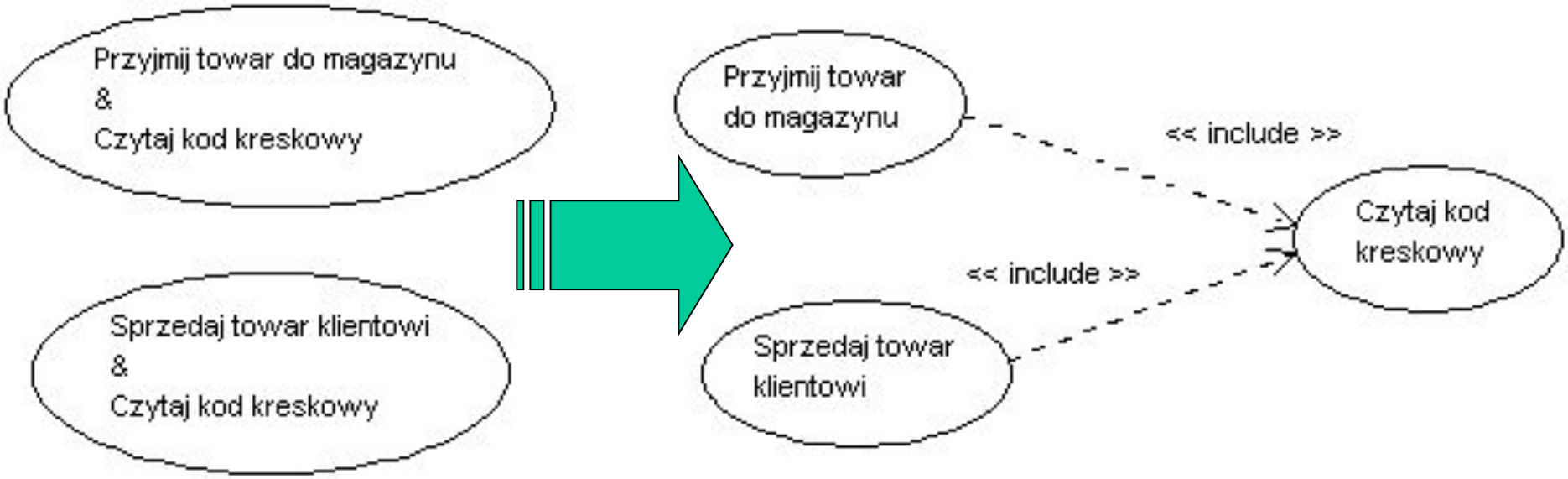


Związki

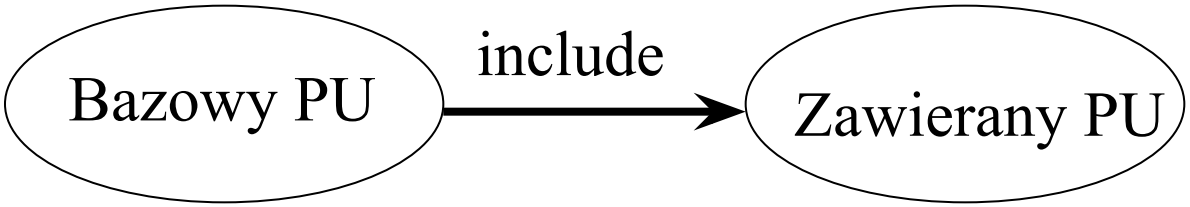
Związek zawierania stosuje się w celu uniknięcia wielokrotnego opisywania tego samego ciągu zdarzeń.

Przyjmij towar... ***zawsze zawiera*** *Czytaj kod...*

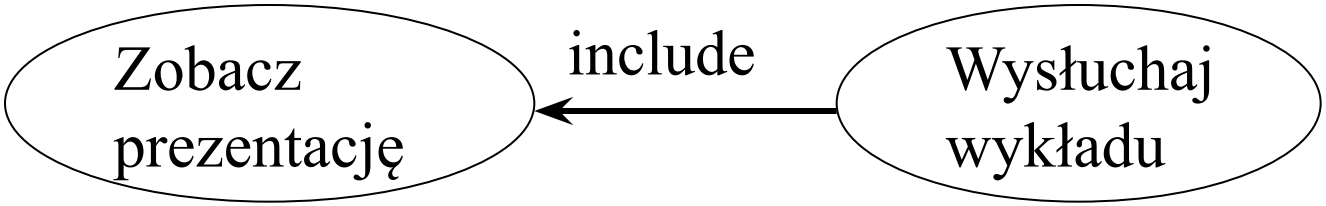
<< include >>



Związek zawierania



Związek zawierania (ang. Include) polega na tym, że bazowy przypadek użycia rozszerza swoją funkcjonalność o zachowanie innego przypadku użycia. Zawierany przypadek użycia nie jest autonomiczny.

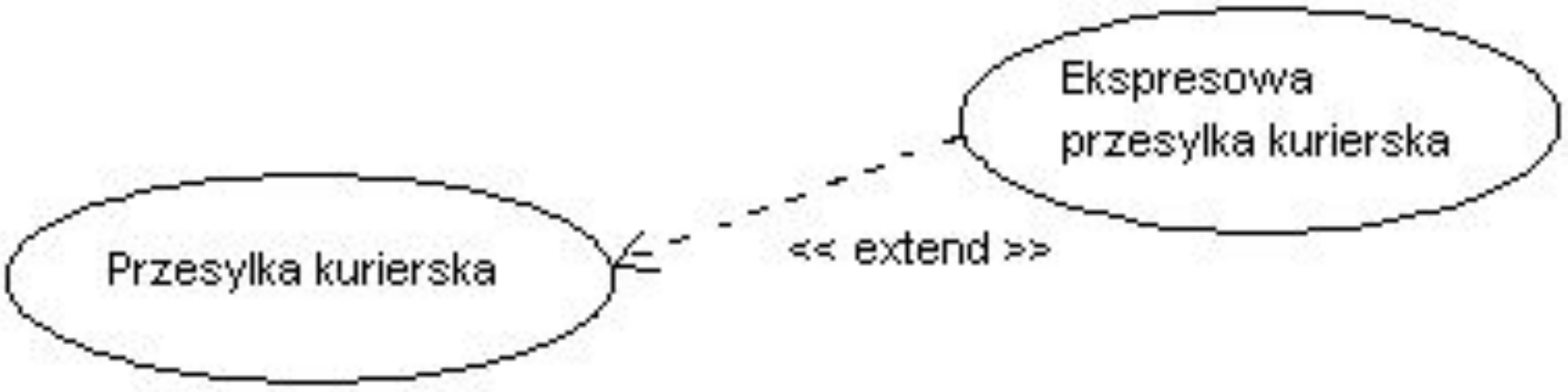


Związki

Związek rozszerzenia służy do modelowania fragmentów przypadku użycia postrzeganych przez użytkownika jako ***opcjonalne zachowanie systemu***.

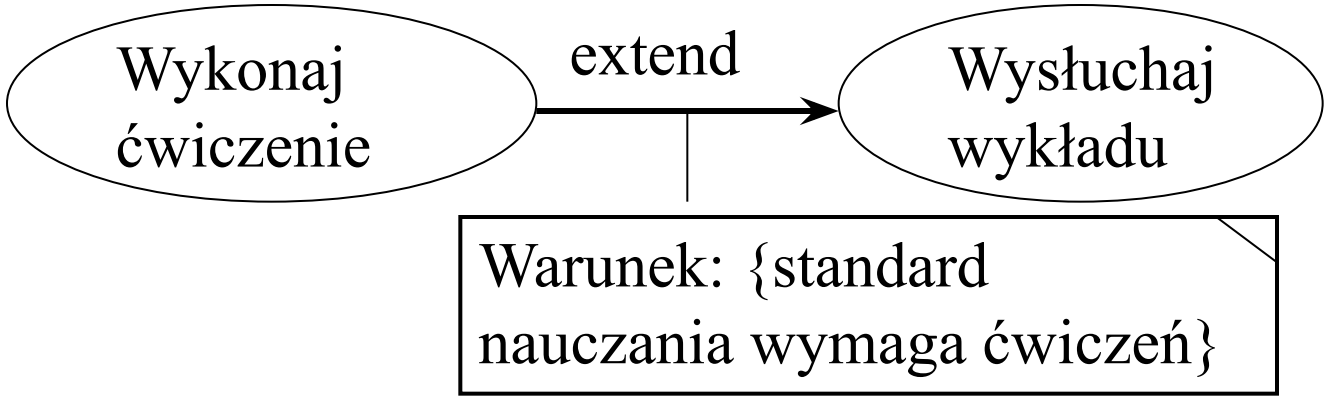
Ekspresowa... ***opcjonalnie rozszerza*** *Przesyłkę...*

<< extend >>

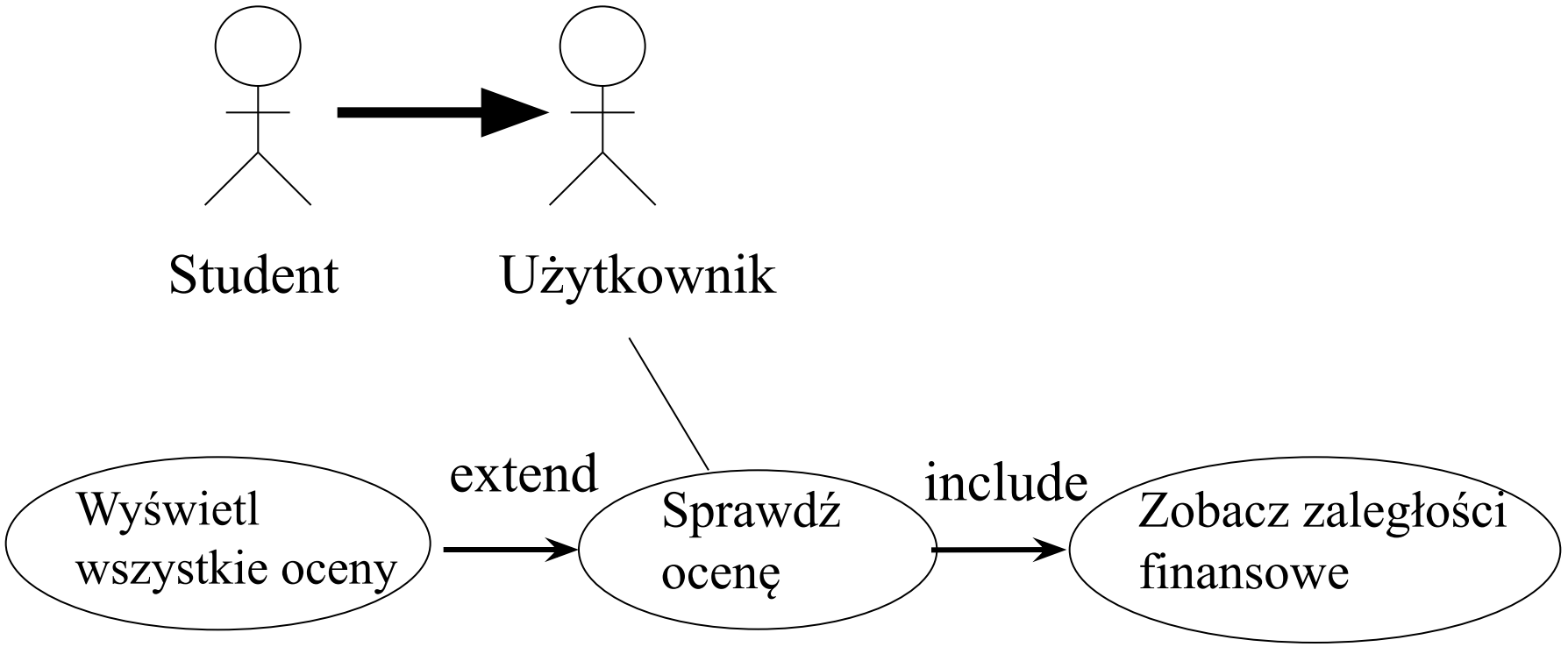


Związek rozszerzania

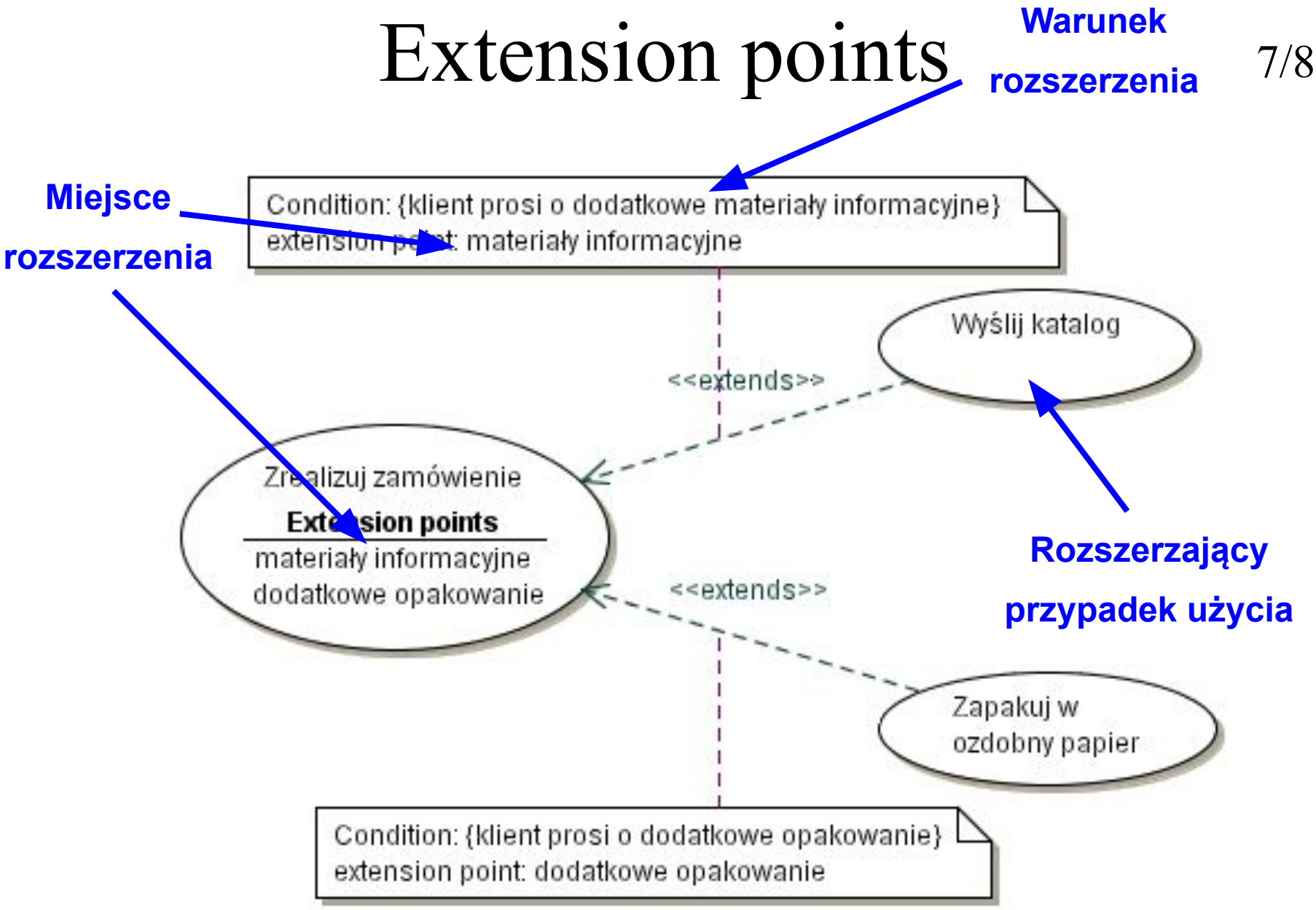
Związek rozszerzania (ang. Extend) wskazuje, że dany przypadek użycia opcjonalnie rozszerza funkcjonalność bazowego przypadku użycia. Funkcjonalność bazowego przypadku użycia jest rozszerzana o inny przypadek użycia po spełnieniu określonego warunku.



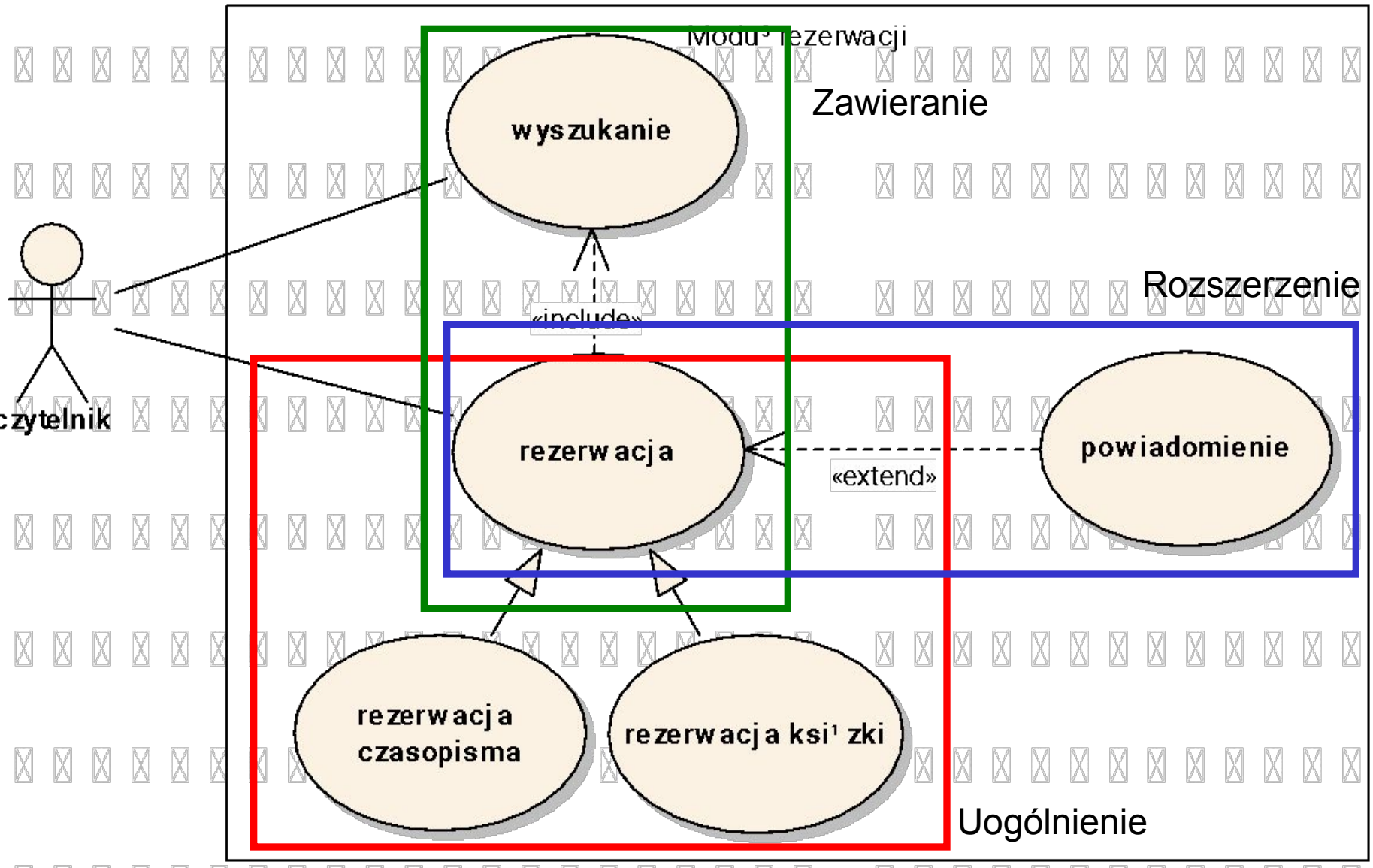
Związek zawierania i rozszerzania



Extension points



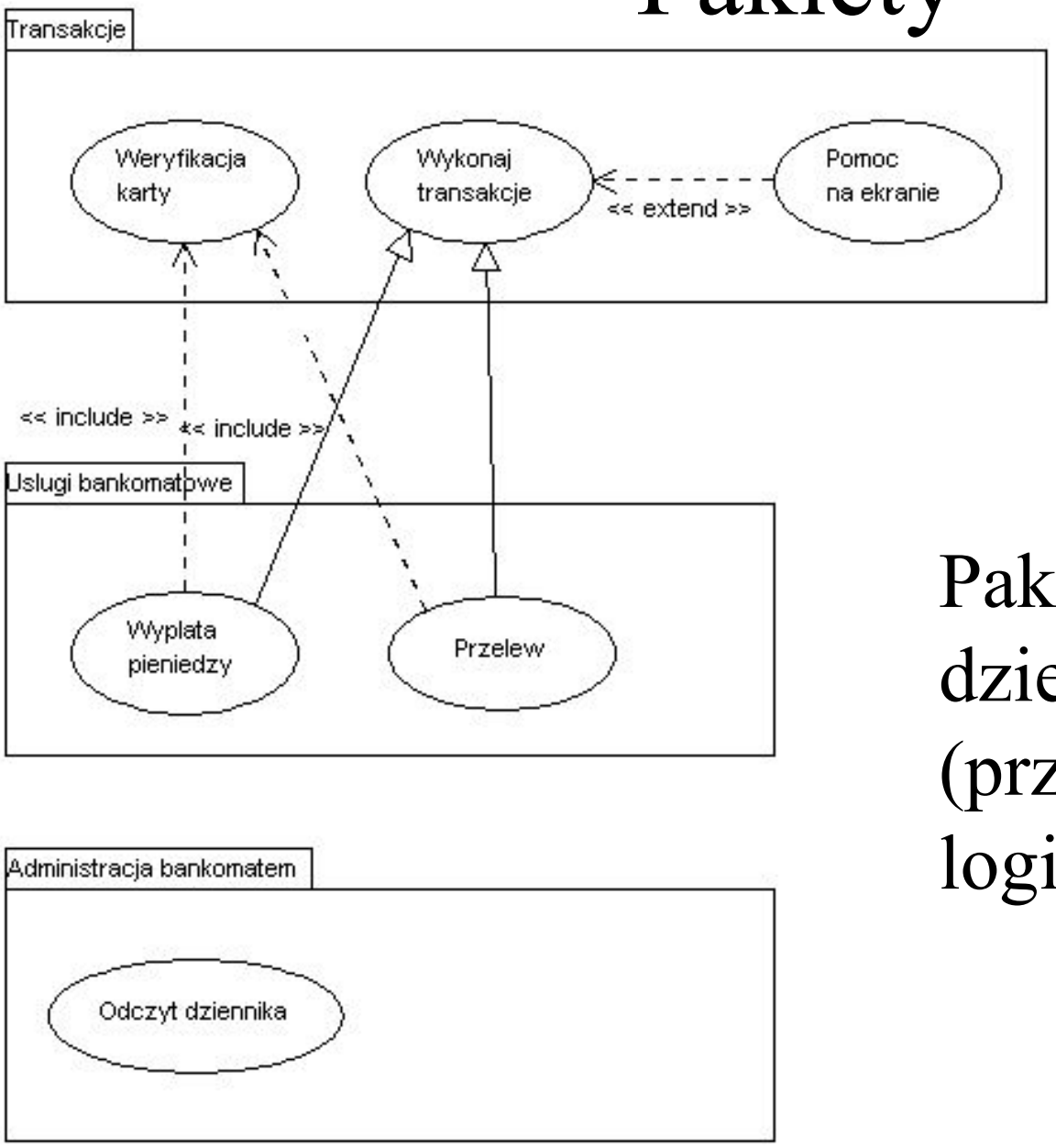
Przykład


Moduł rezerwacji
wyszukanie
Zawieranie
Rozszerzenie
«include»
czytelnik
rezerwacja
«extend»
powiadomienie
rezerwacja czasopisma
rezerwacja książki
Uogólnienie

Zasady (reguły) dla PU

- Gdy trzeba ***powtórzyć coś w kilku różnych PU***, a jednocześnie chce się uniknąć powtórzyć, należy używać ***relacji zawierania***.
- Gdy trzeba ***opisać warianty typowego postępowania przy zachowaniu opisu nieformalnego***, należy używać ***relacji uogólnienia***.
- Gdy trzeba ***opisać warianty typowego postępowania***, ale jest potrzebny ***bardziej formalny opis ze zdefiniowaniem punktów rozszerzeń*** w przypadku podstawowym, należy używać ***relacji rozszerzania***.

Pakiety



Pakiety pomagają dzielić usługi (przypadki użycia) logicznie w systemie.

Diagram przypadków użycia

dobre rady przy budowaniu diagramu

- nazwij diagram zgodnie z przeznaczeniem
- tak rozmieść przypadku użycia i aktorów żeby zminimalizować liczbę przecinających się związków
- poukładaj przypadki użycia blisko siebie, które są podobne pojęciowo
- korzystaj z notatek
- nie musisz przedstawiać wszystkich przypadków użycia na jednym diagramie

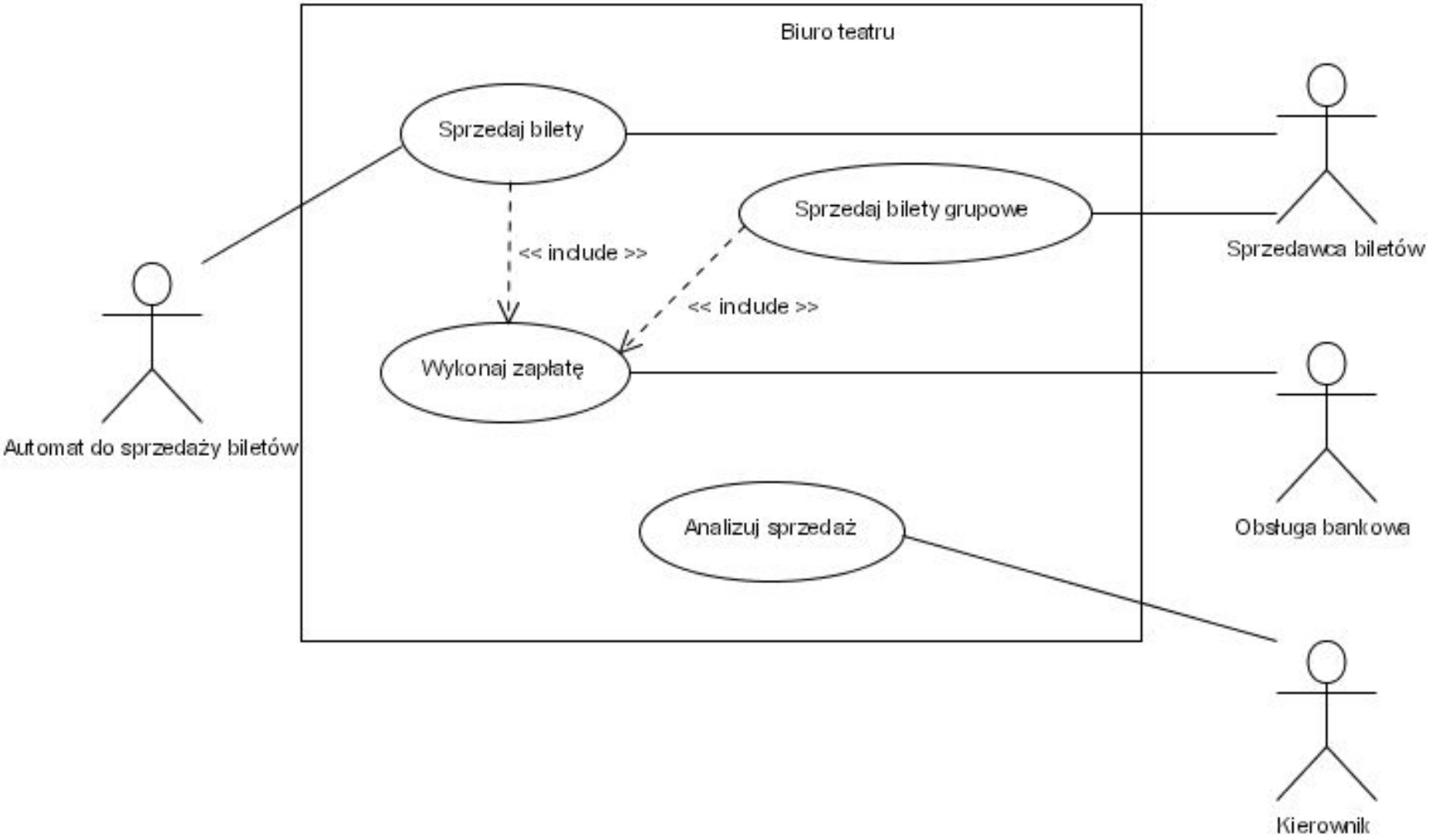
Proces tworzenia DPU

- Proces tworzenia diagramu przypadków użycia jest procesem iteracyjnym składającym się z takich etapów jak:
  - Identyfikacji aktorów
  - Opcjonalnemu opracowaniu diagramu kontekstowego
  - Identyfikacji przypadków użycia
  - Opracowaniu związków – w szczególności asocjacji
  - Wykorzystaniu wszystkich kategorii zaawansowanych do opracowania diagramów przypadków użycia
  - Udokumentowaniu przypadków użycia z wykorzystaniem szablonów

Szablon dokumentacji PU

Nazwa	**Pełna nazwa przypadku użycia**
Numer PU	**Numer identyfikacyjny przypadku użycia**
Twórca	**Dane twórcy PU (nazwisko, imię, stanowisko)**
Poziom ważności	**Określenie poziomu ważności PU z perspektywy użytkownika np. niski, średni, wysoki**
Typ przypadku	**Określenie typu PU z punktu widzenia jego złożoności oraz ważności dla zaspokojenia potrzeb użytkownika ogólny/szczegółowy; niezbędny/istotny/przeciętnie istotny**
Aktorzy	**Lista aktorów będących w związku z przypadkiem użycia**
Krótki opis	**Krótka ogólna charakterystyka przypadku użycia**
Warunki wstępne	**Charakterystyka koniecznych warunków inicjujących PU**
Warunki końcowe	**Charakterystyka stanu systemu po realizacji PU**
Główny przepływ zdarzeń	**Wypunktowana i scharakteryzowana lista przypadków zdarzeń zachodzących podczas PU; scenariusz główny**
Alternatywne przepływy zdarzeń	**Wypunktowana i scharakteryzowana lista możliwych, alternatywnych przepływów zdarzeń PU**
Specjalne wymagania	**Wypunktowana i scharakteryzowana lista dodatkowych zidentyfikowanych wymagań niefunkcjonalnych, które mogą być istotne podczas projektowania czy kodowania**
Notatki	**Lista wszelkich komentarzy dotyczących PU**

Diagram przypadków użycia



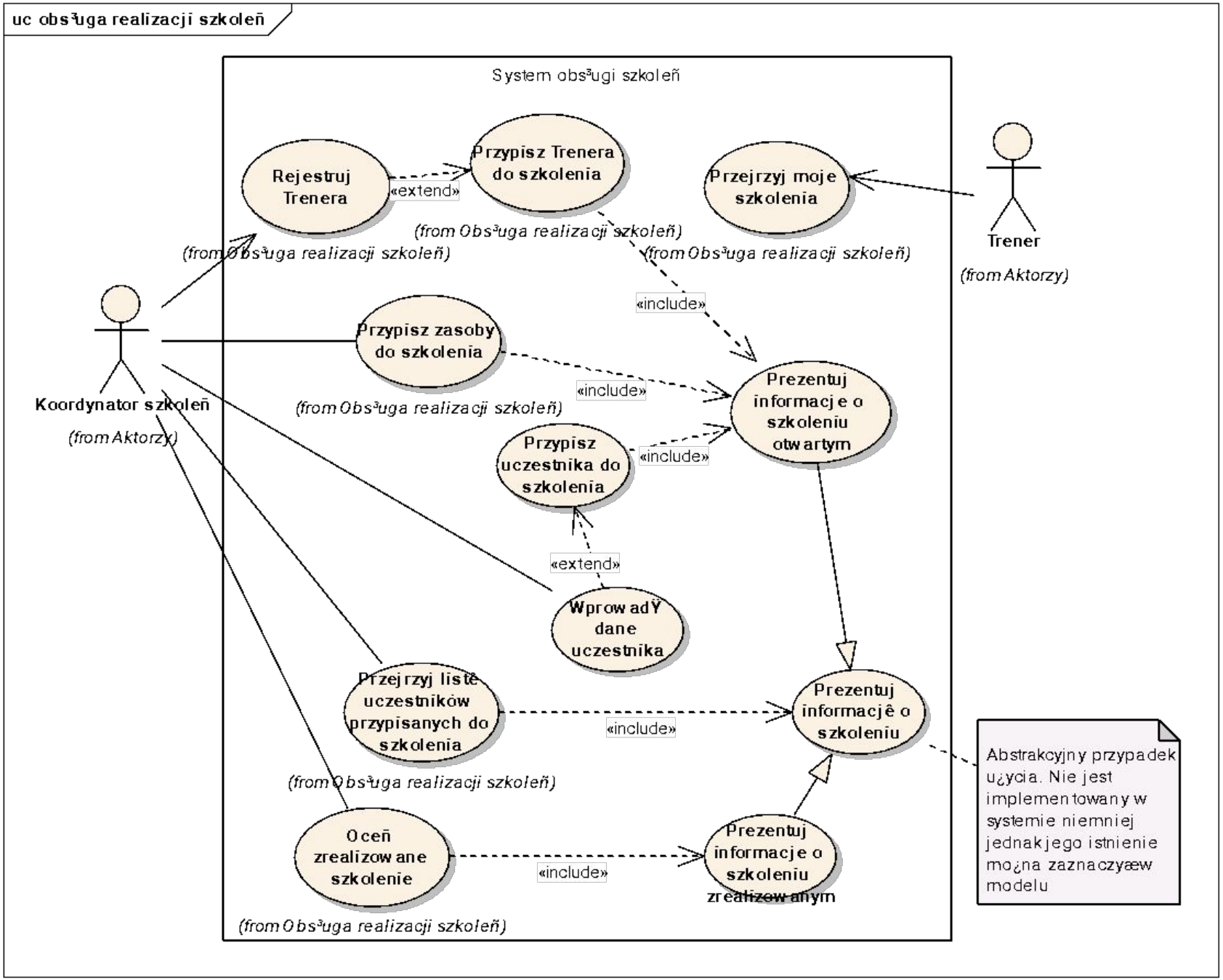

uc obs³uga realizacji szkoleñ
System obs³ugi szkoleñ
Rejestruj Trenera
Przypisz Trenera do szkolenia
«extend»
(from Obs³uga realizacji szkoleñ)
(from Obs³uga realizacji szkoleñ)
Przejrzyj moje szkolenia
(from Obs³uga realizacji szkoleñ)
Trener
(from Aktorzy)
«include»
Koordynator szkoleñ
(from Aktorzy)
Przypisz zasoby do szkolenia
(from Obs³uga realizacji szkoleñ)
«include»
Prezentuj informacje o szkoleniu otwartym
Przypisz uczestnika do szkolenia
«include»
«extend»
WprowadŸ dane uczestnika
Przejrzyj listê uczestników przypisanych do szkolenia
(from Obs³uga realizacji szkoleñ)
«include»
Prezentuj informacjê o szkoleniu
Abstrakcyjny przypadek u¿ycia. Nie jest implementowany w systemie niemniej jednak jego istnienie mo¿na zaznaczyæ w modelu
Oceñ zrealizowane szkolenie
(from Obs³uga realizacji szkoleñ)
«include»
Prezentuj informacje o szkoleniu zrealizowanym

Diagram klas

- Jest to najczęściej spotykany diagram w modelach obiektowych.
- Diagram przedstawia statyczną stronę projektowanego systemu.
- Na diagramie występują
  - klasy
  - interfejsy
  - kooperacje
  - związki między nimi.

Diagram klas (*Class diagram*)

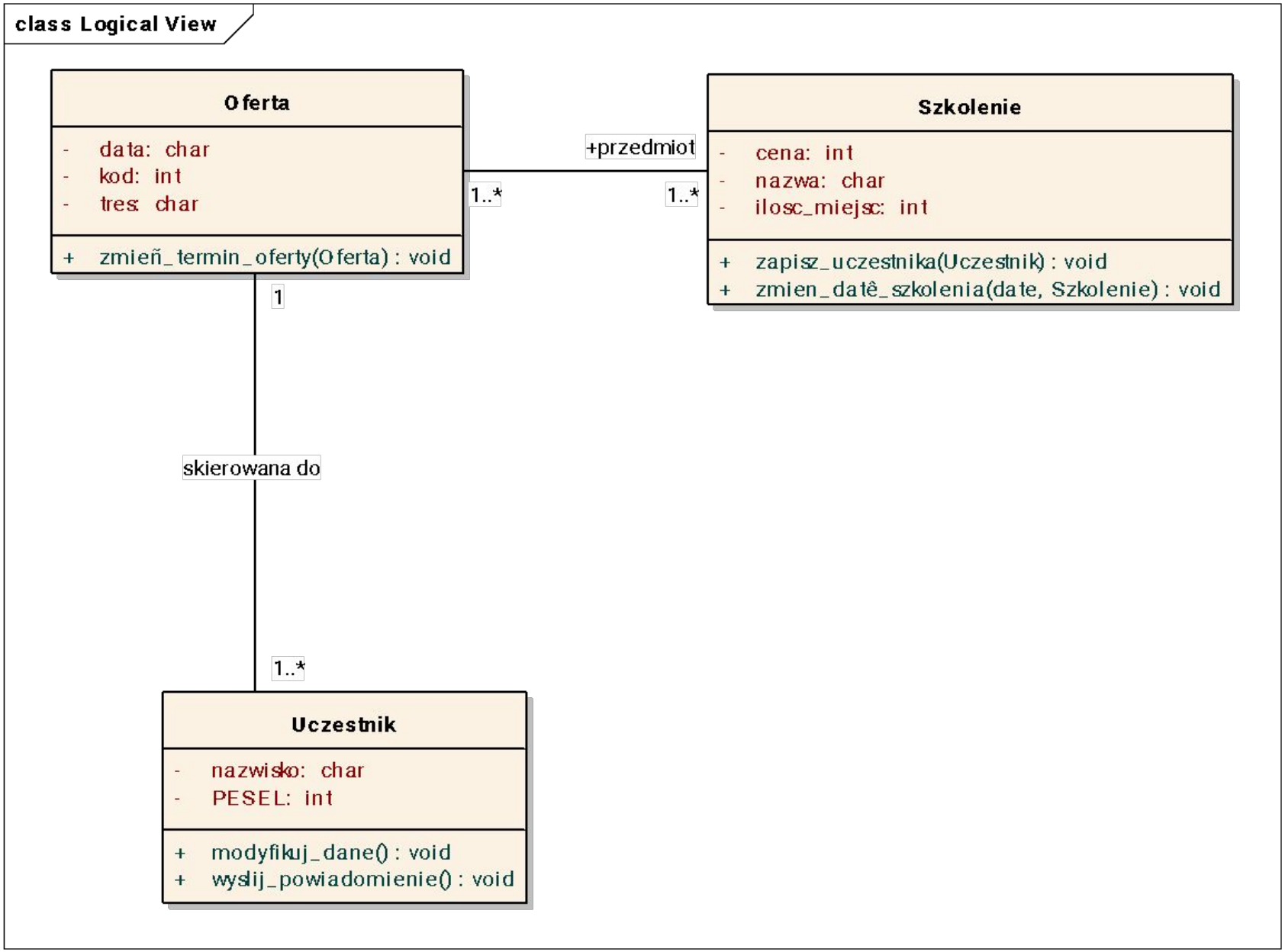


Diagram klas

Kluczowymi elementami są:

- klasy (*class*)
- związki (*association*)
- interfejsy (*interface*)

Dodatkowo diagram może zwierać:

- notatki (*note*)
- ograniczenia (*constraints*)
- pakiety (*packages*)

Klasy – wzorce obiektów

- Klasa
  - opis grupy obiektów o jednolitym zbiorze atrybutów i sposobie zachowania
  - zawiera opis tworzenia obiektów klasy
- Klasę można nazwać „fabryką obiektów”
- Każda klasa posiada „wzorzec” (plany) dla tworzenia obiektów tej klasy.
- Każdy nowo tworzony obiekt klasy posiada osobną tożsamość, a także może mieć różne wartości atrybutów.

Właściciel

wniesienie o rejestrację

własność

Rejestrator

zarejestrowanie

Pojazd

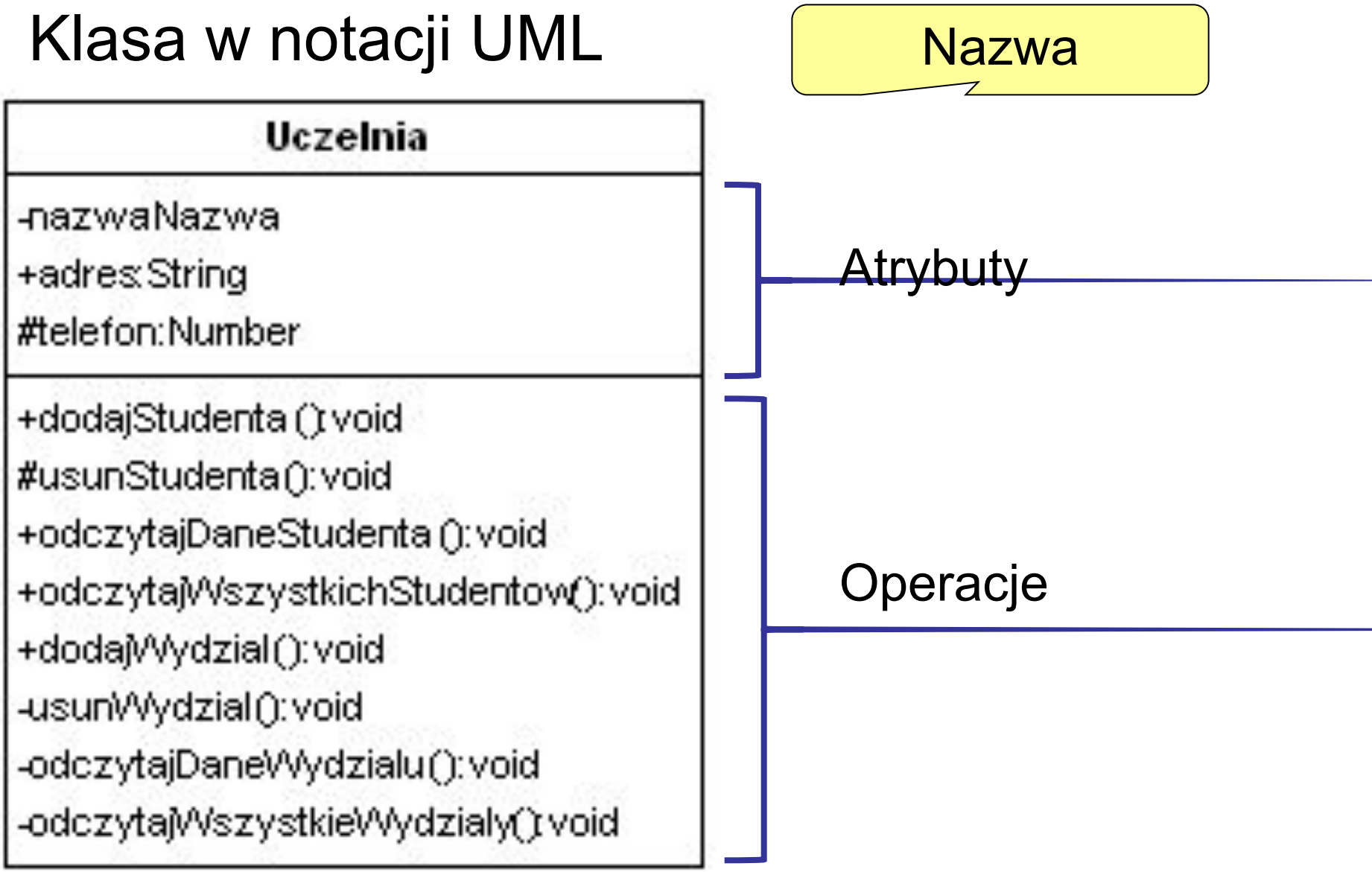
wykonanie czynności urzędowej

Rejestracja

dokonanie rejestracji

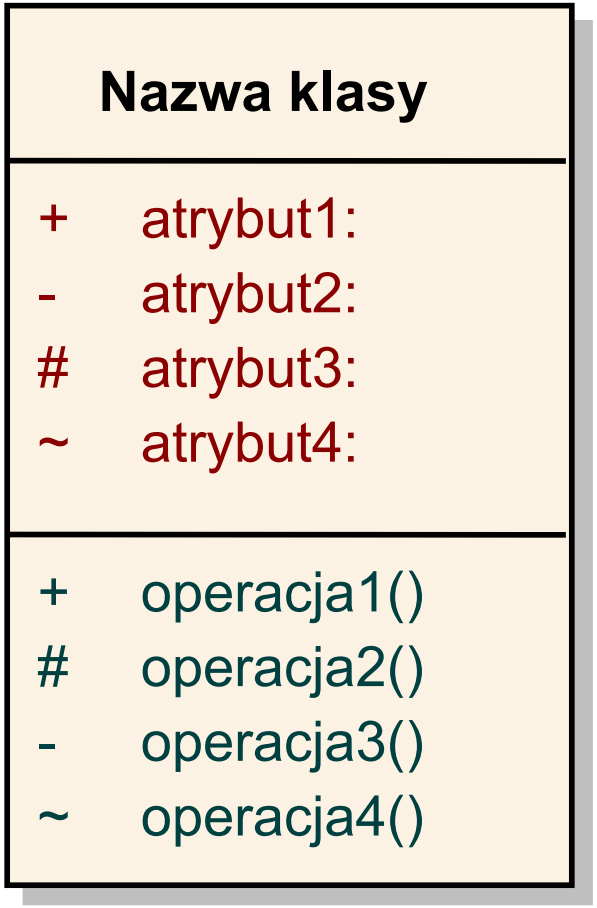
Diagram klas

Klasa w notacji UML



Widoczność składowych klasy

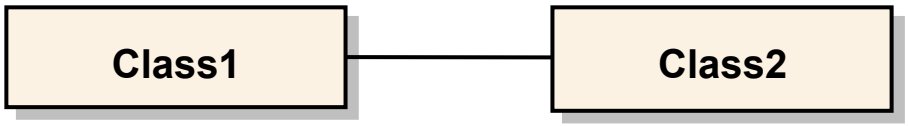
publiczny	+	Obiekty wszystkich klas mają dostęp do atrybutu lub operacji
prywatny	-	Dostęp jedynie w obrębie danej klasy
chroniony	#	Dostęp jedynie dla klas dziedziczących z danej klasy
pakietowy	~	Dostęp tylko dla składowych pakietu, do którego klasa należy



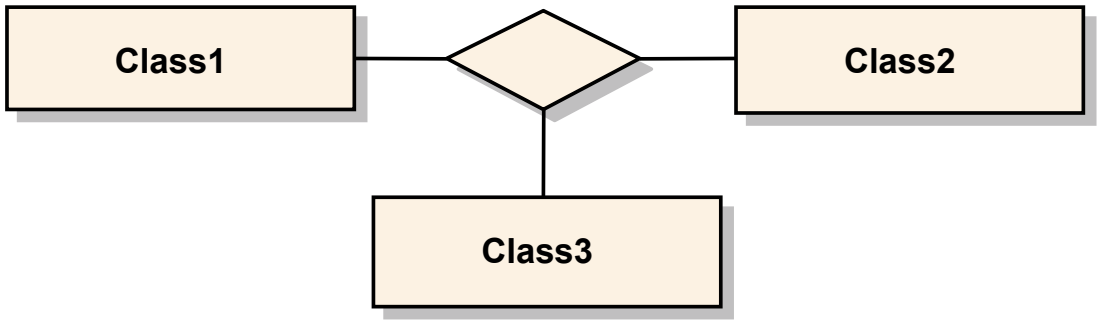
Asocjacja

Asocjacja opisuje związek strukturalny miedzy elementami (klasami). Wskazuje, że obiekty jednego elementu są połączone z obiektami drugiego. Wyróżnia się dwa rodzaje asocjacji:

- binarne

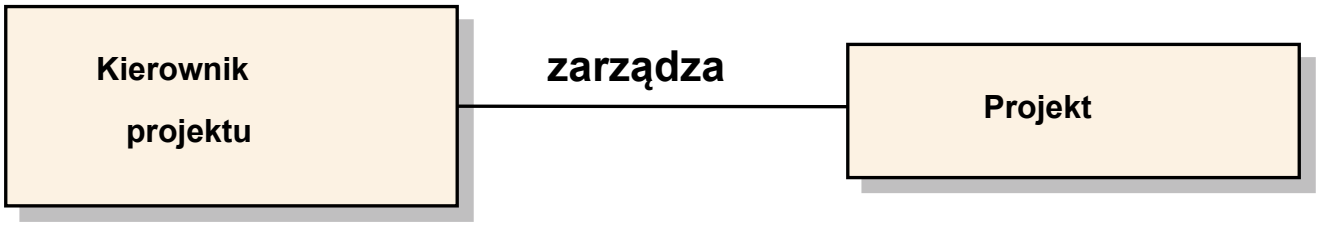


- n – arne

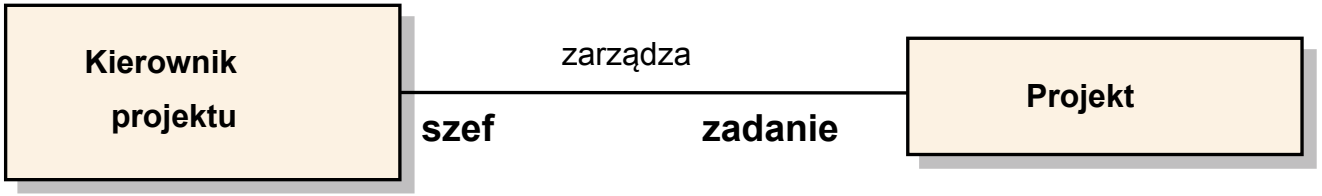


Asocjacje – cechy

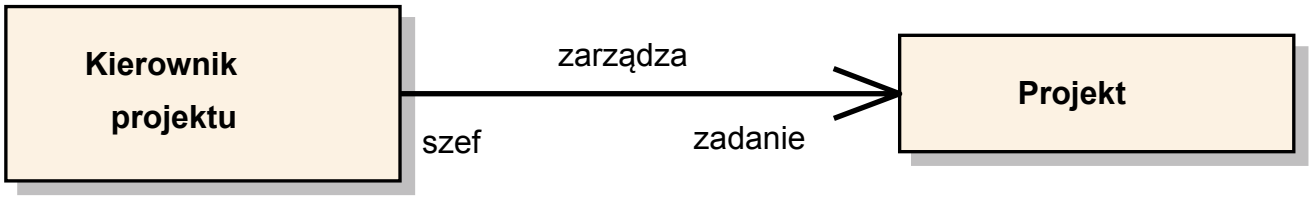
- **nazwa**



- **rola**



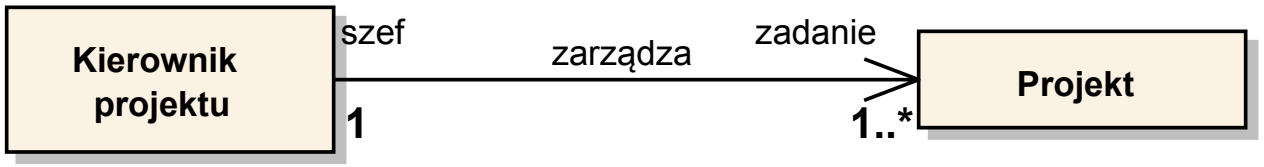
- **nawigacja**



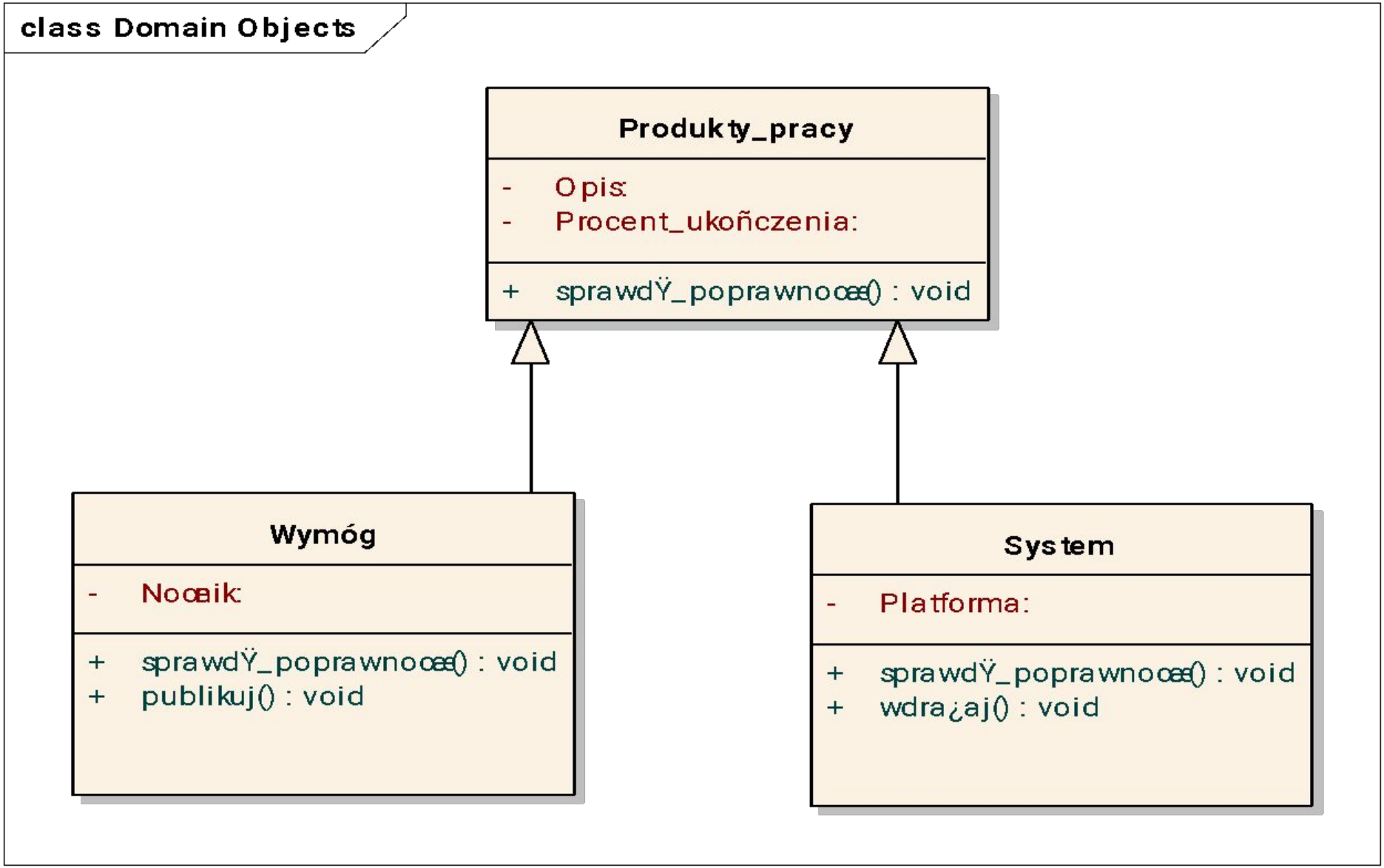
Asocjacje – cechy

- **liczebność - podając liczebność na pewnym końcu powiązania (przy pewnej klasie) wskazujemy ile obiektów tej klasy musi być połączonych z każdym obiektem klasy znajdującej się na przeciwnym końcu powiązania.**

1	dokładnie jeden	n	dokładnie n (n>1)
1..*	jeden lub wiele	1..n	od 1 do n
0..1	zero lub jeden	0..n	od 0 do n
*	wiele	n..m	od n do m (n,m>1)
0..*	zero lub wiele	n,m..o	liczebność złożona
n..*	więcej niż n		



Uogólnienie - generalizacja

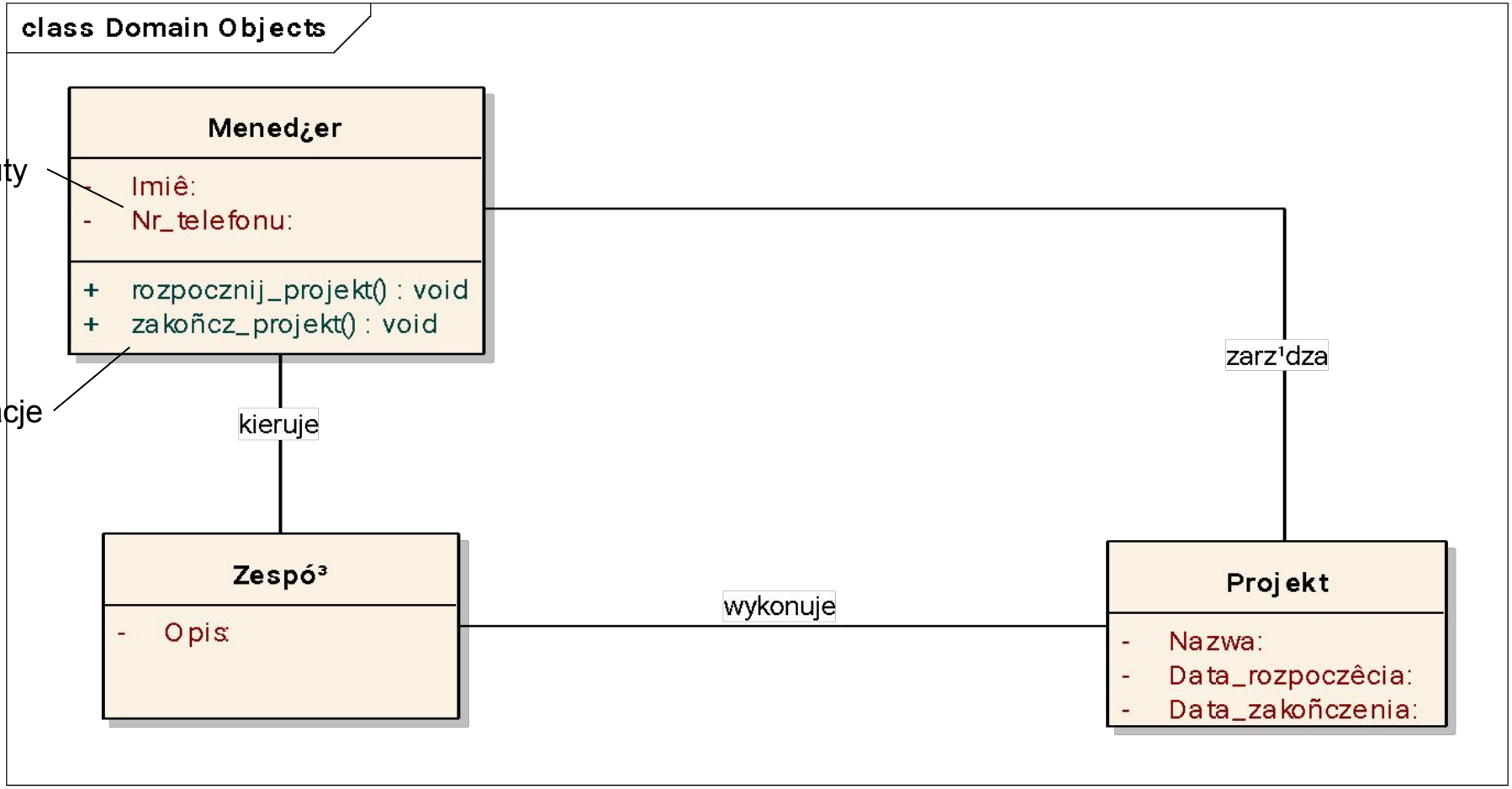


Konceptualny diagram klas zawiera wyłącznie podstawowe elementy, cechując się przystępnością nazewnictwa klas, atrybutów i operacji.

Implementacyjny diagram klas jest stopniowo wzbogacany o elementy opisu niezbędne dla prawidłowej specyfikacji modelu, takie jak typy danych, zobowiązania, widoczność, statyczność, klasy asocjacyjne, kwalifikacje, uogólnienia, zależności czy też realizacje.

Etapy tworzenia diagramu klas

1) zidentyfikowanie i nazwanie klas;
2) opcjonalnie określenie zobowiązań klas;
3) połączenie poszczególnych klas z wykorzystaniem związków asocjacji;
4) zidentyfikowanie oraz nazwanie atrybutów i operacji;
5) wyspecyfikowanie asocjacji z użyciem wszystkich jej cech (nazwy, ról, nawigacji, liczebności, agregacji, kwalifikacji);
6) opracowanie innych rodzajów związków, tj. uogólnień, zależności i realizacji;
7) pełne, precyzyjne wyspecyfikowanie atrybutów i operacji;
8) opcjonalnie opracowanie diagramów obiektów.



Na diagramie przedstawione są następujące informacje:

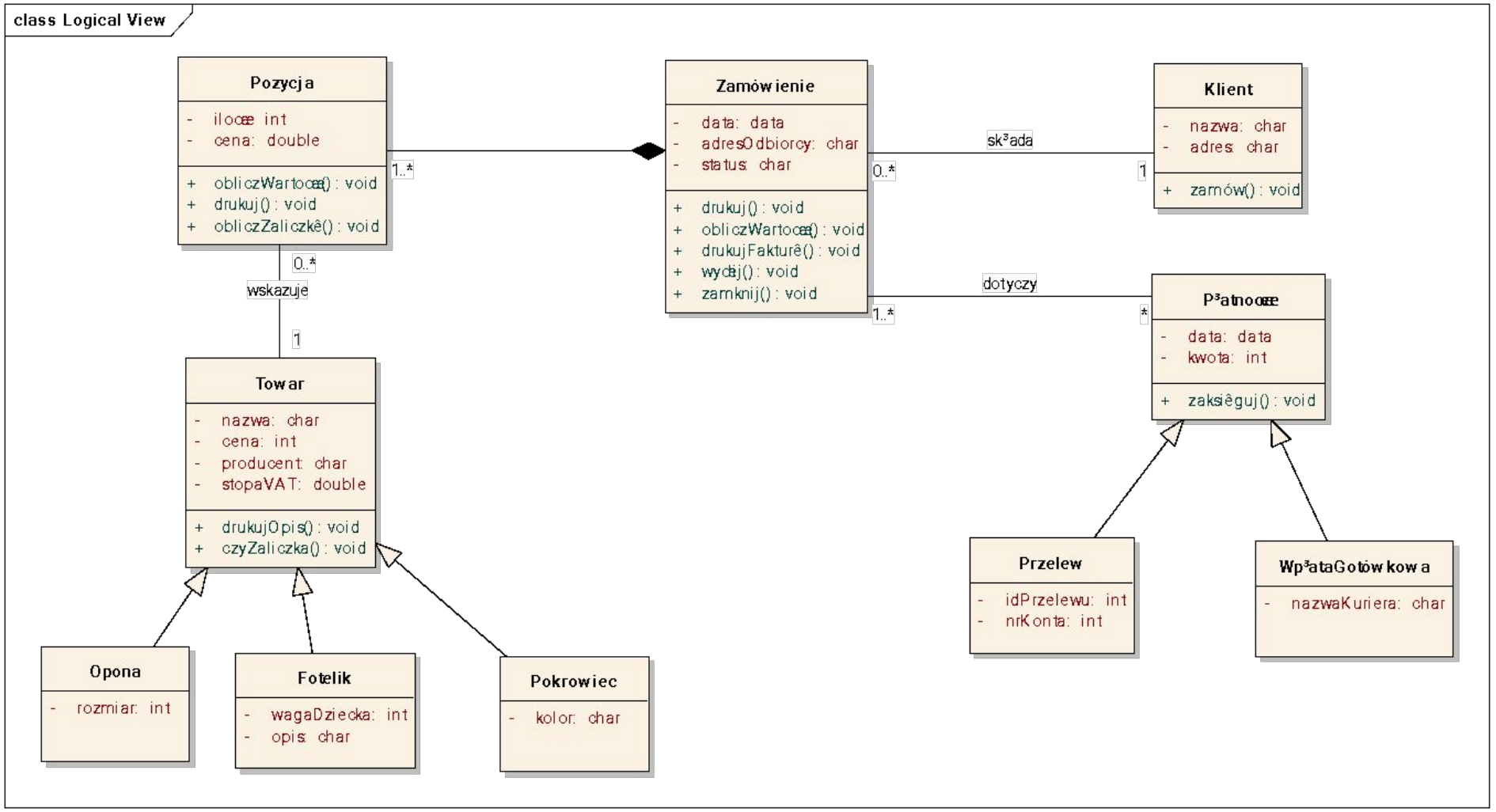
-Menedżer przewodzi zespołowi, który wykonuje projekt

-Każdy menedżer ma imię i numer telefonu, i może zainicjować lub zakończyć (przerwać) projekt

- Każdy projekt ma nazwę, datę rozpoczęcia i datę zakończenia

-Każdy zespół ma opis, i tylko on nas interesuje

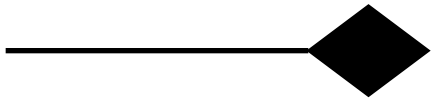
Diagram klas systemu sprzedaży wysyłkowej



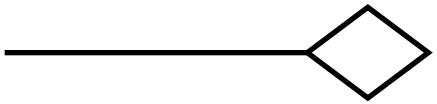
Modelem obrazującym strukturę i wzajemne powiązania obiektów występujących w systemie jest diagram klas.

Asocjacje – cechy

- **agregacja** – opisuje związek całość-część pomiędzy klasami. Wyróżnia się dwa rodzaje agregacji:
  - **agregacja całkowita** – obiekty nie mogą samodzielnie funkcjonować, usunięcie *agregatu* powoduje likwidację *segmentów*,



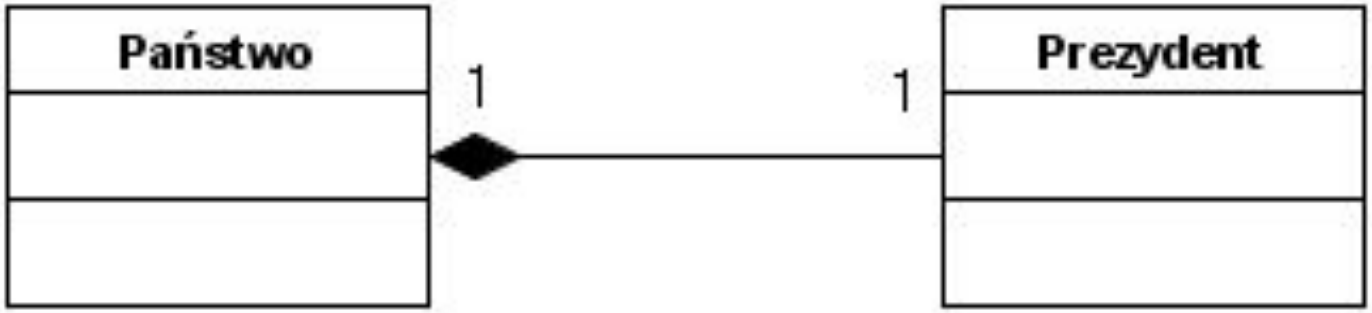
  - **agregacja częściowa** – usunięcie *agregatu* nie powoduje usunięcia jego *części*, obiekty współdzielone mogą funkcjonować samodzielnie



Związki

Kompozycja, zwana również *agregacją całkowitą*, wprowadza dodatkowo zależność czasu życia klas oraz wyłączną własność.

Części o nieustalonej liczebności mogą powstawać po utworzeniu całości, ale potem żyją i umierają razem z nią.



Związki

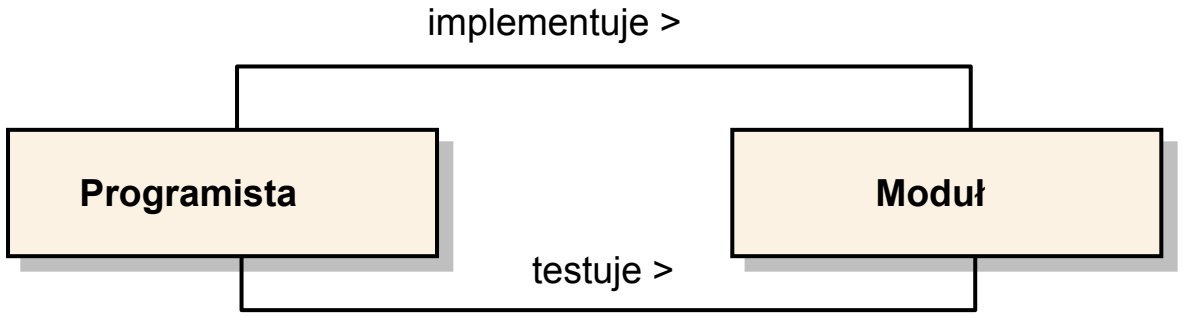
Dodatkową funkcjonalnością powiązań jest **agregacja (*aggregation*)** i **kompozycja (*composition*)**.

Zwykłe powiązanie sprawia, że związane elementy są sobie równorzędne.

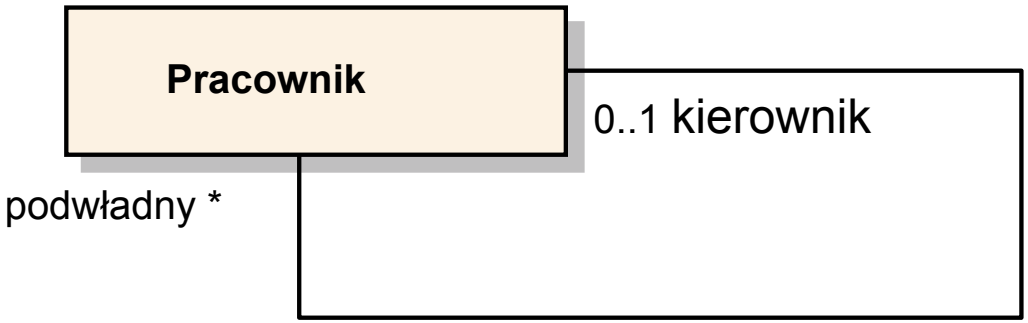
W celu zaprezentowania związku „część-całość” należy użyć agregacji.

Asocjacja zwrotna i wielokrotna

- **asocjacja wielokrotna** – połączenie klas kilkoma asocjacjami w zależności od pełnionych ról



- **asocjacja zwrotna** – powiązanie danej klasy z samą sobą



Agregacja jest specjalną formą asocjacji, której podstawową intencją jest odwzorowanie związków część-całość

1. Czy właściwe jest użycie frazy “jest częścią”?
2. Czy operacje na całości automatycznie są propagowane na jej części?
3. Czy jakiś atrybut całości jest automatycznie propagowny do jej części?
4. Czy istnieje wewnętrzna asymetria w asocjacji, kiedy jakaś klasa jest podporządkowana innej klasie?

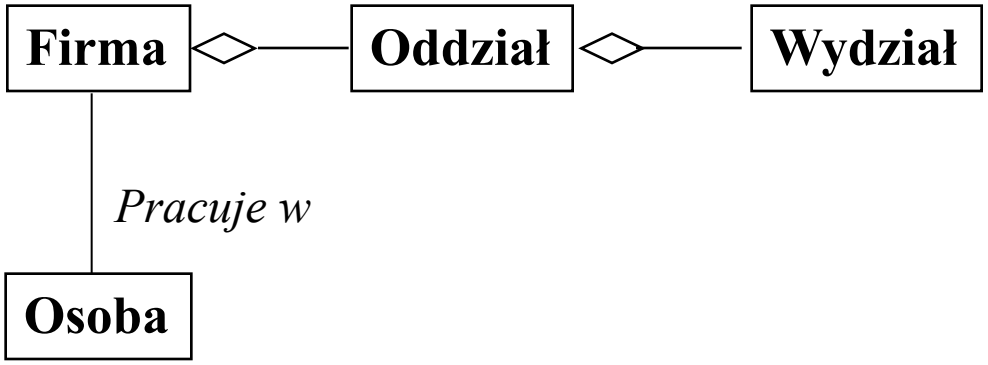


Diagram klas

Klasa jest opisem zbioru obiektów, które mają takie same:

- atrybuty
- operacje
- związki
- znaczenia

Diagram klas

- Diagramy klas służą do *obrazowania statycznych aspektów projektowanych* systemów jako:
  - Projekt struktury logicznej baz danych
  - Projekt składników systemu stanowiący podstawę do stworzenia informatycznego systemu (kodu)
- Na podstawie diagramów klas bardzo prosto *można generować kod* (SQL, Java, C++ itd.)
- Diagramy klas są *wykorzystywane przez analityków na etapie opracowywania koncepcji systemu jak i przez projektantów na etapie projektowania implementacji*

Definicja obiektu

- Obiekt jest to struktura danych stanowiąca w implementacji komputerowej *odwzorowanie wyróżnialnego w analizowanym fragmencie dziedziny problemowej bytu*, który posiada dobrze określone granice i własności. Z koncepcją obiektu ściśle związane są takie pojęcia jak:
  - Tożsamość obiektu, która odróżnia go od innych obiektów i jest niezależna od wartości jego atrybutów, od powiązań z innymi obiektami, od lokalizacji bytu w świecie rzeczywistym oraz od lokalizacji obiektu w przestrzeni adresowej komputera.
  - Stan obiektu, który jest określony przez aktualne wartości jego atrybutów i powiązań z innymi obiektami. Stan obiektu może zmieniać się w czasie.
  - Zachowanie obiektu przypisane do obiektu, tj. zestaw operacji, które można na nim wykonać.
  - Typ obiektu, tj. wyrażenie językowe, które przez specyfikację budowy obiektu ogranicza kontekst, w którym do tego obiektu można się odwoływać.

Diagram obiektów (*Object diagram*)

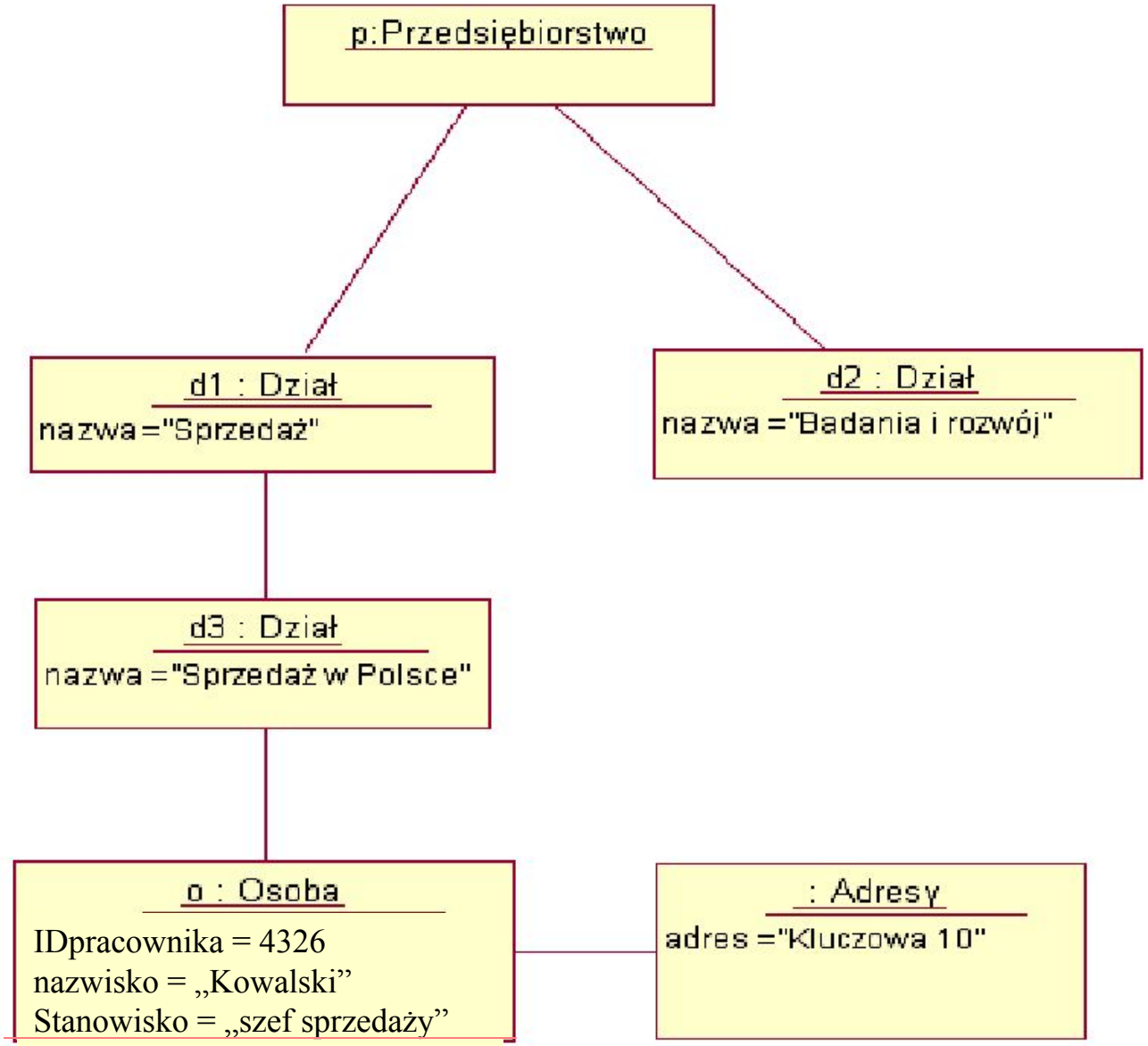


Diagram obiektów

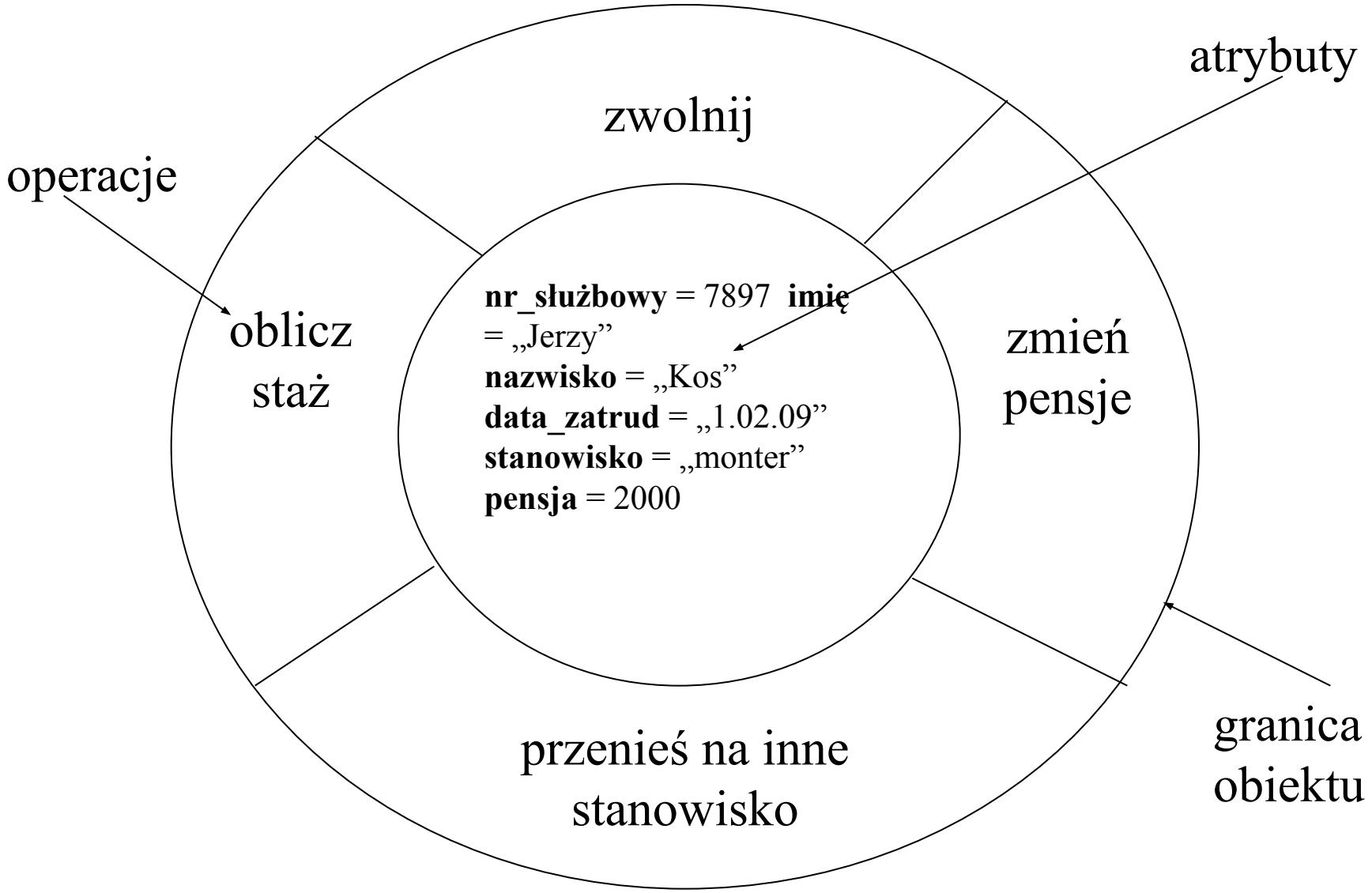
- Diagram obiektów *ukazuje elementy i związki z diagramu klas w ustalonej chwili*.

- Diagram obiektów jest grafem
  złożonym z wierzchołków i krawędzi.

- Diagram obiektów wyraża
  zrzut systemu w określonym czasie.

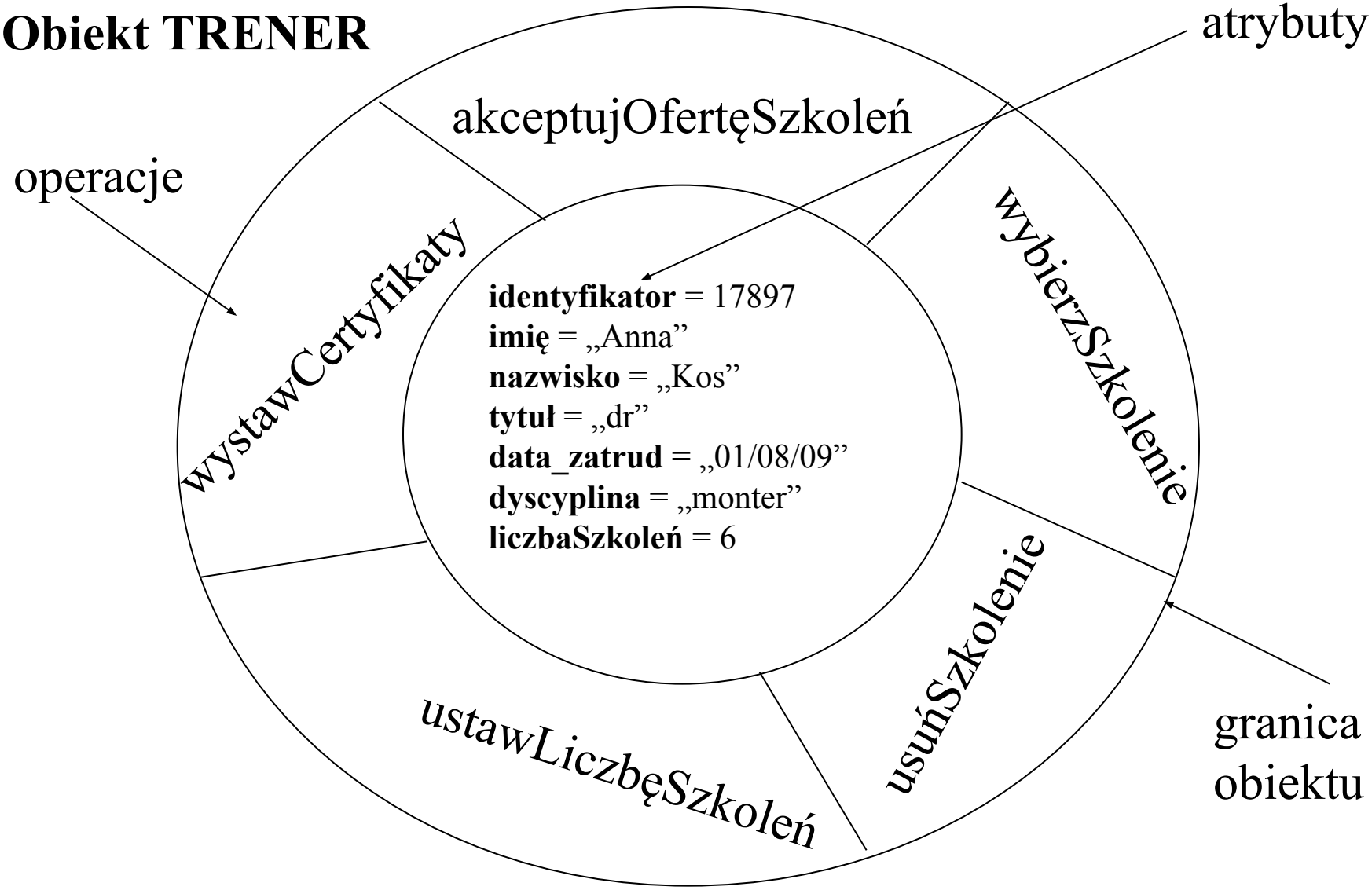
Cechy obiektów

- Każdy obiekt posiada trzy podstawowe cechy:
  - Tożsamość – unikalny „uchwytny” (metkę), po którym obiekt odróżniany jest od innych.
  - Stan – zawartość komórek przechowujących dane opisujące obiekt.
  - Sposób zachowania – zbiór akcji, które potrafi wykonywać obiekt.
- Obiektami mogą być: *osoby, przedmioty, miejsca, jednostki organizacyjne, wydarzenia, ekrany, raporty, pojęcia abstrakcyjne*.



Przykładowy obiekt *pracownik,* który opisuje pewnego pracownika z dziedziny problemowej. Obiekt ten posiada atrybuty, m.in. *imię*, *nazwisko*; na obiekcie można wykonać operacje, m.in. *zmień pensję*

Obiekt TRENER



Obiekt to byt z dobrze zdefiniowaną granicą (co nim jest, a co nie), identyfikowalny, który ukrywa swój stan (reprezentowany przez wartości atrybutów i zależności) i zachowania (reprezentowane przez operacje).

Stan obiektów

- Każdy obiekt opisujemy jako zbiór jego cech (atrybutów).
- Aktualne wartości atrybutów obiektu stanowią jego stan.
- Stan obiektu może się zmieniać przez cały czas jego życia.
- Modelując obiekty, do ich opisu wybieramy tylko zestawy atrybutów występujących w danej dziedzinie problemu.

Zachowanie obiektów

- Obiekty nie tylko przechowują swój stan, ale również „wiedzą” jak się należy zachować.
- Obiekty mogą wykonywać dla innych obiektów określone usługi.
- Każdy obiekt może mieć określoną listę usług, które potrafi wykonać.
- Obiekty możemy porównać do „czarnych skrzynek” z przyciskami; naciśnięcie przycisku powoduje wykonanie określonych operacji i (lub) zwrócenie wyniku.

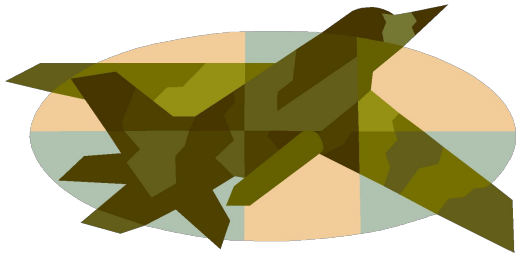


Diagram obiektów

Zawartość diagramu:

- obiekty,
- związki.

Na diagramie mogą się również znaleźć:

- pakiety,
- podsystemy,
- notatki.

Diagram obiektów

- Diagram obiektów, jako *technika modelowania struktury systemu*, przedstawia obiekty (elementy modelu), związki między nimi (relacje: asocjacje, agregacje, kompozycje, generalizacje) oraz ograniczenia.
- Diagram obiektów może być traktowany jako zrzut systemu, na dowolnym poziomie abstrakcji w danej chwili.
- Diagram obiektów wskazuje konkretne egzemplarze klas oraz interakcje, jakie zachodzą pomiędzy nimi w ustalonej chwili.

Stan obiektu

Graficzna reprezentacja obiektu składa się z:

✔nazwy – tekst podkreślony

nazwa : typObiektu np.: k : Klient

: typObiektu np.: : SterownikODBC

nazwa np.: KlientKorporacyjny

nazwa : np.: agent :

✔atrybutów obiektu

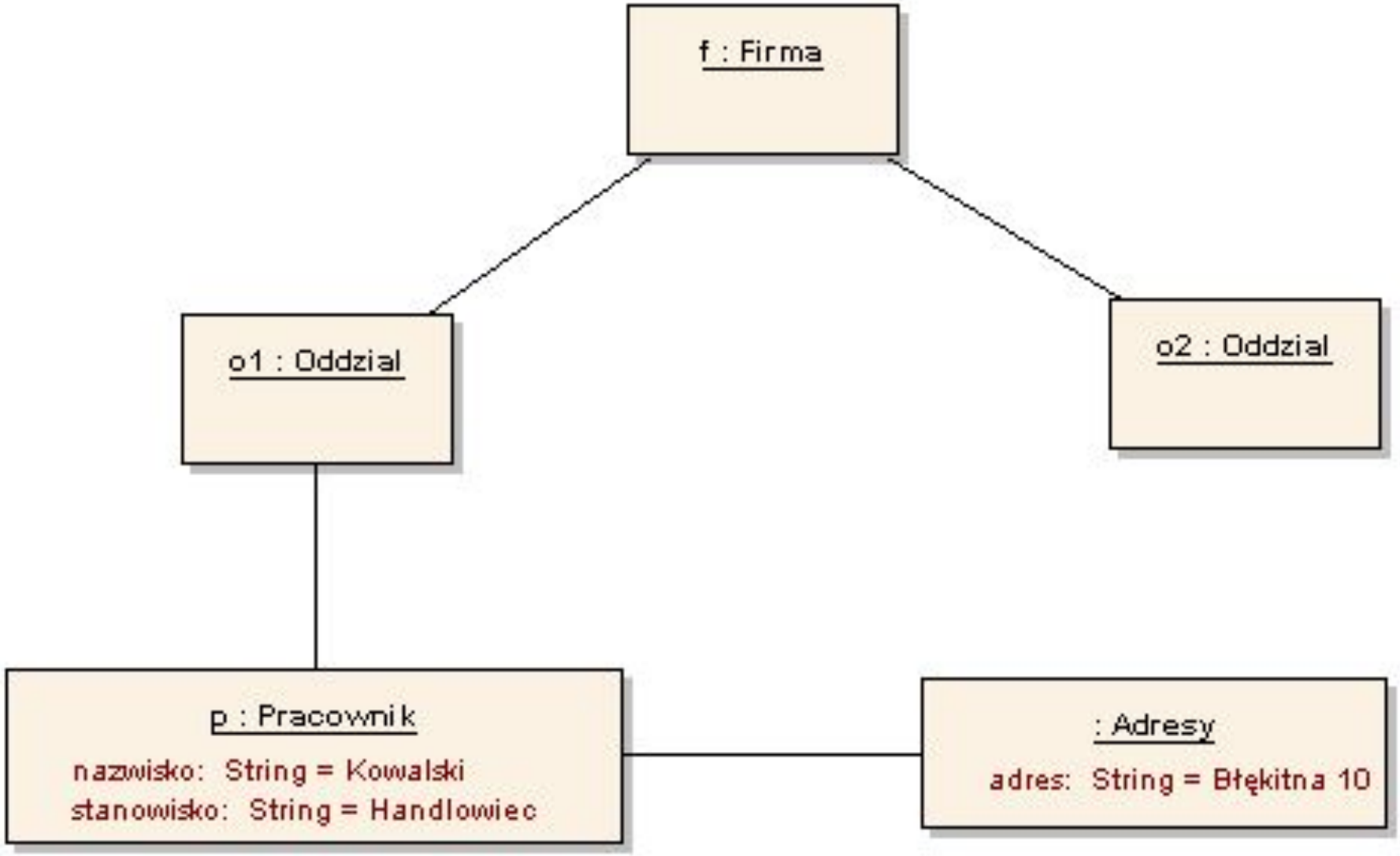
atrybut [: typ] = wartość

np.: index : int = 1001

ulica = „Poziomkowa”

Diagram obiektów

Przykładowy diagram obiektów



Struktura wydziału – diagram obiektów

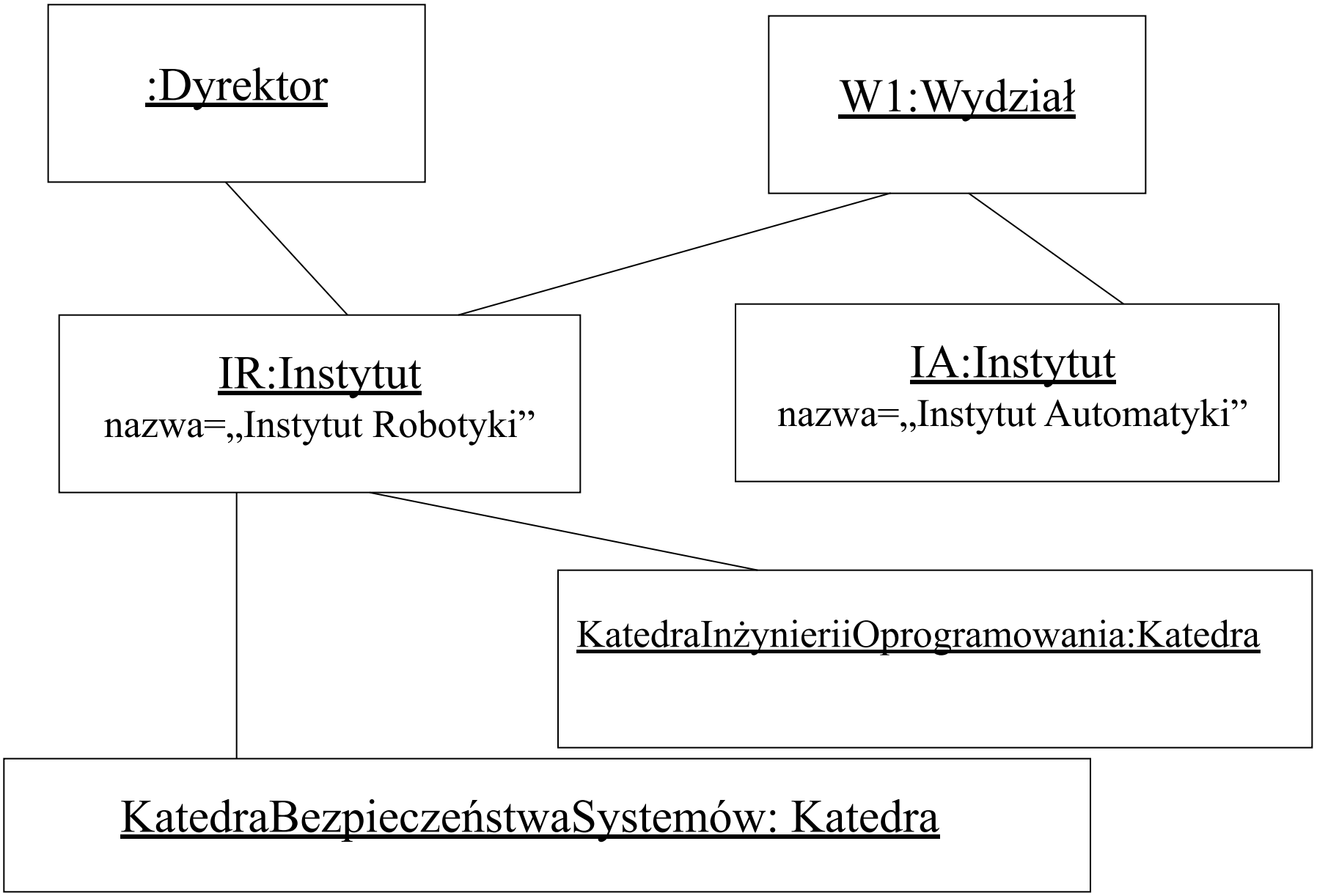


Diagram aktywności (czynności)

- ***Diagram czynności*** (*activity diagram*) służy do modelowania dynamicznych aspektów systemu.
- Diagram czynności przedstawia sekwencyjne lub współbieżne kroki procesu obliczeniowego.
- Diagram czynności jest pewną mutacją diagramu stanów.

Diagram aktywności (czynności)

- ***Diagramy czynności*** (*activity diagram*) służą do *modelowania przepływów operacji wykonywanych w celu realizacji zadań zlecanych systemowi przez jego aktorów*.
- Diagramy czynności łączą idee pochodzące z trzech źródeł: diagramów zdarzeń J. Odella, technik modelowania stanów oraz sieci Petriego.

Diagram czynności a diagram stanów

- ***Diagram czynności*** (aktywności) skupia się na *opisaniu jakiegoś procesu, w którym uczestniczy wiele obiektów*.
- ***Diagram stanów*** pokazuje jakie są możliwe *stany konkretnego obiektu*.
- ***Diagram aktywności*** jest dobrym narzędziem, gdy chcemy przedstawić *odpowiedzialność obiektów w ramach jakiegoś procesu*.

Graf aktywności

- Diagram aktywności jest grafem skierowanym, którego *wierzchołki stanowią aktywności odpowiadające operacjom wyróżnianym w trakcie przetwarzania*, a *łuki opisują przejścia pomiędzy aktywnościami*.
- Można powiedzieć, że graf aktywności to maszyna stanów, której podstawowym zadaniem nie jest przedstawianie stanów obiektu, jak ma to miejsce w przypadku diagramów stanów, ale modelowanie przepływów operacji.
- Pojedynczy stan grafu aktywności może być interpretowany:
  - **Z perspektywy pojęciowej** jako zadanie do wykonania przez człowieka i przez komputer.
  - **Z perspektywy projektowej** jako grupa metod, pojedyncza metoda czy też nawet fragment metody.

Diagram czynności

Diagram czynności składa się z:

- początek (*initial*)
- koniec (*final*)
- akcji i czynności (*activity*)
- przejść (*flow*)
- rozwidlenie/złączenie (*fork/join*)
- punkt synchronizacji (*synch*)
- rozgałęzienie decyzyjne (*decision*)
- wysłanie (*send*)/odebranie (*receive*)

Początek i koniec

Początek jest rozpoczęciem diagramu czynności. Od niego rozpoczyna się wędrówka zdarzeń i stanów.

Initial

Koniec jest zakończeniem działań systemu w diagramie czynności.

Final

Akcja

Stany akcji to niepodzielne zdarzenia jak:

- obliczenie
- wywołanie operacji obiektu
- wysłanie sygnału do obiektu
- utworzenie/zniszczenie obiektu

Stany akcji nie mogą być dekomponowane.

Czynność

Czynności są bardzo podobne do akcji. Różnica polega na tym, że stany czynności mogą być dekomponowane.

Czynność może mieć dodatkowo akcje wejściowe i akcje wyjściowe.

Czynność - akcja

- ***Czynności*** na diagramie mogą charakteryzować się złożoną, rozbudowaną funkcjonalnością.
- Czynność to określone zachowanie złożone z logicznie uporządkowanych ciągów podczynności, akcji oraz obiektów w celu wykonania pewnego procesu.
- ***Akcja*** to elementarna jednostka specyfikacji zachowania, która reprezentuje transformację lub przetwarzanie w modelowanym systemie.

Dekompozycja czynności

Czynności można dekomponować stosując następującą regułę:

- czynności
- podczynności
- akcje

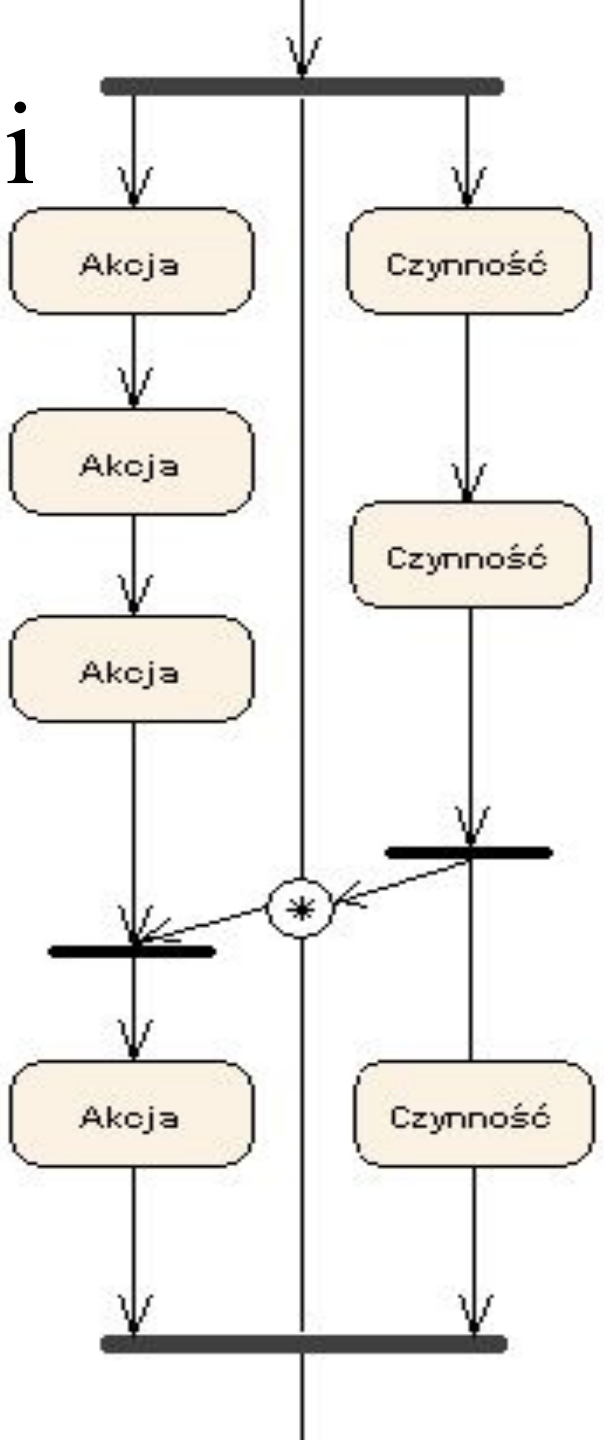
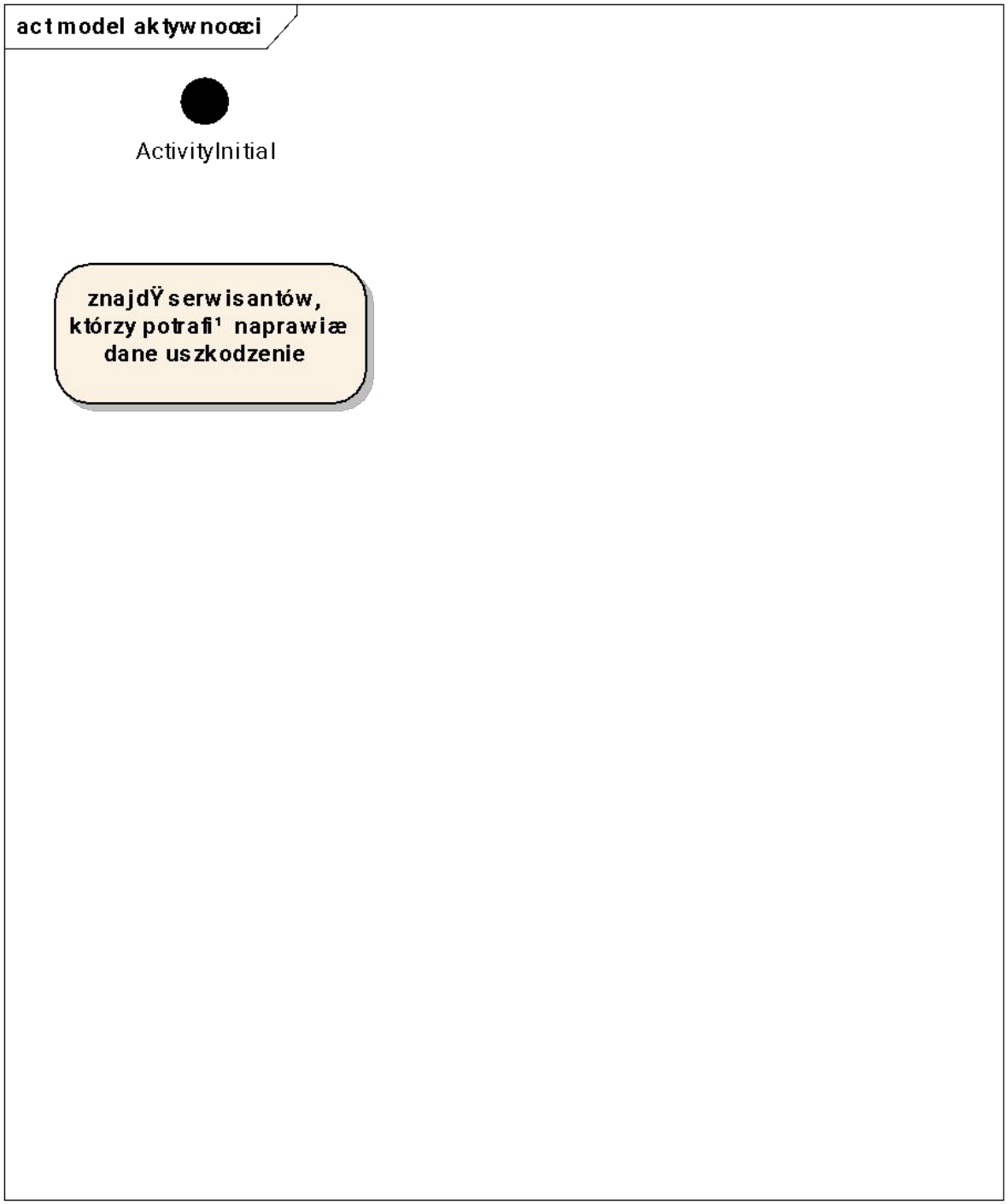
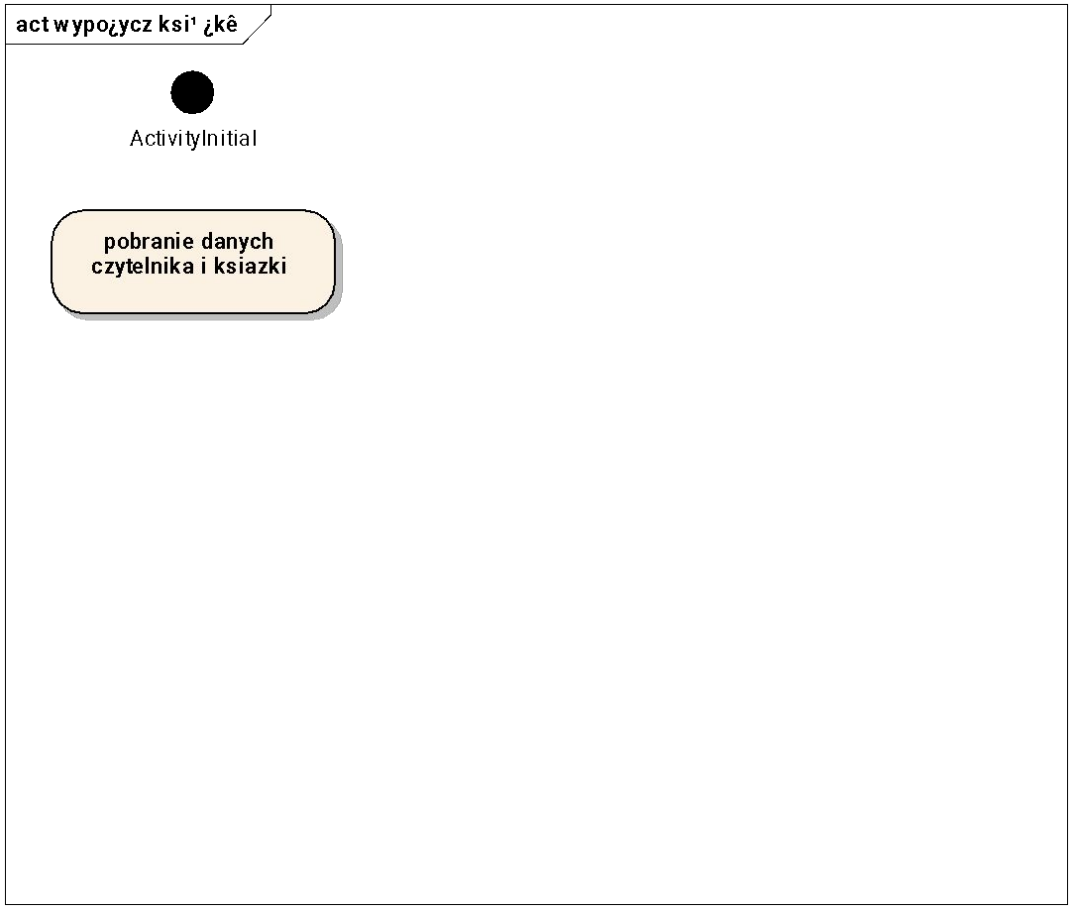


Diagram czynności

- Diagram czynności służy do *obrazowania dynamicznych aspektów systemu*.
- Diagram czynności można *kojarzyć z przypadkami użycia i z kooperacjami*.
- Istotą diagramu są *czynności i akcje oraz przepływ sterowania między nimi*.
- Na diagramie czynności można ukazać części systemu, które odpowiedzialne są za różne zadania.


act model aktywnoœci
ActivityInitial
znajdŸ serwisantów, którzy potrafi¹ naprawiæ dane uszkodzenie

Przykład - biblioteka



Biblioteka posiada książki i czasopisma. Może być kilka egzemplarzy tej samej książki. Tylko personel może wypożyczać czasopisma. Członek biblioteki może mieć jednocześnie wypożyczonych sześć pozycji, podczas gdy osoba pracująca w bibliotece może mieć ich wypożyczonych dwanaście. System ma rejestrować wypożyczenia i zwroty oraz pilnować, by przestrzegano wymienionych wyżej reguł (ograniczeń).

PU – przydziel prawnika do sprawy

Przypadek użycia rozpoczyna aktor Szef kancelarii System pyta o dane prawnika i dane sprawy Szef kancelarii wprowadza potrzebne dane System sprawdza, czy prawnik nie jest zleceniodawcą sprawy Jeżeli prawnik nie jest zleceniodawcą sprawy, system sprawdza czy prawnik nie zajmuje się aktualnie inną sprawą Jeżeli prawnik w danym momencie jest już przydzielony do innej sprawy, system informuje o zajętości prawnika i kończy PU
Jeżeli prawnik jest aktualnie wolny, system rejestruje przydzielenie prawnika do sprawy i kończy PU

Uwarunkowania

- Diagramy aktywności obrazują przetwarzanie na wysokim poziomie abstrakcji, dlatego są używane jako punkt startowy dla procesu modelowania zachowań, podczas którego każda aktywność jest rozpisywana na szereg operacji.
- Diagramy aktywności znajdują zastosowanie przede wszystkim w następujących obszarach:
  - Do analizowania PU – gdy bardziej interesują nas operacje niezbędne do realizacji danego przypadku czy też wzajemne zależności między tymi operacjami
  - Do zrozumienia interakcji zachodzących między PU
  - Do modelowania przetwarzania wielowątkowego

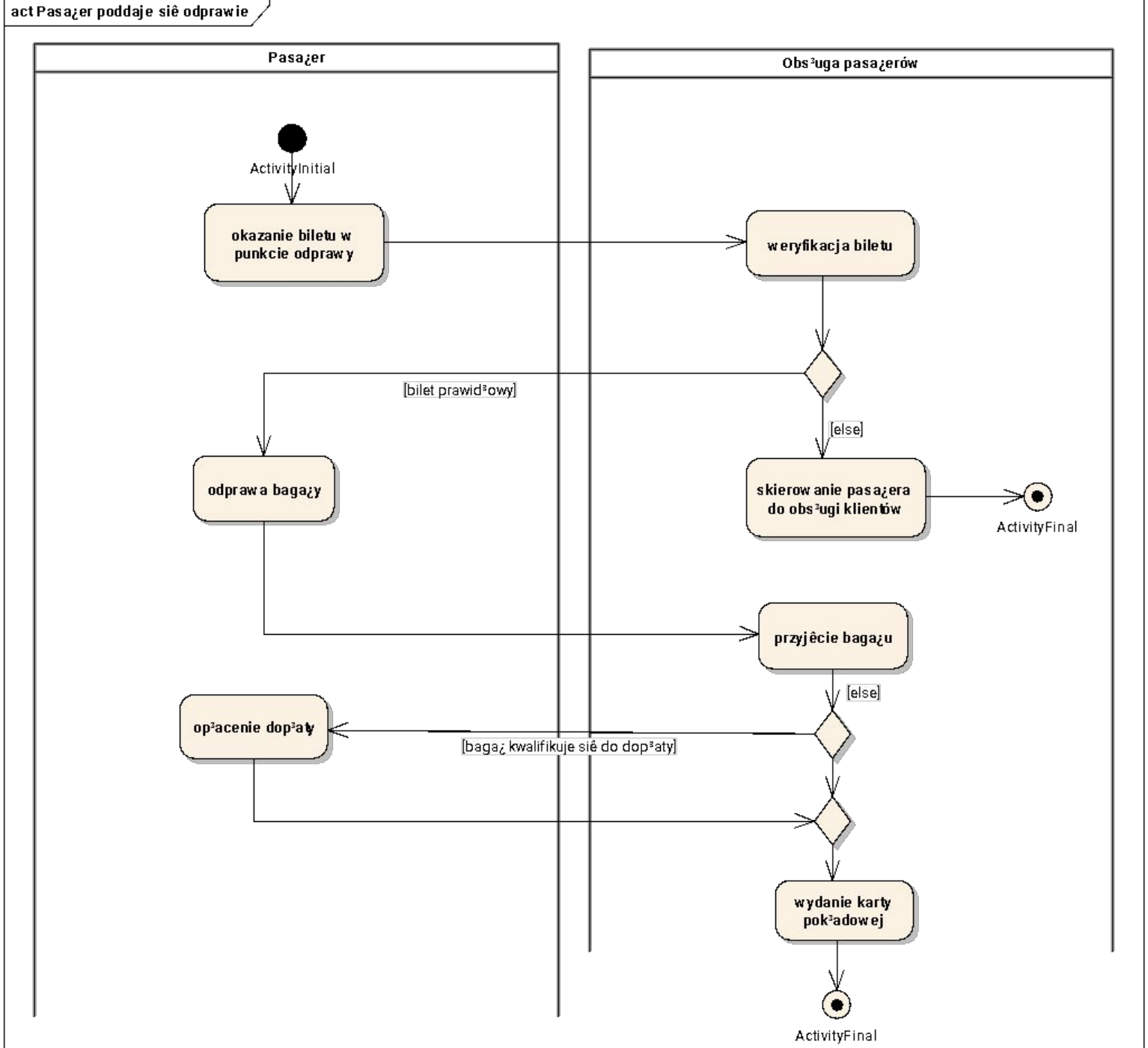

act Pasa¿er poddaje siê odprawie
Pasa¿er
Obs³uga pasa¿erów
ActivityInitial
okazanie biletu w punkcie odprawy
weryfikacja biletu
[bilet prawid³owy]
[else]
odprawa baga¿y
skierowanie pasa¿era do obs³ugi klientów
ActivityFinal
przyjêcie baga¿u
[else]
op³acenie dop³aty
[baga¿ kwalifikuje siê do dop³aty]
wydanie karty pok³adowej
ActivityFinal

Diagram komponentów (*Component diagram*)

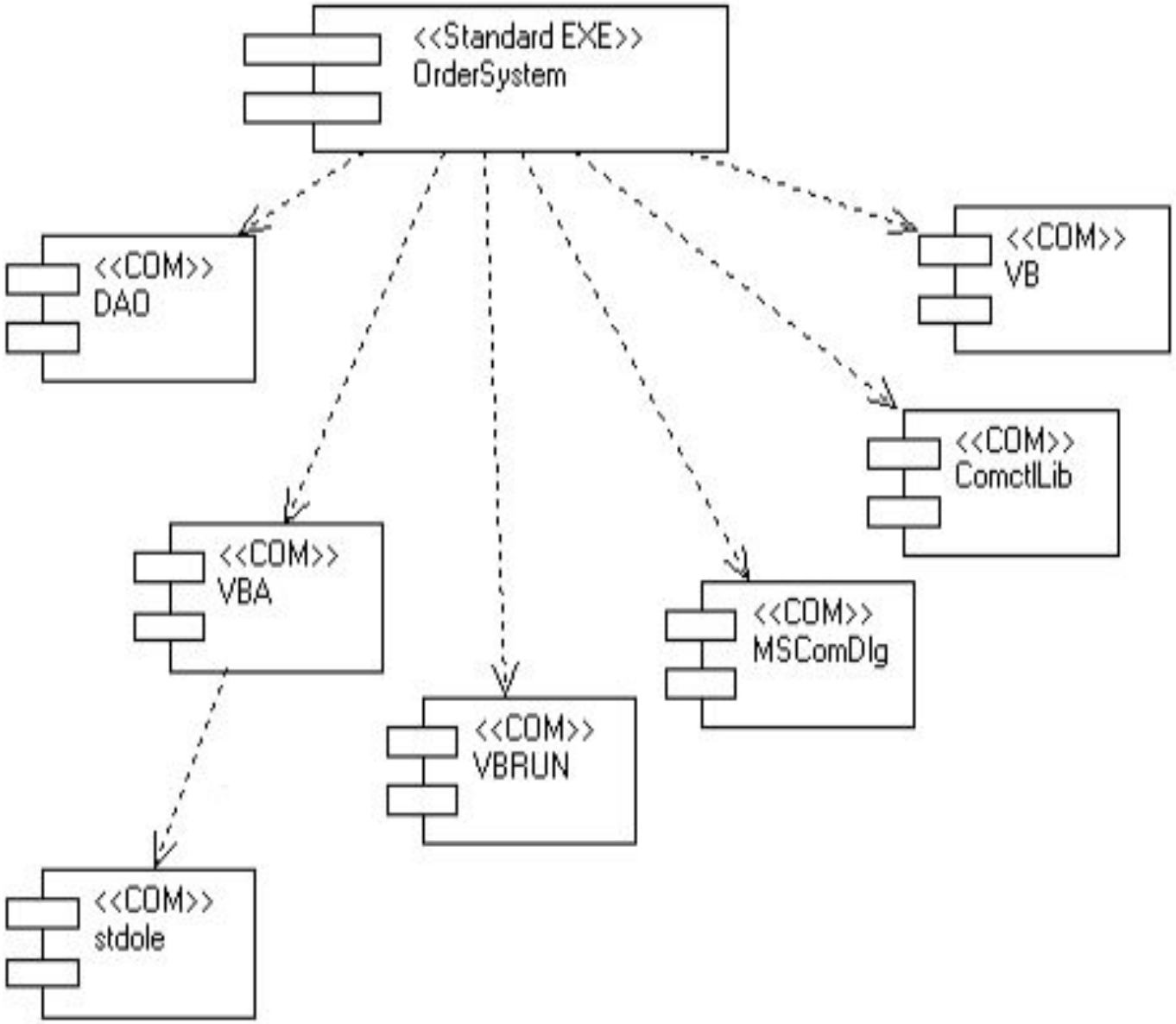


Diagram pakietów (*Package diagram*)

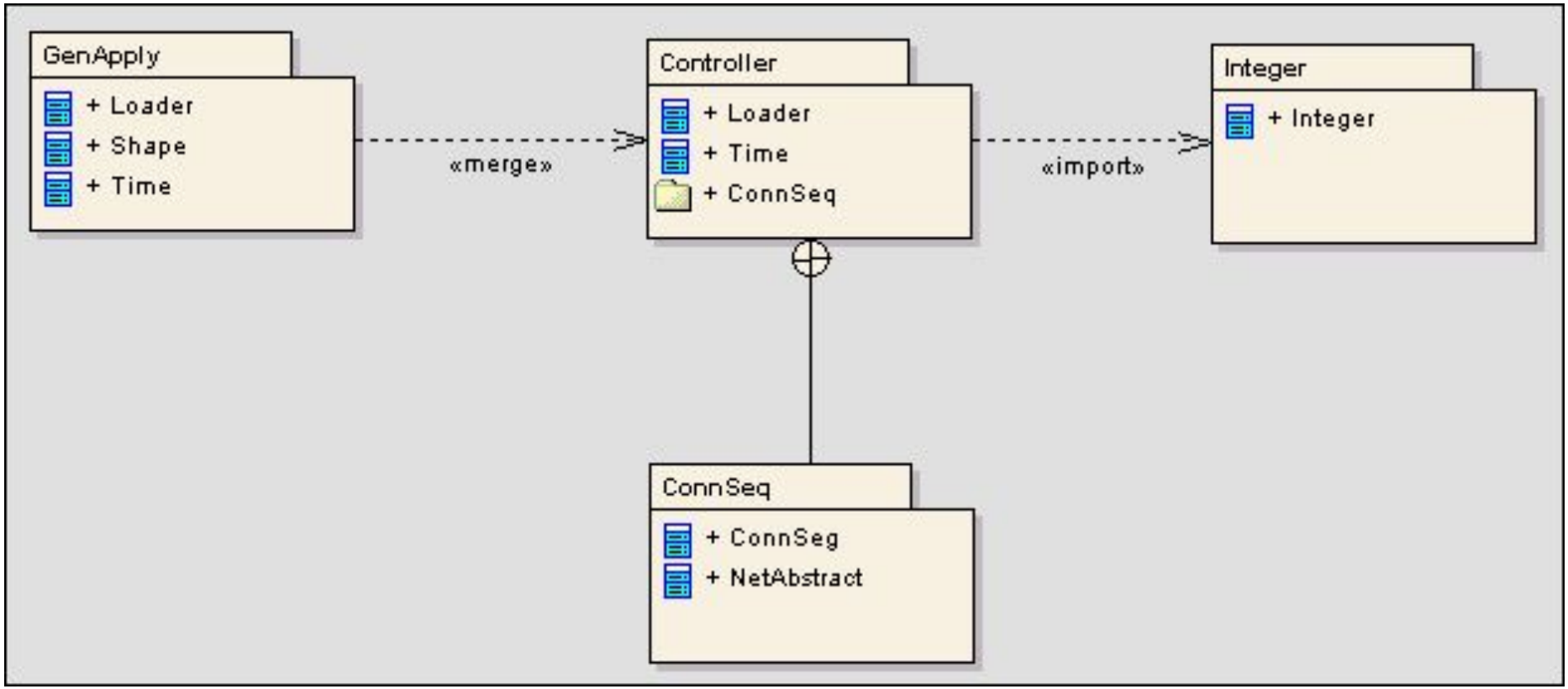
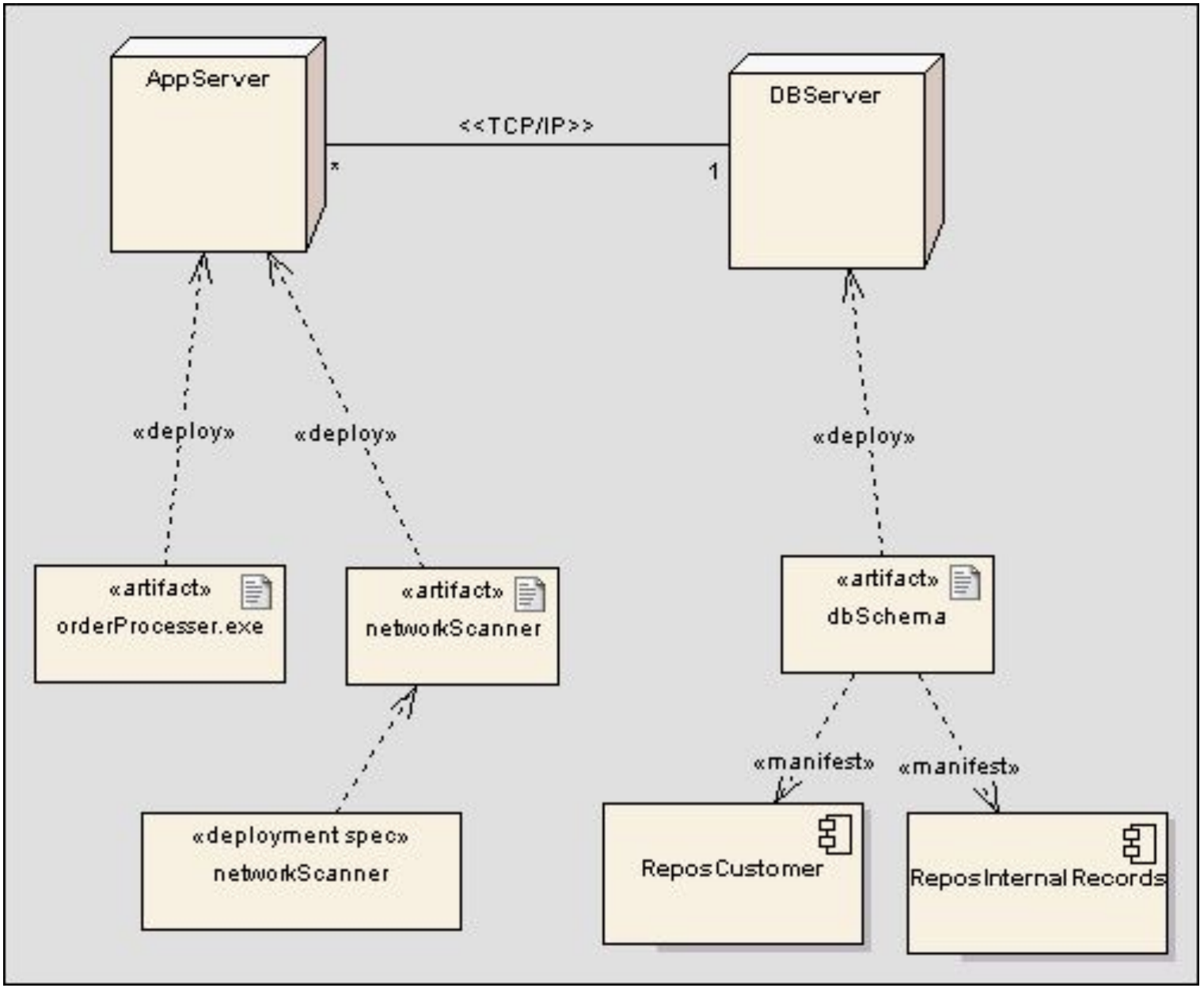


Diagram wdrożenia (*Deployment diagram*)



Zbiorowy diagram komponentów (*Composite structure diagram*)

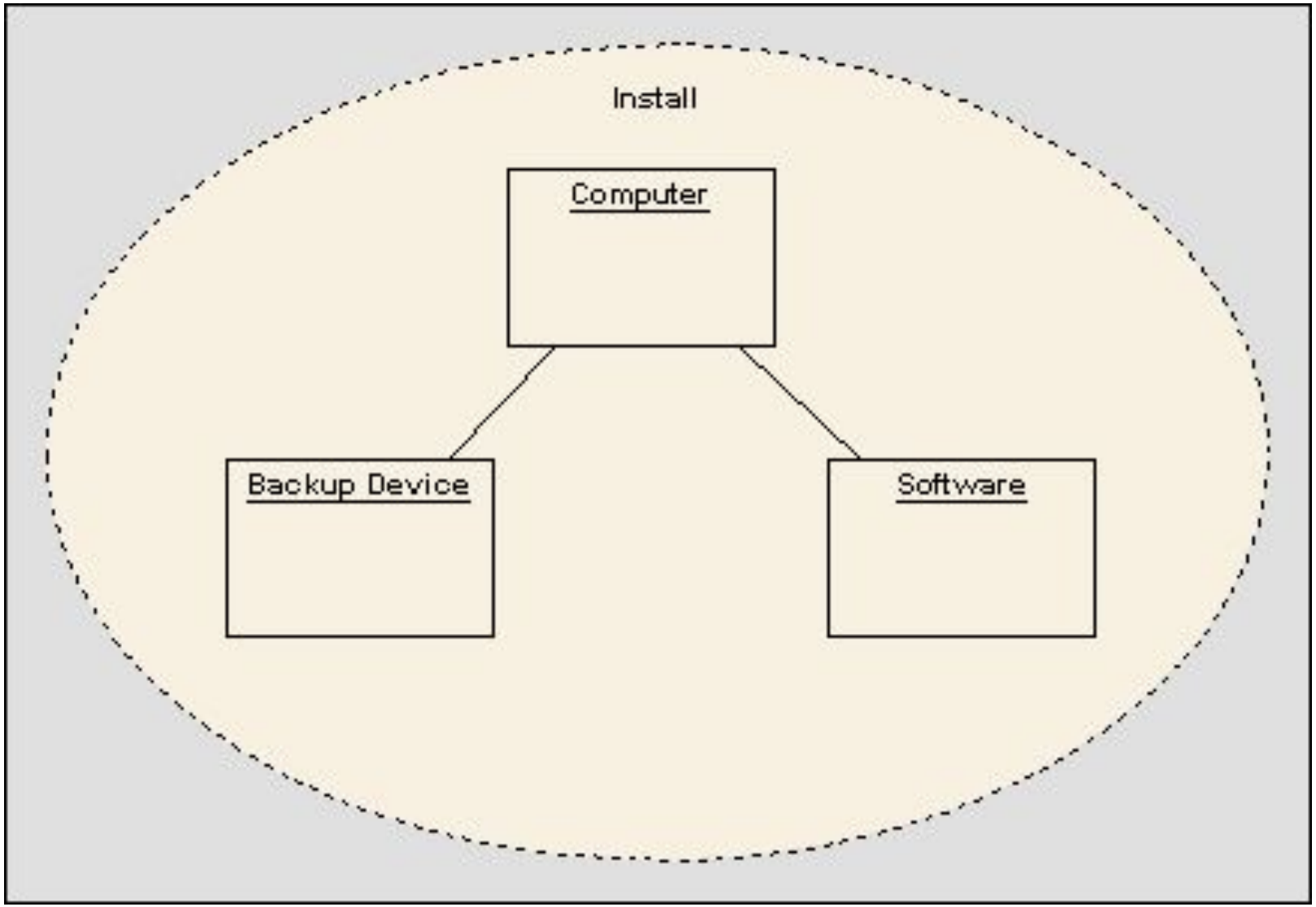


Diagram czynności (*Activity diagram*)

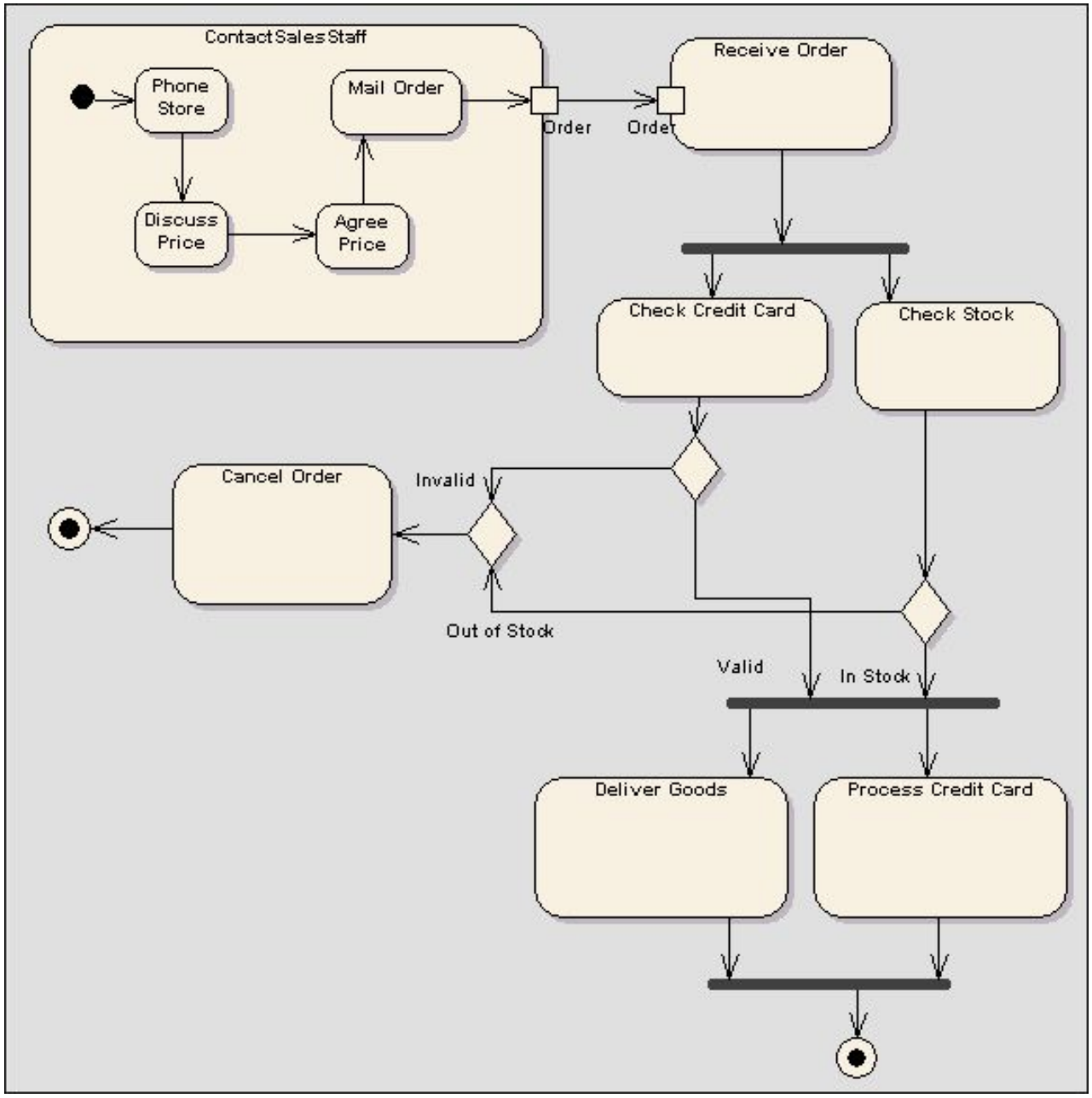


Diagram maszyny stanów (*State machine diagram*)

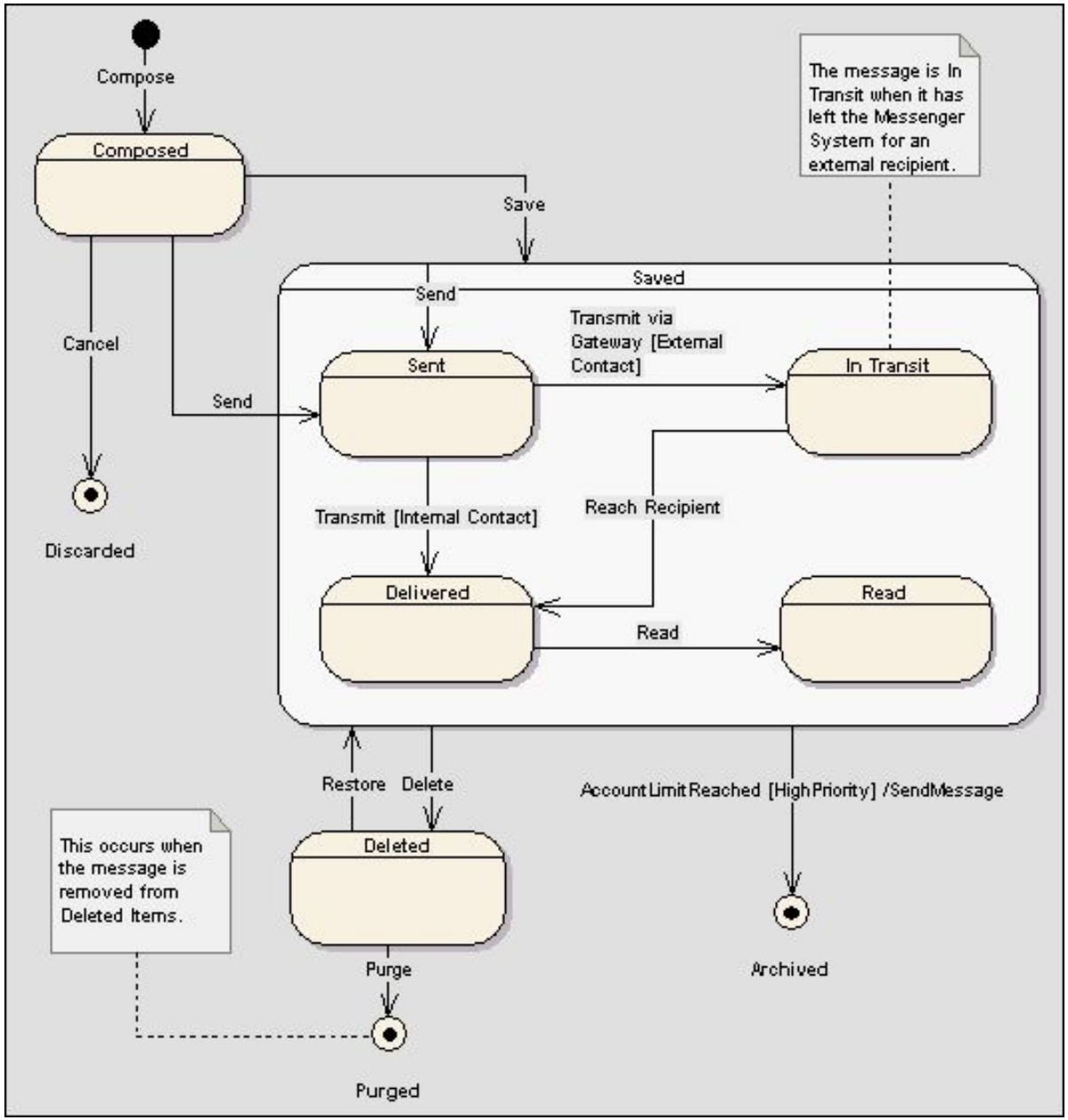


Diagram sekwencji (*Sequence diagram*)

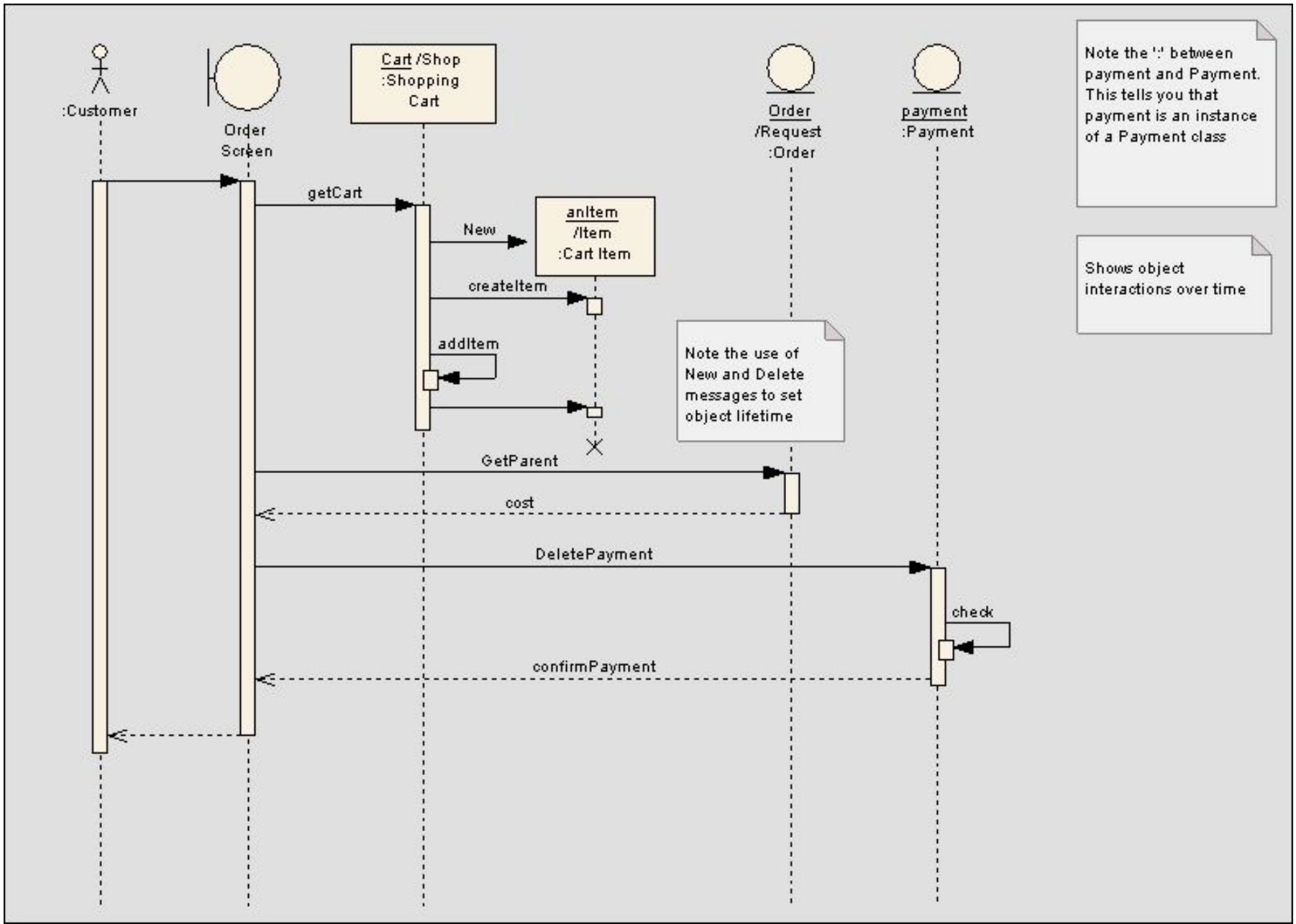


Diagram komunikacji (*Communication diagram*)

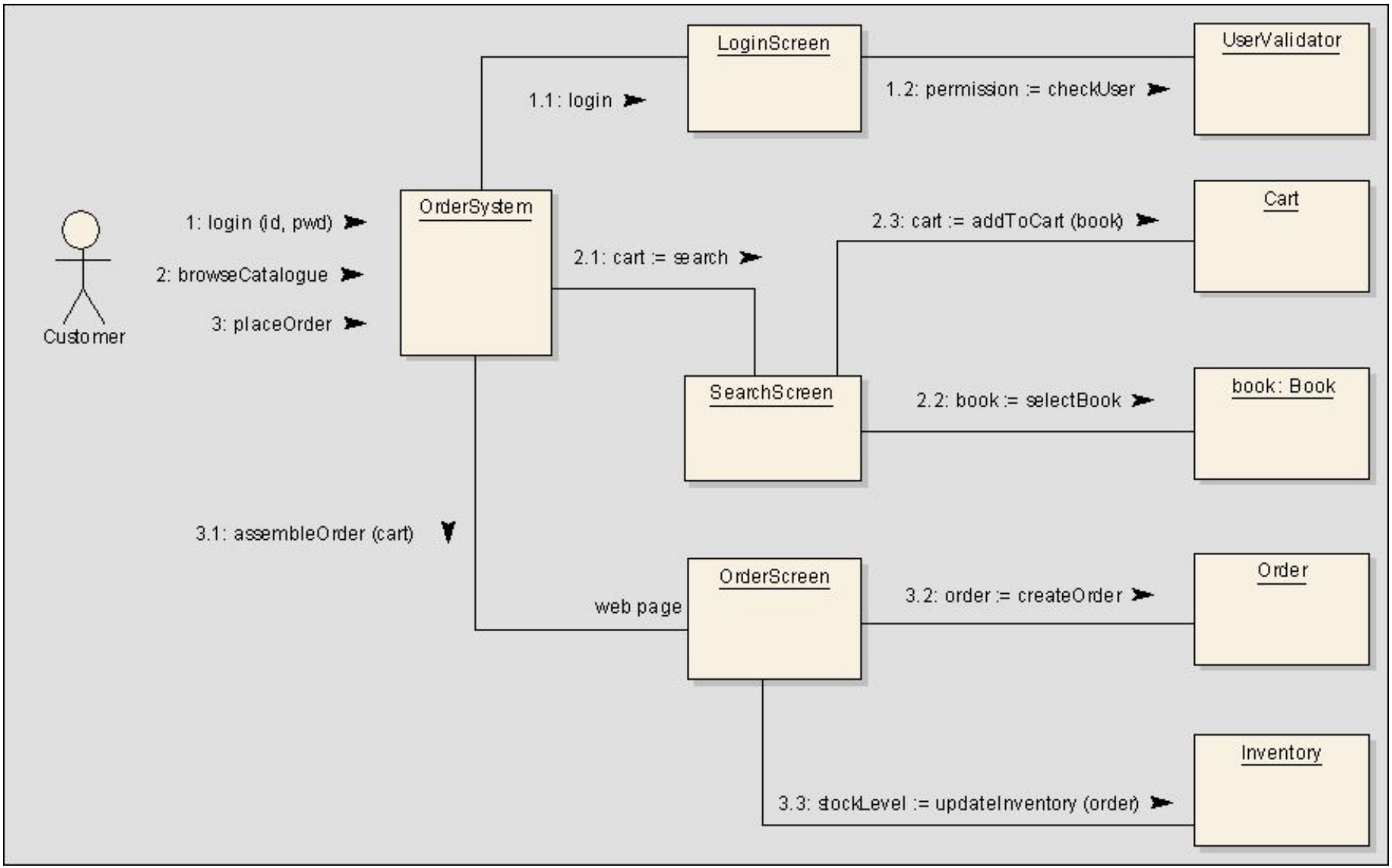


Diagram przeglądu współdziałania (*Interaction overview diagram*)

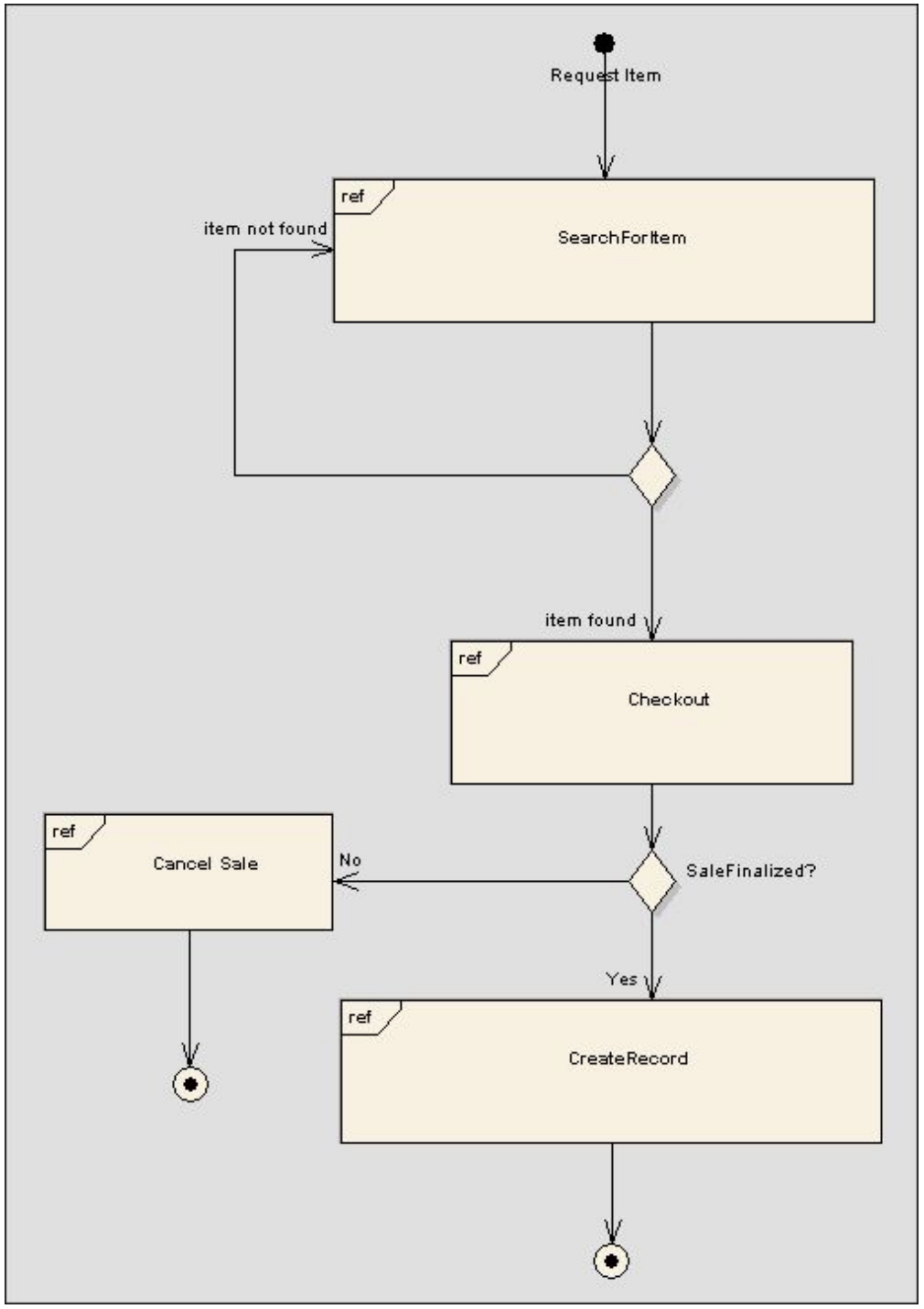
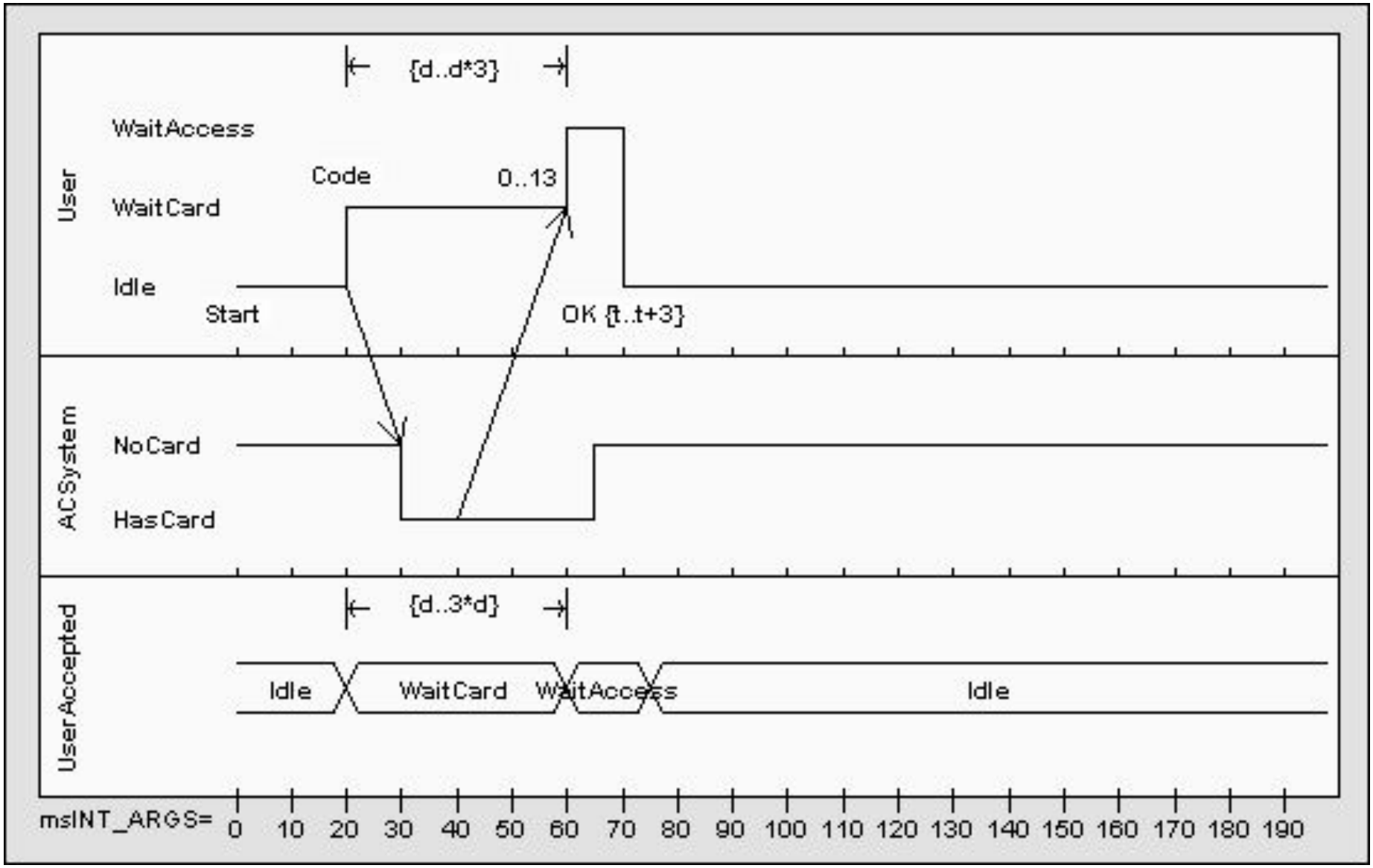


Diagram czasowy (*Timing diagram*)



Unified Modeling Language

Diagramy struktury – diagramy statyczne systemu

Dokumentują statyczne aspekty systemu:

- klasy
- obiekty
- komponenty
- wdrożenie systemu

Unified Modeling Language

Diagramy czynności – dynamiczne diagramy systemu

Dokumentują zachowania systemu:

- interakcje ze światem zewnętrznym
- współpracę obiektów systemu ze sobą
- przepływ danych w systemie
- komunikacja wewnątrz systemu