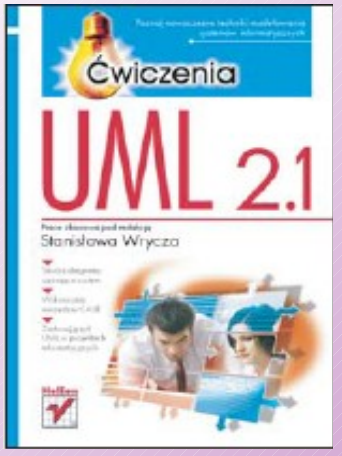
TECHNIKI PROGRAMOWANIA

Wykład 2:

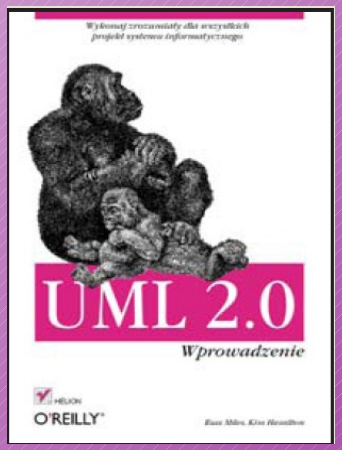
Języki programowania - UML

Ryszard Myhan

UML - bibliografia



UML 2.1. Ćwiczenia
Stanisław Wrycza i inni
ISBN: 978-83-246-0612-2



UML 2.0. Wprowadzenie
Russ Miles, Kim Hamilton
ISBN: 978-83-246-0632-0

UML 2.0 w akcji. Przewodnik oparty na projektach
Patrick Graessle, Henriette Baumann, Philippe Baumann
ISBN: 83-246-0646-7

Czym jest UML?

UML (*Unified Modeling Language*)

jest to sposób formalnego opisu modeli zazwyczaj reprezentujących projekty informatyczne.

UML

jest językiem znormalizowanym, służącym do zapisywania projektu systemu.

- Może być stosowany do obrazowania, specyfikowania, tworzenia i dokumentowania artefaktów powstałych podczas procesu budowy systemu informatycznego.

- **Wysoki poziom abstrakcji gwarantuje całkowitą niezależność od platformy i języka.**

Czym jest UML?

UML

dostarcza mechanizmów ułatwiających efektywną wymianę informacji i przekazywanie projektów innym.

UML

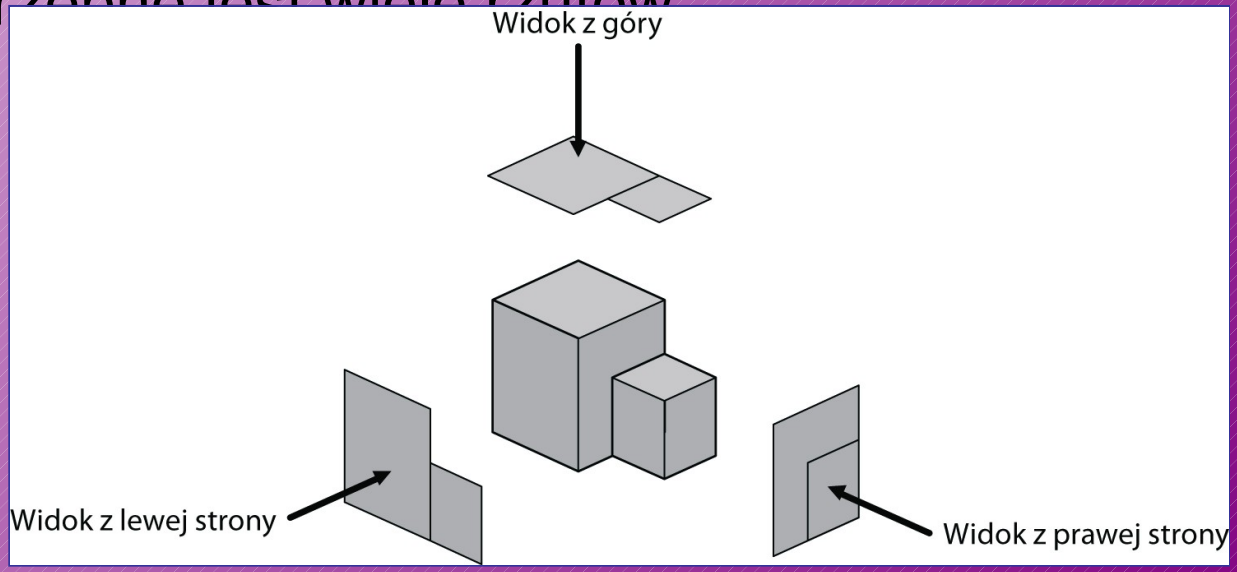
zawiera wiele elementów graficznych grupowanych w postaci diagramów. Ponieważ jest językiem, określa zasady łączenia tych elementów.

Celem diagramów jest pokazanie wielu perspektyw systemu, zestaw perspektyw stanowi model.

Należy podkreślić, że model opisuje, co system ma robić, ale nie określa, jak system ten ma zostać zaimplementowany.

Diagramy definiowane w UML

- Ze względu na mnogość aspektów projektowanych systemów żadna pojedyncza perspektywa spojrzenia na projektowany system nie jest wystarczająca.
- Tę sytuację można porównać do projektu złożonego mechanizmu - pojedynczy rzut tego mechanizmu na jedną płaszczyznę nie jest wystarczający do zrozumienia jego budowy; potrzebne jest wiele rzutów i przekrojów.



Diagramy definiowane w UML

- Wyróżnia się 14 diagramów podzielonych na dwie kategorie: strukturalne
  opisujące zachowania
- Każdy z nich opisuje dany system pod innym kątem, z innej perspektywy, i na różnym poziomie abstrakcji.
- Najistotniejsze z nich to:
  - ⇨ Diagram klas (diagram strukturalny)
  - ⇨ Diagram obiektów
  - ⇨ Diagram przypadków użycia
  - ⇨ Diagram aktywności
  - ⇨ Diagram Automatu Stanów
  - ⇨ Diagram sekwencyjny

Diagramy przypadków użycia

- **Diagramy przypadków użycia** (*use case*) odwzorowują pewne funkcje systemu w taki sposób, w jaki będą je widzieć jego przyszli użytkownicy.
- Dla dużych systemów o wielu złożonych i wzajemnie powiązanych funkcjach tego rodzaju podejście pozwala zapomnieć o strukturze/architekturze systemu i jego detalach technicznych i skoncentrować się na zewnętrznych funkcjach systemu.
- Podejście do projektowania od strony przypadków użycia sprzyja punktowi widzenia, w którym centralnym ośrodkiem zainteresowania staje się człowiek - przyszły użytkownik systemu - a nie budowa mechanizmu systemu.

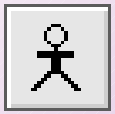
Diagramy przypadków użycia

- Celem tej metody jest:
  - Głębsze zrozumienie użycia systemu będącego przedmiotem procesu projektowania.
  - Zwiększenie stopnia świadomości analityków i projektantów co do celów tego systemu.
  - Umożliwienie interakcji zespołu projektowego z przyszłymi użytkownikami systemu.
  - Weryfikacja poprawności i kompletności projektu.
  - Ustalenie wszystkich strukturalnych i funkcjonalnych własności systemu.
  - Ustalenie składowych systemu i związanego z nimi planu konstrukcji systemu.
  - Dostarczenie podstawy do sporządzenia planu testów systemu.

Diagramy przypadków użycia

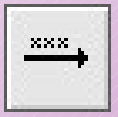
- Model przypadków użycia dostarcza bardzo abstrakcyjnego poglądu na system z pozycji aktorów, którzy go używają.
- Diagram przypadków użycia zawiera znaki graficzne oznaczające aktorów (ludziki) oraz przypadki użycia (owale z wpisanym tekstem).
  - Te oznaczenia połączone są liniami odwzorowującymi powiązania poszczególnych aktorów z poszczególnymi przypadkami użycia.
- Metoda przypadków użycia wymaga od analityka określenia wszystkich aktorów związanych w jakiś sposób z projektowanym systemem.
  - Każdy aktor używa lub będzie używać systemu na kilka specyficznych sposobów (przypadków użycia).

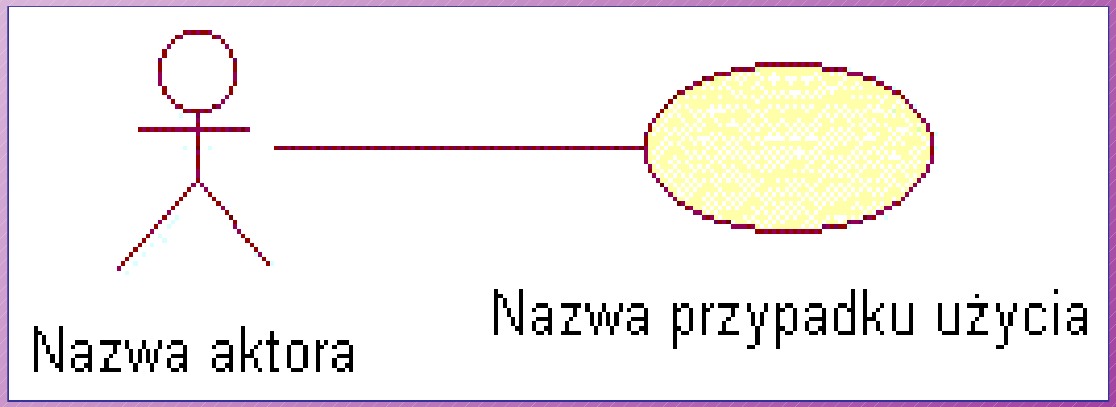
Diagramy przypadków użycia - notacja

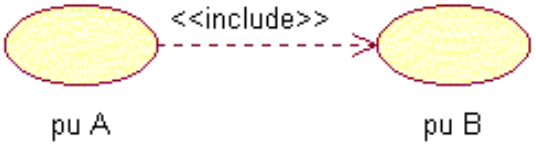
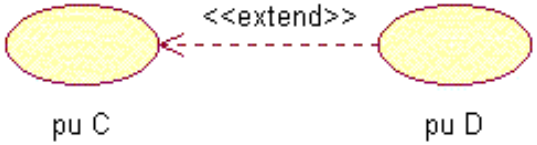
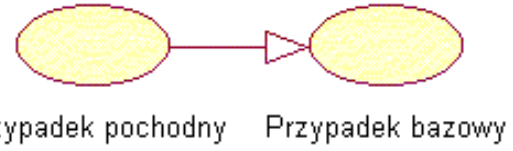

Aktor


Przypadek użycia


Specjalizacja


Powiązania między przypadkami użycia



Związki pomiędzy przypadkami użycia		
Zawieranie	Rozszerzanie	Uogólnienie
		
przypadek użycia A **ZAWSZE** włącza przypadek użycia B	przypadek użycia D **CZASAMI** (w pewnych sytuacjach) rozszerza przypadek użycia C	analogiczne do dziedziczenia klas

Diagramy przypadków użycia

- Przypadek użycia reprezentuje sekwencję operacji lub transakcji wykonywanych przez system w trakcie interakcji z użytkownikiem.
- Przypadki użycia reprezentują przepływ zdarzeń w systemie i są uruchamiane (inicjowane) przez aktorów.
- Przypadek użycia jest zwykle charakteryzowany przez krótką nazwę, ale może też zawierać informację:
  - Jak i kiedy przypadek użycia zaczyna się i kończy?
  - Opis interakcji przypadku użycia z aktorami, włączając w to *kiedy* interakcja ma miejsce i *co* jest przesyłane.
  - Kiedy i do czego przypadek użycia potrzebuje danych zapamiętanych w systemie, lub jak i kiedy zapamiętuje dane w systemie?
  - Wyjątki przy przepływie zdarzeń.
  - Jak i kiedy używane są pojęcia dziedziny problemowej?

Diagramy przypadków użycia

- Typowa dokumentacja przypadków użycia powinna zawierać następujące elementy:
  - ⇨Krótki opis przypadku użycia.
  - ⇨Przepływ zdarzeń opisany nieformalnie.
  - ⇨Związki pomiędzy przypadkami użycia.
  - ⇨Uczestniczące obiekty.
  - ⇨Specjalne wymagania (np. czas odpowiedzi, wydajność).
  - ⇨Obrazy interfejsu użytkownika.
  - ⇨Ogólny pogląd na przypadki użycia (powiązania w postaci diagramów).
  - ⇨Diagramy interakcji dla każdego aktora.

:

Diagram przypadków użycia wymiany wiadomości

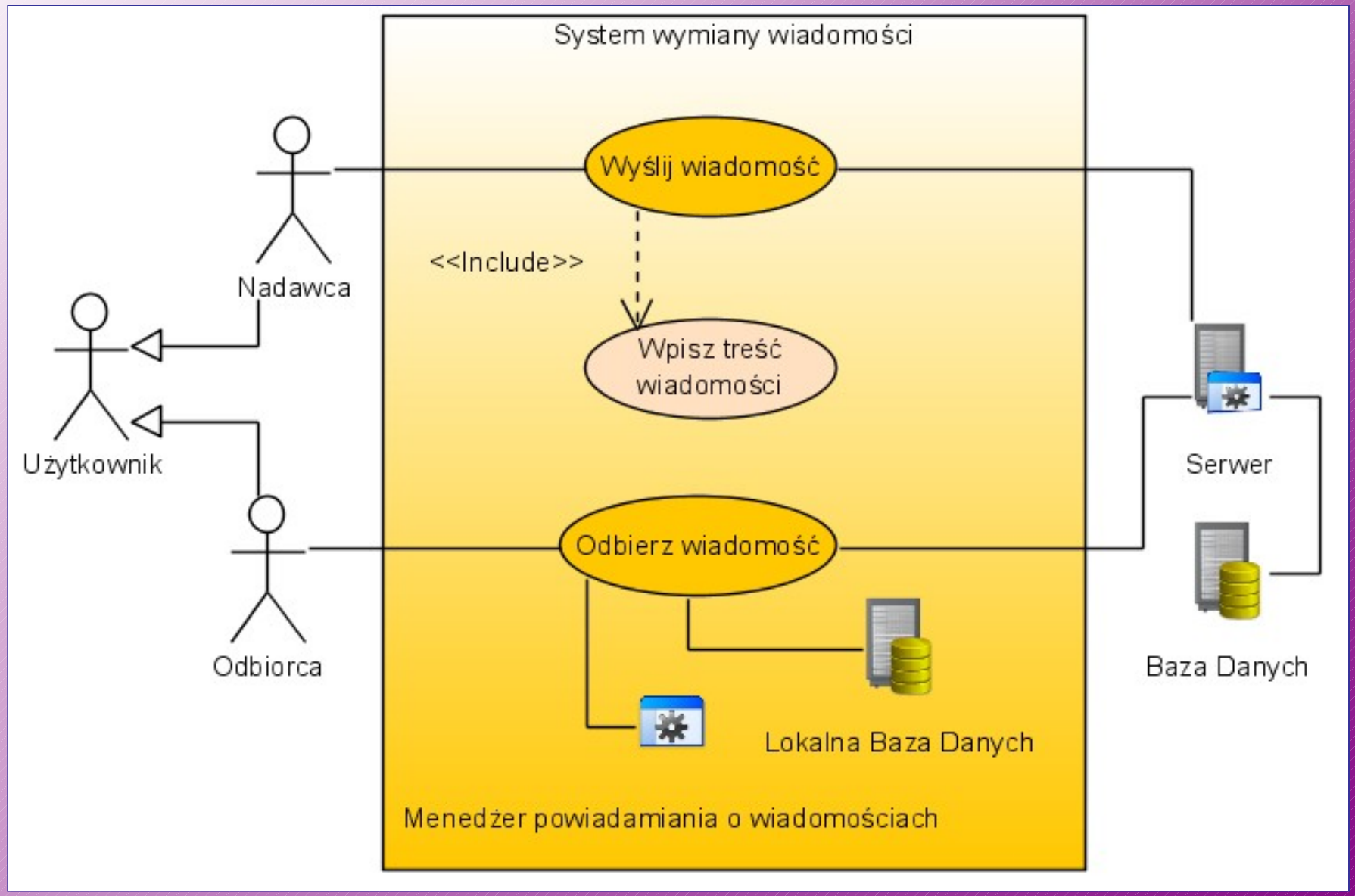


Diagram przypadków użycia dostępu do usług serwera

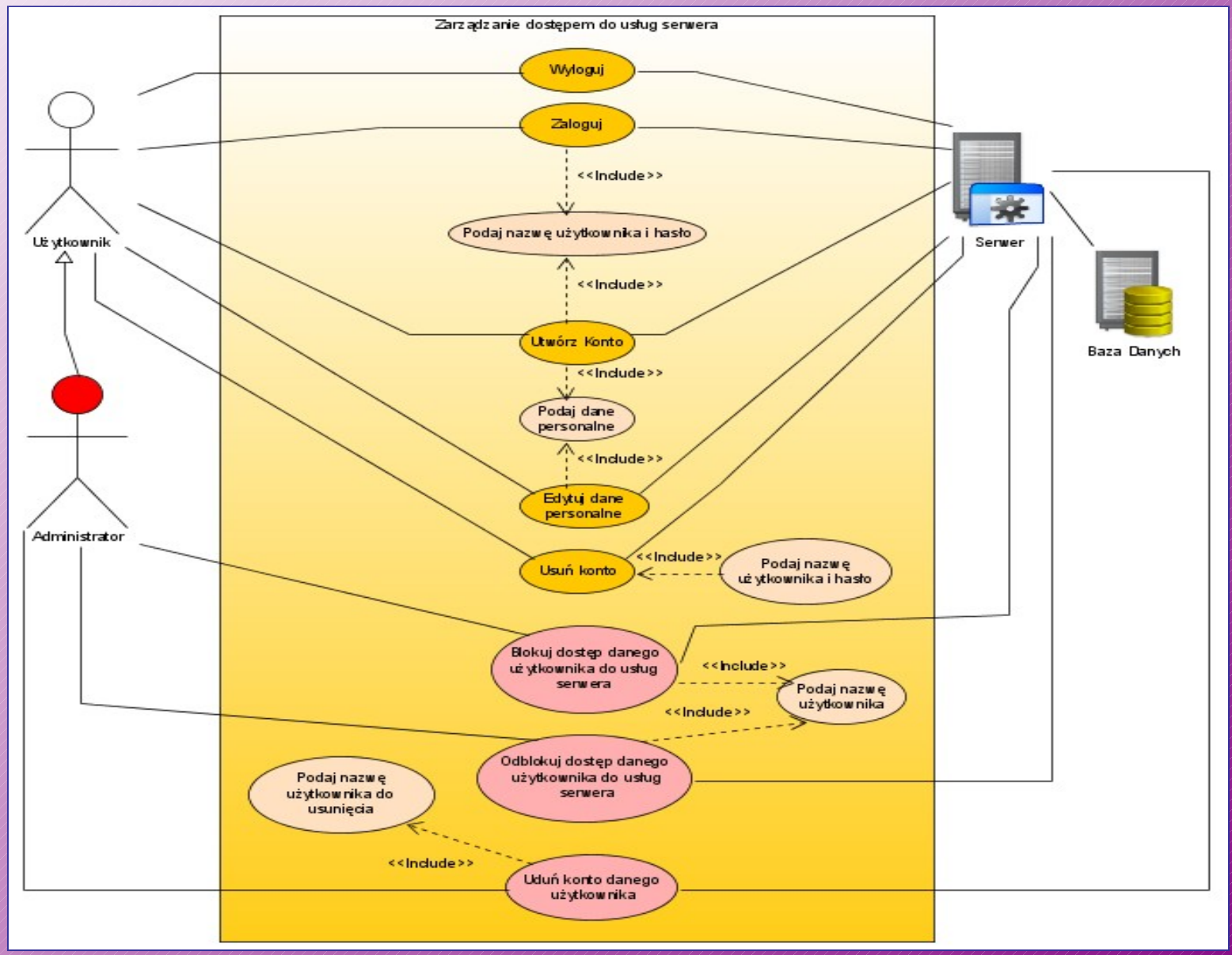


Diagram przypadków użycia dostępu do usług serwera

Dla potrzeb zarządzania kontami wyszczególnieni zostali następujący aktorzy:

- **Użytkownik**
- **Administrator** — dziedziczy funkcje po użytkowniku, ponad to posiada możliwość zarządzania dostępem użytkowników do logowania się.
- **Serwer** — aplikacja świadcząca usługi oraz zarządzająca przepływem danych i ich przetwarzaniem.
- **Baza danych** — system przechowujący informacje.

Diagram przypadków użycia dostępu do usług serwera

Logowanie: proces mający na celu uwierzytelnienie użytkownika w wyniku, którego zostaje nawiązana sesja komunikacyjna między serwerem a aplikacją kliencką.

aktorzy:	Użytkownik, Administrator, Serwer, Baza Danych
wymagania wstępne:	Użytkownik musi posiadać konto
wynik działania:	Dostęp do usług wymagających zalogowania
dane wejściowe:	Nazwa użytkownika, hasło użytkownika
wyjście:	Klucz sesji

przepływ zdarzeń:

1. Użytkownik wykonuję próbę logowania.
2. Serwer autoryzuje dane użytkownika.
3. Zmiana statusu lub wyświetlana jest informacja o błędzie autoryzacji.

Diagram przypadków użycia dostępu do usług serwera

Wylogowanie: proces mający na celu wysłanie informacji do serwera o zakończeniu sesji komunikacyjnej i zmiany statusu.

aktorzy:	Użytkownik, Administrator, Serwer, Baza Danych
wymagania wstępne:	Użytkownik musi posiadać konto oraz musi być zalogowany
wynik działania:	Zakończenie sesji komunikacyjnej
dane wejściowe:	Brak
wyjście:	Brak

przepływ zdarzeń:

1. Użytkownik wysyła żądanie wylogowania.
2. Serwer zamyka sesję komunikacyjną.

Diagram klas

- **Diagram klas** (zwany także *diagramem asocjacji klas* lub *modelem obiektowym*) jest pojęciem centralnym we wszystkich znanych metodykach obiektowych.
- **Diagram klas** ukazuje wzajemne powiązania między klasami tworzącymi dany system – nie ukazuje on jednak żadnych relacji pomiędzy samymi obiektami.
- Z reguły diagram klas jest zmodyfikowanym diagramem encji, rozbudowanym o nowe elementy.
- W porównaniu do diagramów encji diagramy klas wprowadzają istotną nowość, mianowicie metody przypisane do specyfikowanych klas.
- Oprócz tego zasadniczego elementu pojawiają się w diagramach klas różnorodne oznaczenia o charakterze pomocniczym.

Diagram klas - związki

- **Diagram klas** pokazuje klasy w postaci pewnych oznaczeń graficzno-językowych powiązanych w sieć zależnościami należącymi do trzech kategorii:

⇨ **Dziedziczenie** (*inheritance*), czyli ustalenie związku generalizacji/specjalizacji pomiędzy klasami.

⇨ **Asocjacja** (*association*), czyli dowolny związek pomiędzy obiektami dziedziny przedmiotowej, który ma znaczenie dla modelowania.

⇨ **Agregacja** (*aggregation*), czyli szczególny przypadek asocjacji, odwzorowujący stosunek całość-część pomiędzy obiektami z modelowanej dziedziny przedmiotowej.

Diagram klas - zastosowanie

- **Diagram klas** jest stosowany zarówno do zapisu wyników analizy jak i do specyfikowania założeń projektowych.
- Istnieją trzy podstawowe zastosowania diagramów klas:
  - Zapis modelu pojęciowego - model nie musi być związany z jakimkolwiek oprogramowaniem; jest wyłącznie sformalizowaną wizją wyobrażeń powstających w trakcie twórczych procesów myślowych związanych z analizowanym problemem.
  - Sformalizowana specyfikacja danych i metod - jest ona bardziej związana z oprogramowaniem, ale dotyczy jego zewnętrznego opisu (interfejsów) bez wchodzenia w szczegóły implementacyjne.
  - Implementacja - może służyć jako graficzny środek pokazujący szczegóły implementacji klas, np. w C++.

Diagram klas - oznaczenia

★ W wersji abstrahującej od szczegółów klasa jest zapisana w postaci prostokąta z wpisaną w środku nazwą klasy.

★ W wersjach bardziej szczegółowych klasa jest reprezentowana przez prostokąt z trzema polami, gdzie:

- pierwsze pole zawiera nazwę klasy,
- drugie pole zawiera specyfikację atrybutów, które posiadają obiekty tej klasy,
- trzecie pole zawiera specyfikację metod.

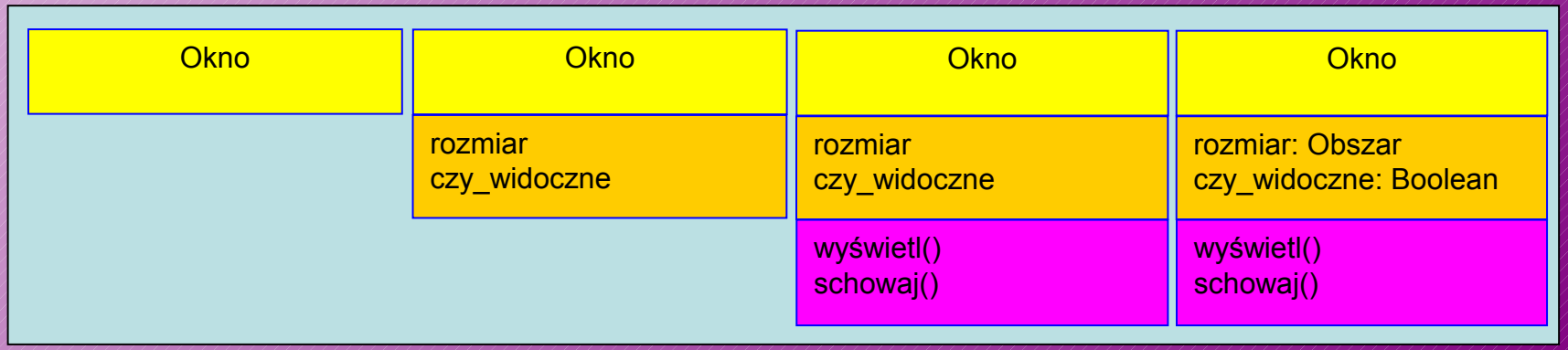
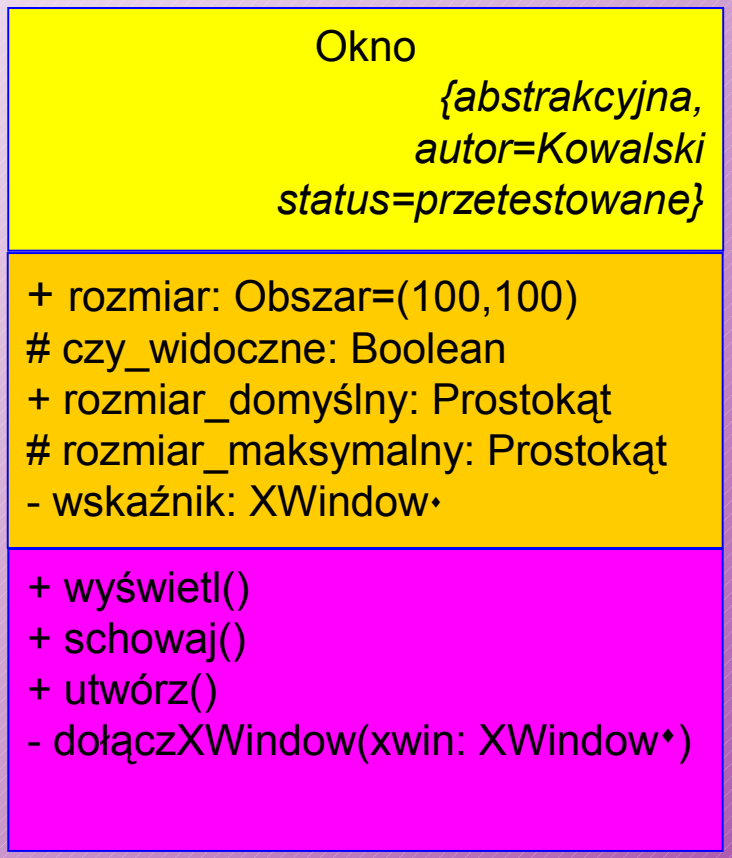


Diagram klas - oznaczenia

★ bardzo Klasy mogą też być wyspecyfikowane w sposób zbliżony do specyfikacji w języku programowania.



gdzie zasięg określany jest w jeden z następujących sposobów:

\+ publiczny (*public*)

\# chroniony (*protected*)

\- prywatny (*private*)

Diagram klas - przykład

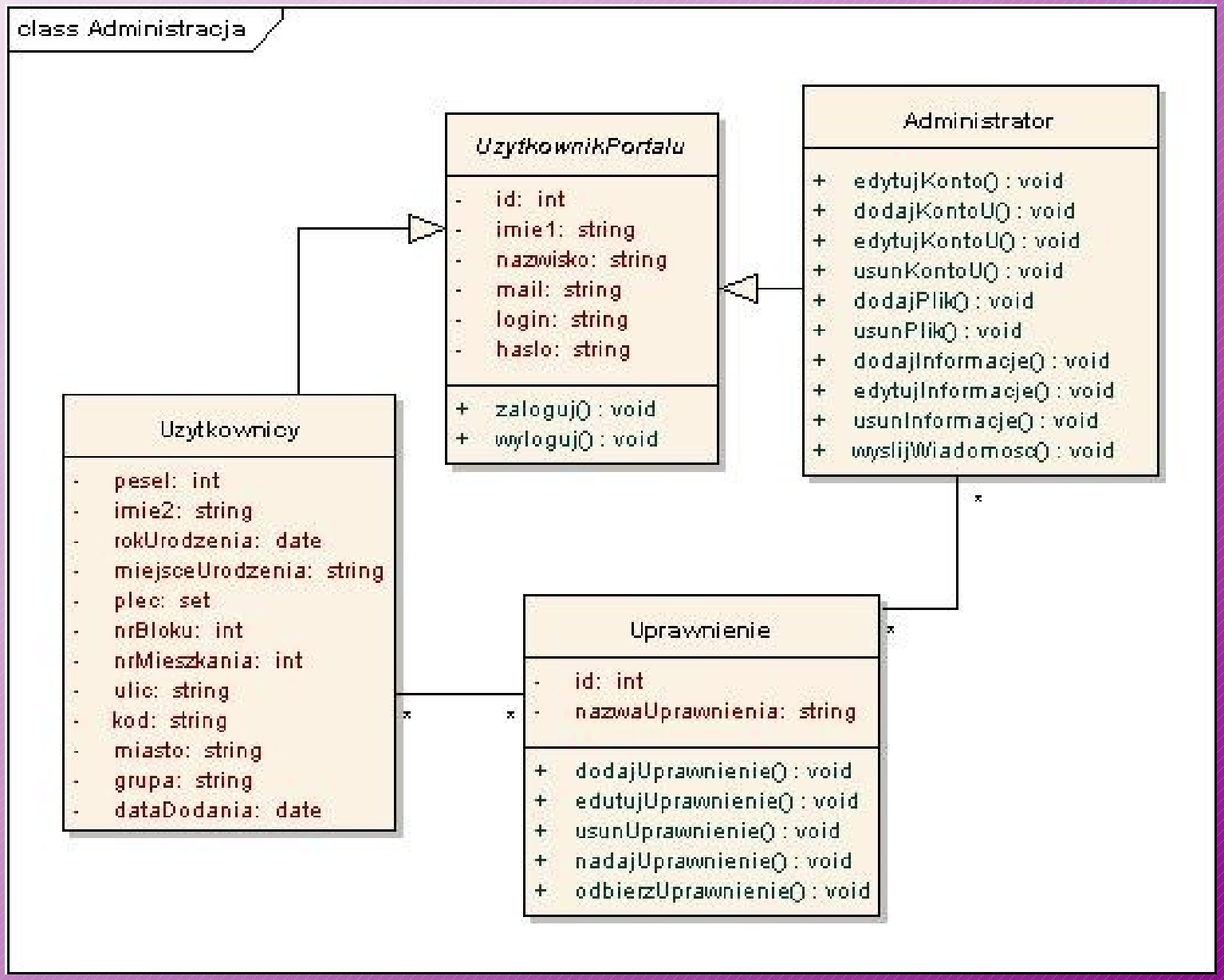


Diagram klas - przykład

Użytkownicy portalu: klasa abstrakcyjna, zawierająca podstawowe dane o użytkownikach i administratorze portalu.

Atrybuty:

id (int)	typ całkowitoliczbowy, który jest nadawany jako auto_increment (automatycznie);
imie1 (string)	typ tekstowy, który zawiera imię użytkownika portalu;
nazwisko (string)	typ tekstowy, który zawiera nazwisko użytkownika portalu;
mail (string)	typ tekstowy, który zawiera adres email użytkownika portalu;
login (string)	typ tekstowy, który zawiera login użytkownika portalu, potrzebny przy dostępie do danych, które wymagają rejestracji;
hasło (string)	typ tekstowy, który zawiera hasło użytkownika portalu, potrzebne przy dostępie do danych, które wymagają rejestracji.

Diagram klas - przykład

Użytkownicy portalu: klasa abstrakcyjna, zawierająca podstawowe dane o użytkownikach i administratorze portalu.

Operacje:

Zaloguj	operacja, która sprawdza poprawność danych wprowadzonych przez użytkownika, z danymi zawartymi w bazie.
Wyloguj	wprowadzone przy zalogowaniu dane zostają utracone - użytkownik nie może korzystać z części serwisu wymagających rejestracji i logowania

Diagram klas - relacje

★ Pojedyncze klasy nie mają praktycznie żadnej wartości. Dopiero współpraca kilku lub większej liczby klas ma rzeczywiste znaczenie.

★ Współpraca klas jest definiowana poprzez tzw. relacje, reprezentowane przez linie (krawędzie) z odpowiednim zakończeniem – grotem. Grot wskazuje zawsze tę klasę, która jest w danej relacji ważniejsza.

PRZYKŁAD:

W tym przypadku klasa SettingsWindow (na którą grot wskazuje) jest rodzicem klasy SettingsWindowWin32

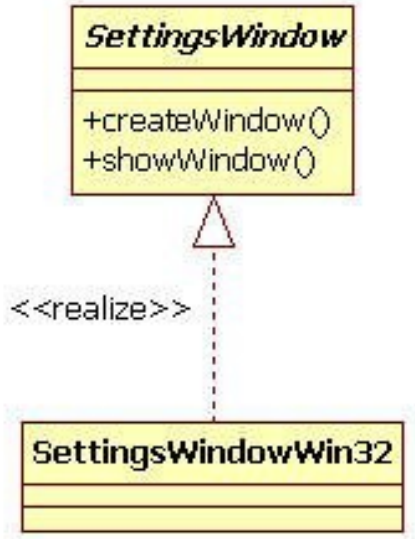
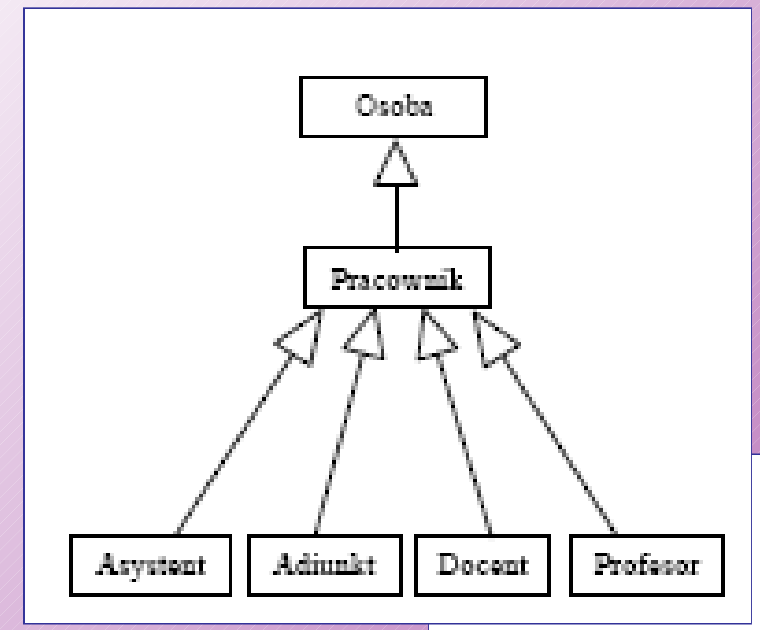
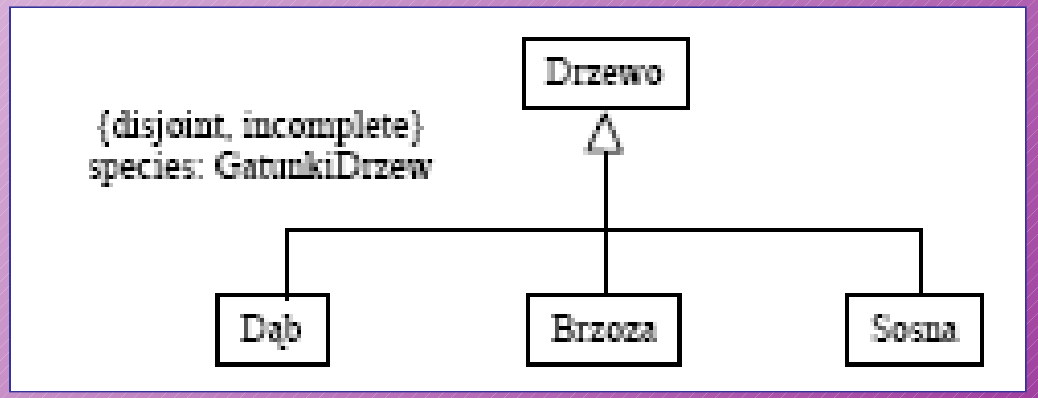


Diagram klas - relacje





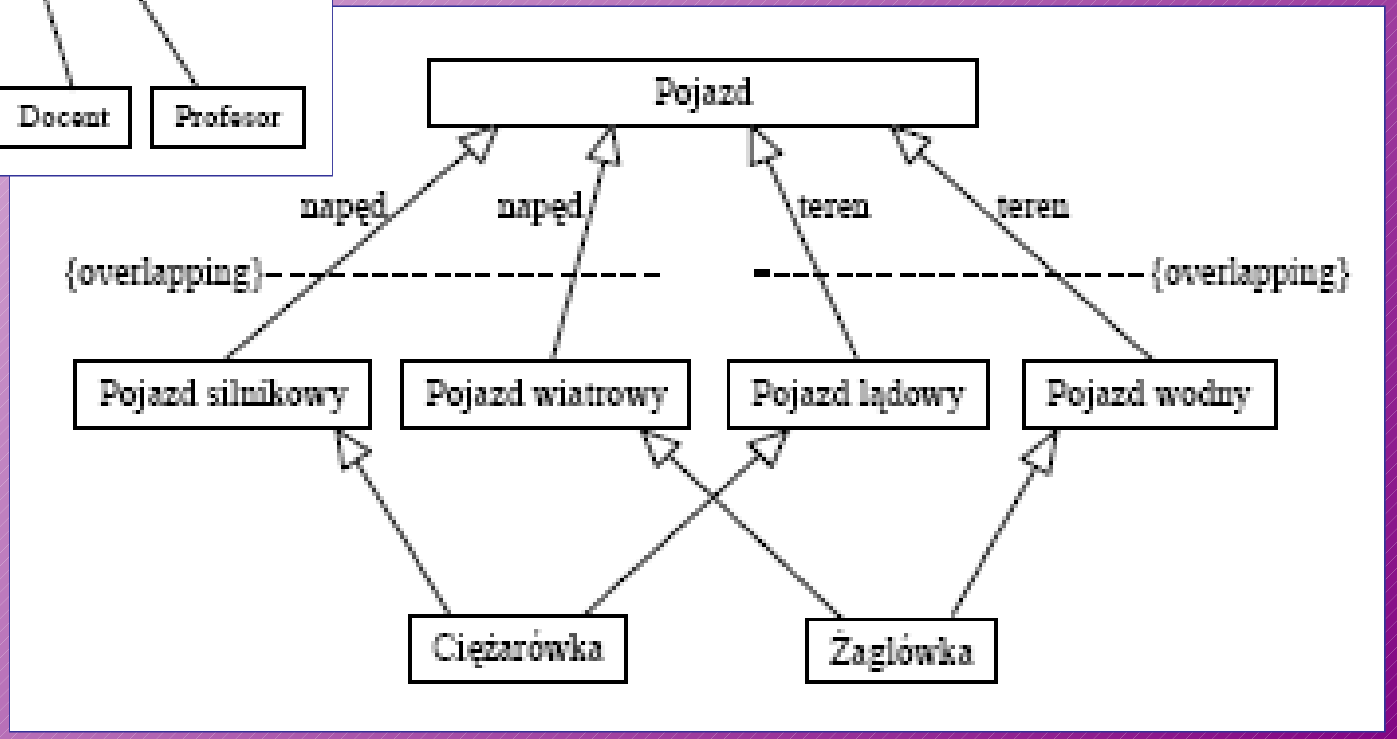
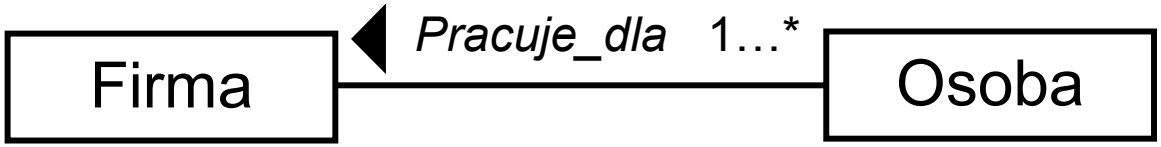


Diagram klas - asocjacje

★ Oznaczenia klas w UML mogą być połączone liniami oznaczającymi asocjacje, czyli powiązania pomiędzy obiektami tych klas.

★ Asocjacje mogą być wyposażone w oznaczenia liczności - liczność oznacza, ile obiektów innej klasy może być powiązane z jednym obiektem danej klasy

PRZYKŁAD:



Rysunek pokazuje specyfikację asocjacji *Pracuje_dla* pomiędzy obiektami klasy *Osoba* i obiektami klasy *Firma*. Czarny trójkącik określa kierunek wyznaczony przez nazwę powiązania (w danym przypadku określa on, że osoba pracuje dla firmy, a nie firma pracuje dla osoby)

Diagram klas – asocjacje (przykład)

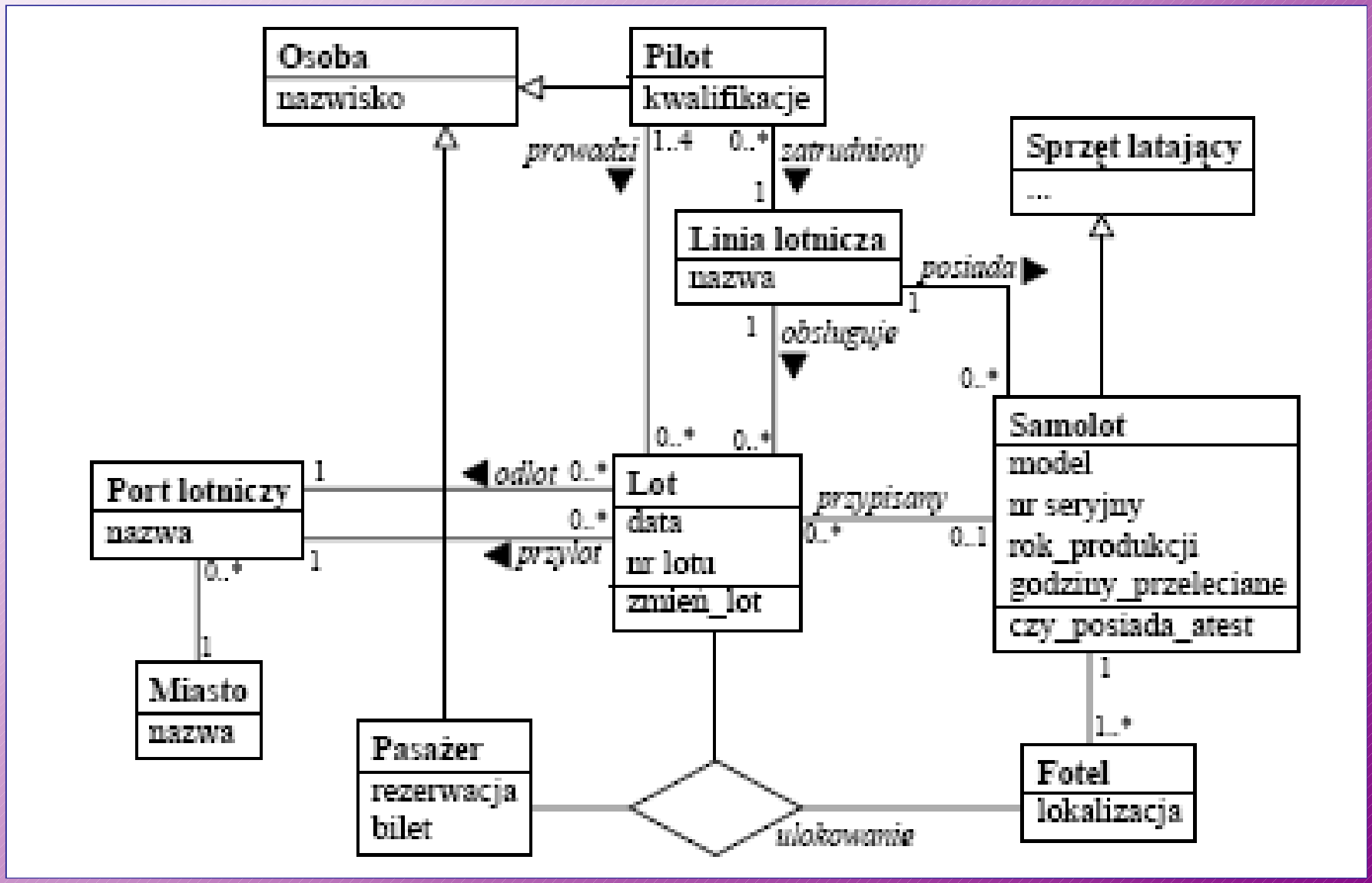


Diagram klas – agregacje i kompozycje

★ **Agregacja** jest szczególnym przypadkiem asocjacji wyrażającym zależność część-całość. Np. silnik jest częścią samochodu, czyli obiekt-samochód jest agregatem obiektów będących jego częściami.

★ **Związek kompozycji** oznacza, że dana część może należeć tylko do jednej całości. Co więcej, część nie może istnieć bez całości, pojawia się i jest usuwana razem z całością. Usunięcie całości powoduje automatyczne usunięcie wszystkich jej części związanych z nią związkiem kompozycji.

Diagram klas – agregacje i kompozycje

PRZYKŁAD:

Każde wystąpienie obiektu *Punkt* należy albo do obiektu *Wielobok* albo do obiektu *Okrąg*; nie może należeć do dwóch obiektów naraz.
Wystąpienie obiektu *Styl* może być dzielone przez wiele obiektów *Wielobok* i *Okrąg*.

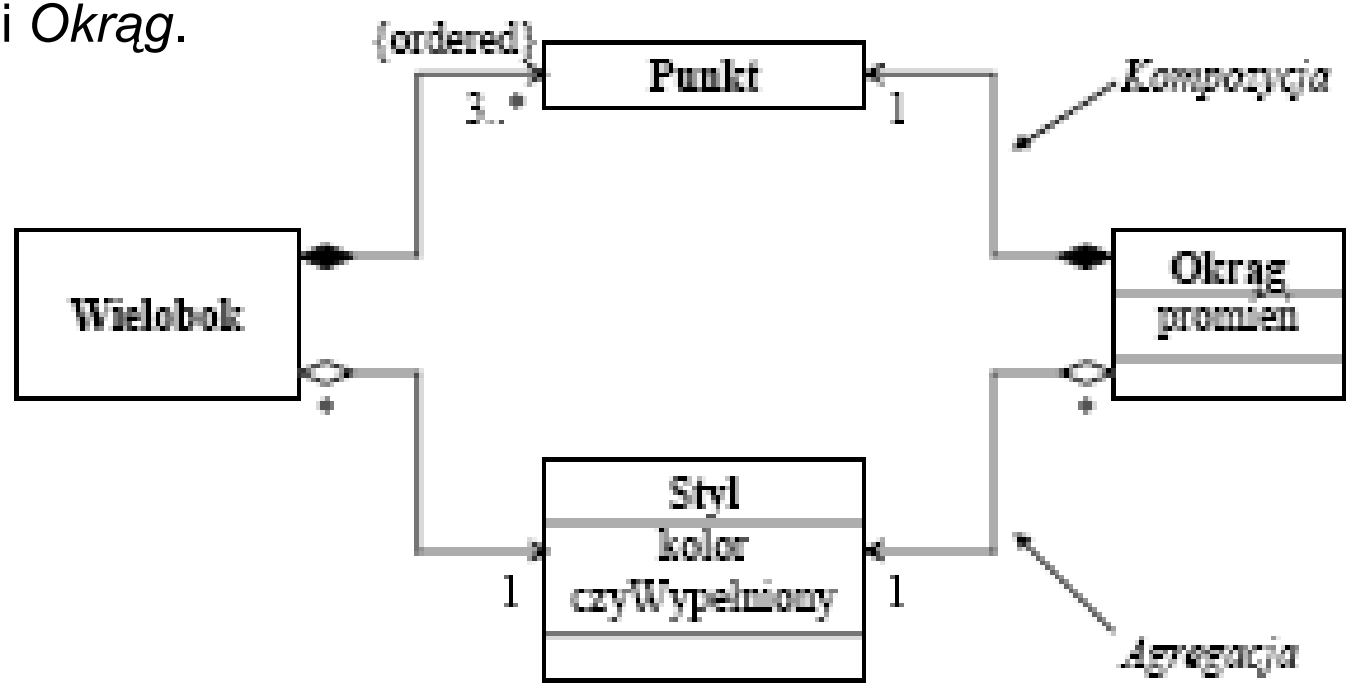
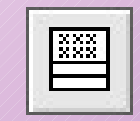
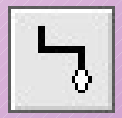


Diagram klas – notacja


Klasa


Agregacja


Asocjacja


Zależność


Dziedziczenie

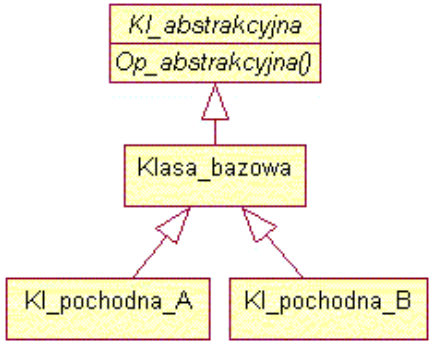
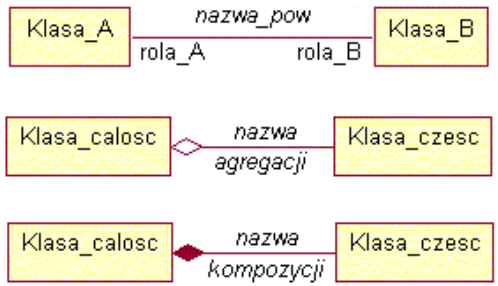
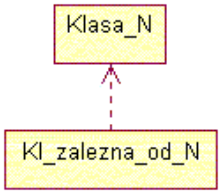
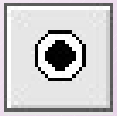
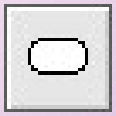
Związki pomiędzy klasami		
Uogólnienia	Powiązania	Zależności
		
Dziedziczenie klas; kl. abstrakcyjna nie może mieć bezpośrednio egzemplarzy	Związki strukturalne pomiędzy klasami; agregacja - znaczenie pojęciowe; kompozycja (agregacja całkowita) - relacja wyłącznej własności i jedność czasu życia	Związek użycia (zmiany w definicji mogą mieć wpływ na klasę zależną)

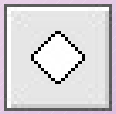
Diagram czynności

- Przypadki użycia pokazują, *co* powinien robić system. Diagramy czynności umożliwiają określenie tego, *w jaki sposób* system będzie osiągał swoje zamierzone cele.
- **Diagramy czynności** przedstawiają akcje zamodelowane na wysokim poziomie oraz połączone razem w łańcuch, reprezentujące procesy zachodzące w systemie.
- Wykonanie czynności rozpoczyna się w jej *węźle początkowym* (*initial node*) przedstawionym pod postacią wypełnionego koła. Na drugim końcu diagramu występuje *węzeł końcowy czynności* (*activity final node*) oznaczający jej koniec i przedstawiony pod postacią dwóch koncentrycznych kół, z których środkowe jest wypełnione.
- Pomiędzy węzłem początkowym a końcowym czynności występują *akcje* (*actions*) obrazowane za pomocą prostokątów o zaokrąglonych narożnikach.

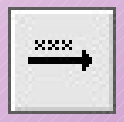
Diagram czynności - notacja

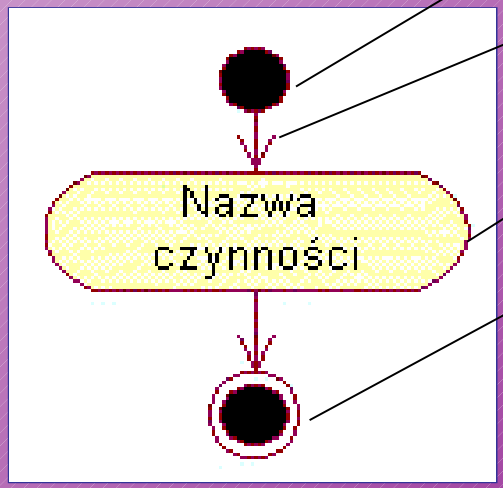

Stan początkowy/ końcowy


Czynność


Rozgałęzienie


Wykonanie współbieżne


Powiązanie między czynnościami



węzeł początkowy

przepływ

akcja

węzeł końcowy

Diagram czynności - notacja

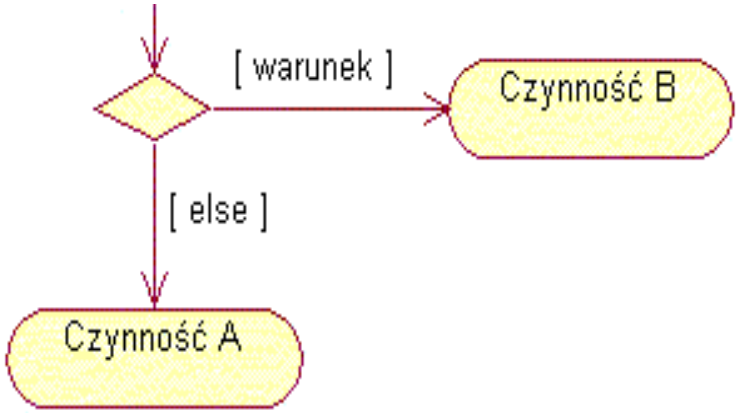
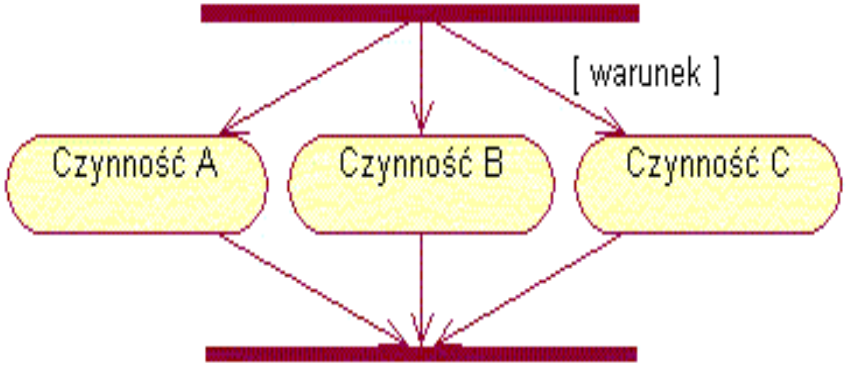
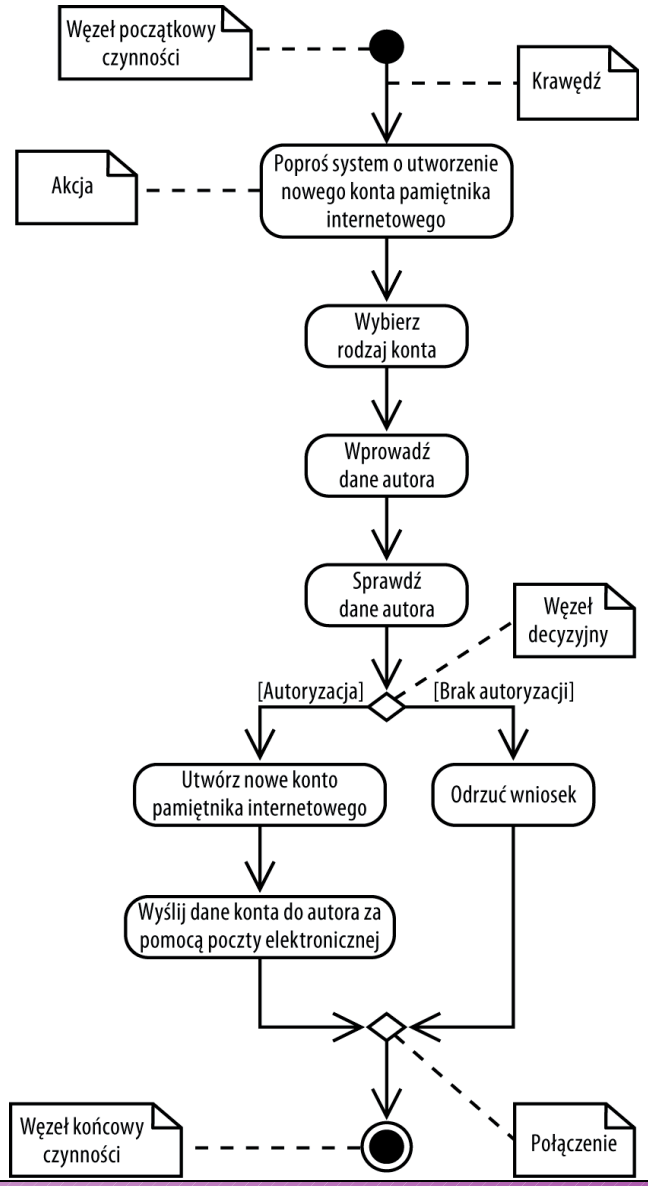
Rozgałęzienia	Rozwidlenia i scalenia
	
Opisują ścieżki alternatywne; do wyboru jednej z nich dochodzi na podstawie wyliczonych wartości warunków (wyrażeń logicznych) `else` - reprezentującego ścieżkę wybieraną, gdy wszystkie inne warunki nie są spełnione	Współbieżne wykonanie czynności; w punkcie scalenia dochodzi do synchronizacji współbieżnych przepływów sterowania. Wątek warunkowy - jeśli warunek jest fałszywy, zakłada się, że z punktu widzenia scalenia wątek ten jest juz zakończony

Diagram czynności - przykład



Na rysunku przedstawiony został proces tworzenia konta pamiętnika internetowego zapisany przy użyciu notacji diagramu czynności.

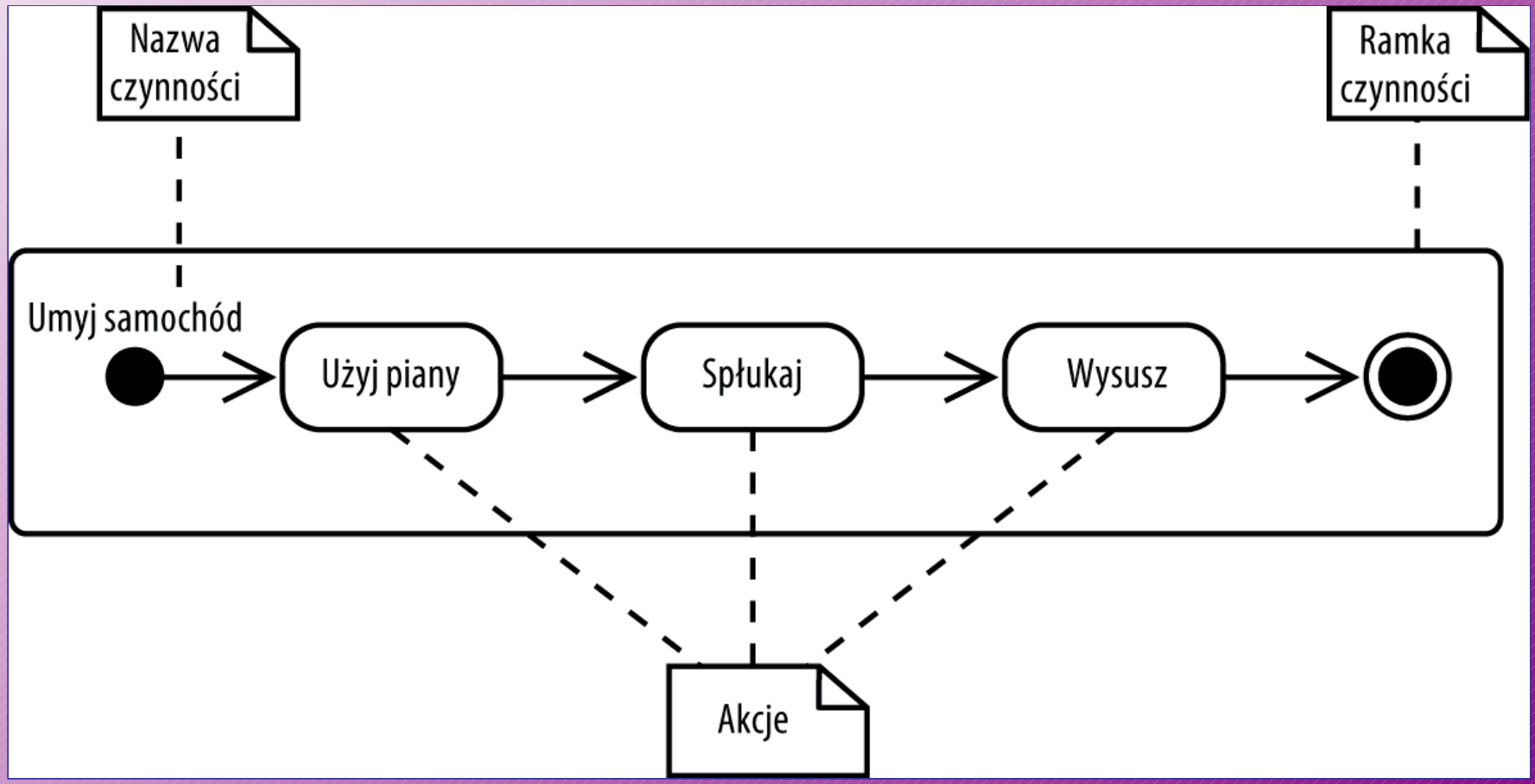
Wykonanie czynności rozpoczyna się w jej *węźle początkowym* przedstawionym pod postacią wypełnionego koła. Na drugim końcu diagramu występuje *węzeł końcowy czynności* przedstawiony pod postacią dwóch koncentrycznych kół, z których środkowe jest wypełnione.

Pomiędzy węzłem początkowym a końcowym czynności występują *akcje* obrazowane za pomocą prostokątów o zaokrąglonych narożnikach.

Przepływ czynności przedstawiony jest przy użyciu strzałek nazywanych *krawędziami* lub *ścieżkami*.

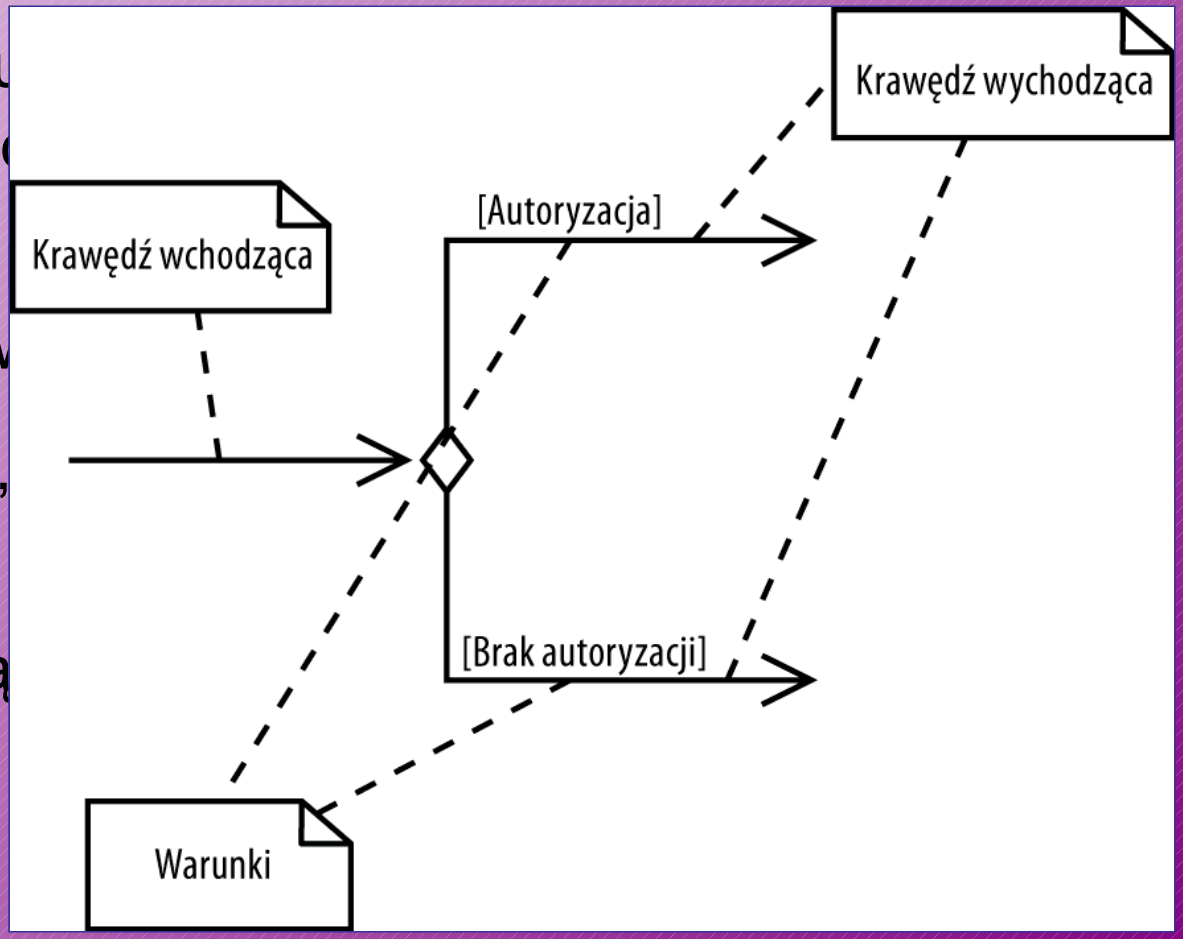
Czynności a akcje

- **Akcje** są aktywnymi krokami niezbędnymi do ukończenia procesu.



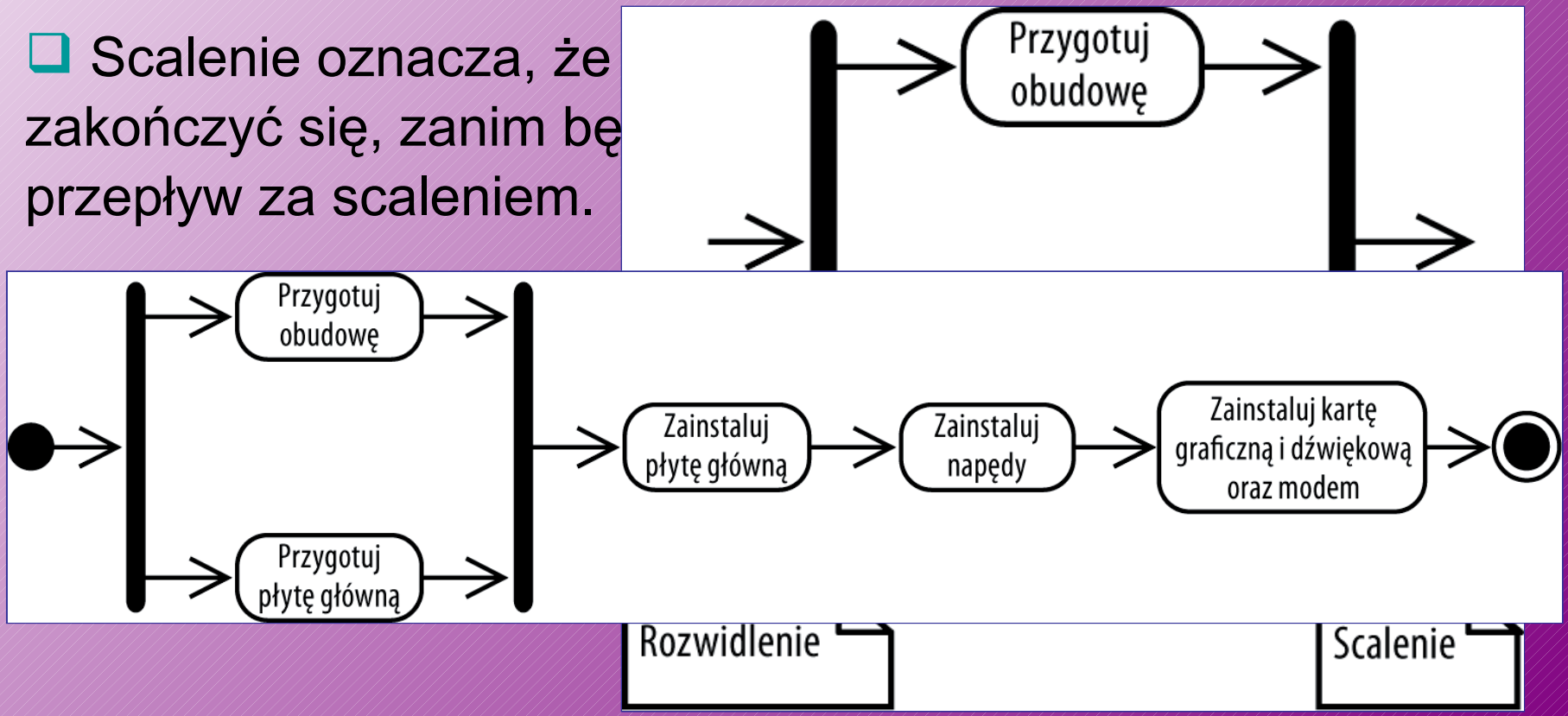
Węzły decyzyjne i połączenia

- **Węzły decyzyjne** (*decisions*) używane są w przypadku, gdy w zależności od warunku chcemy wykonać inną sekwencję akcji.
- Węzły tego rodzaju
jedną krawędzią wch
- Każda rozwidlona
condition) zapisany w
- Warunki określają,
węźle decyzyjnym.
- Warunki przyjmują



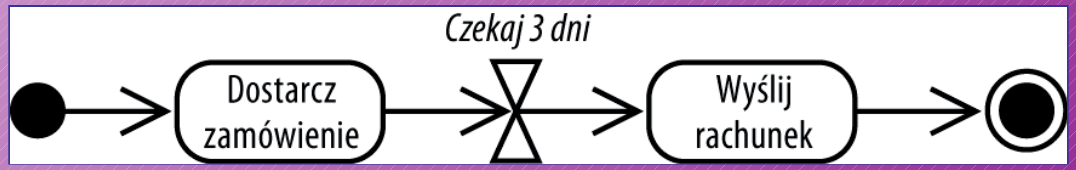
Jednoczesne wykonywanie wielu zadań

- Kroki, które zachodzą w tym samym czasie, są nazywane **współbieżnymi** (*concurrent*) lub **równoległymi (***parallel*).
- Równoległe akcje prezentowane są na diagramach przy użyciu *rozwidleń* (*forks*) oraz *scaleń* (*joins*)
- Scalenie oznacza, że zakończyć się, zanim bę przepływ za scaleniem.

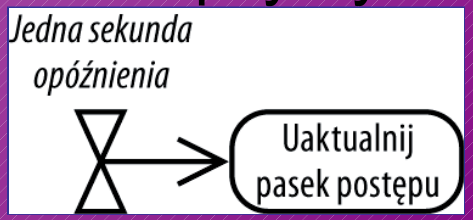


Zdarzenia czasowe

- **Zdarzenia czasowe** (*time events*) przedstawiane są przy użyciu symbolu *klepsydry*.
- Tekst umieszczony obok klepsydry określa ilość czasu, jaki musi upłynąć.
- Krawędź wchodząca do zdarzenia czasowego oznacza, że jest ono aktywowane tylko raz.

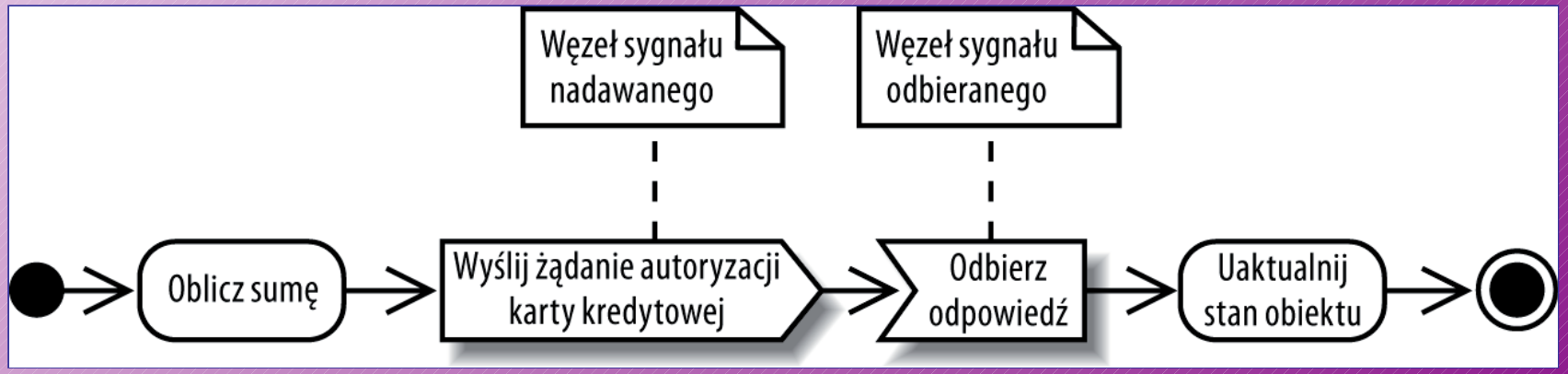


- Zdarzenie czasowe bez wchodzących przepływów jest **cykliczne** (*recurring*), co oznacza, że jest aktywowane w odstępach czasu podanych obok symbolu klepsydry.



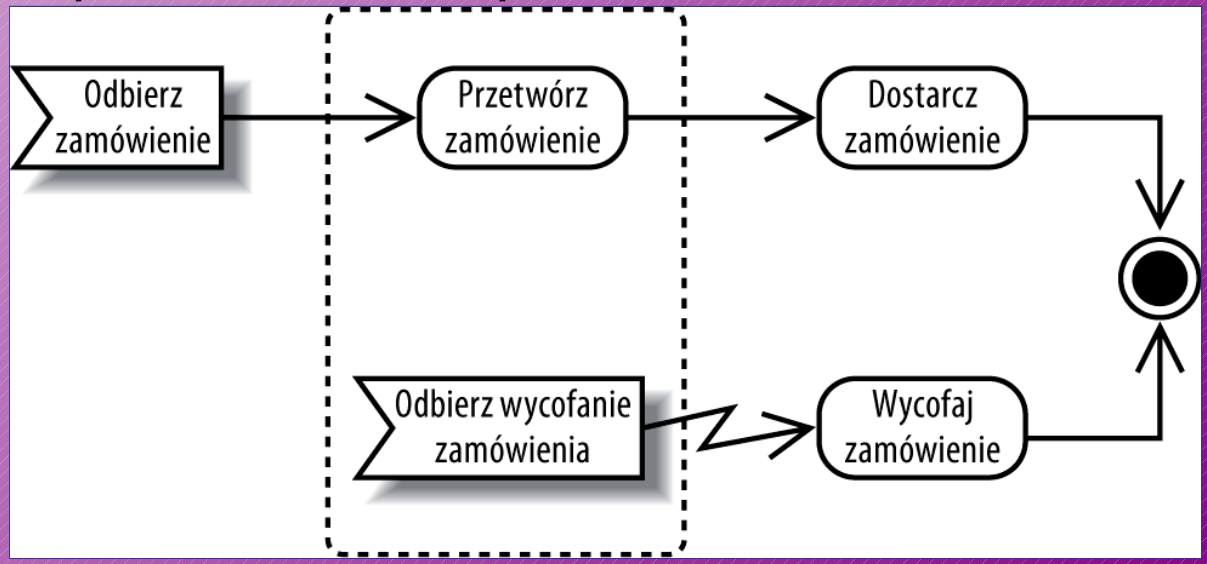
Nadawanie i odbieranie sygnałów

- Czynności mogą wymagać interakcji z zewnętrznymi osobami, systemami lub procesami.
- Na diagramach czynności **sygnały** (*signals*) reprezentują interakcje z zewnętrznymi uczestnikami.
- Sygnałami są nadawane i odbierane komunikaty.



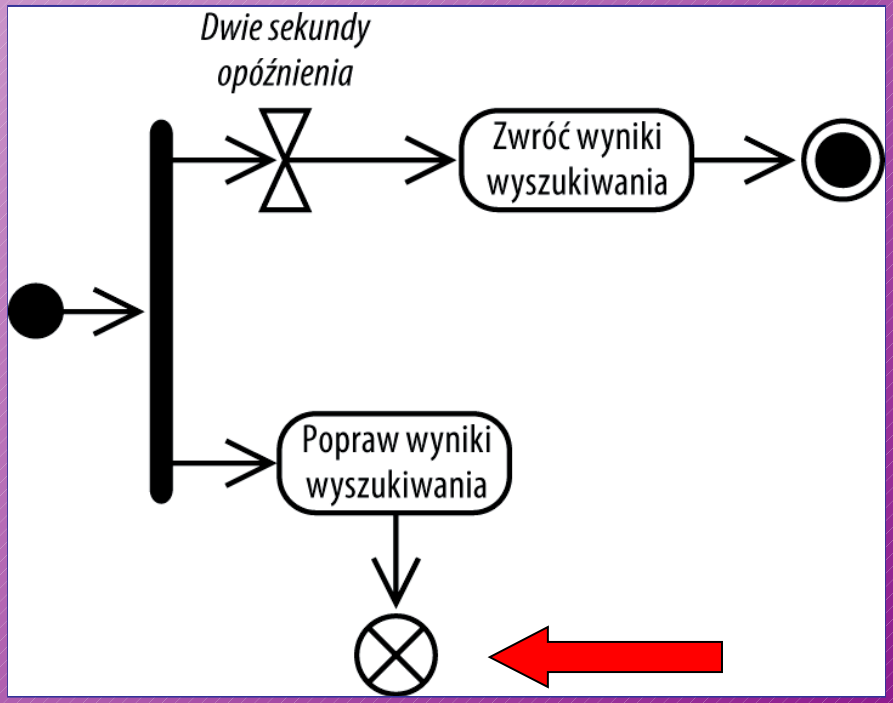
Przerwanie czynności

- Niekiedy istnieje konieczność zamodelowania procesu, który może być przerwany przez zdarzenie
- Tego rodzaju przerwania mogą być przedstawiane przy użyciu **obszarów przerwań** (*interruption regions*).
- Obszar przerwań oznacza się zaokrąglonym prostokątem otaczającego akcje, które mogą zostać przerwane, oraz zdarzenie mogące powodować przerwanie.



Kończenie przepływu

- Nową funkcją wersji UML 2.0 jest możliwość pokazania końca przepływu bez konieczności kończenia całej czynności.
- **Węzeł końcowy przepływu** (*flow final node*) kończy jedynie własną ścieżkę, a nie całą czynność.



Partycje

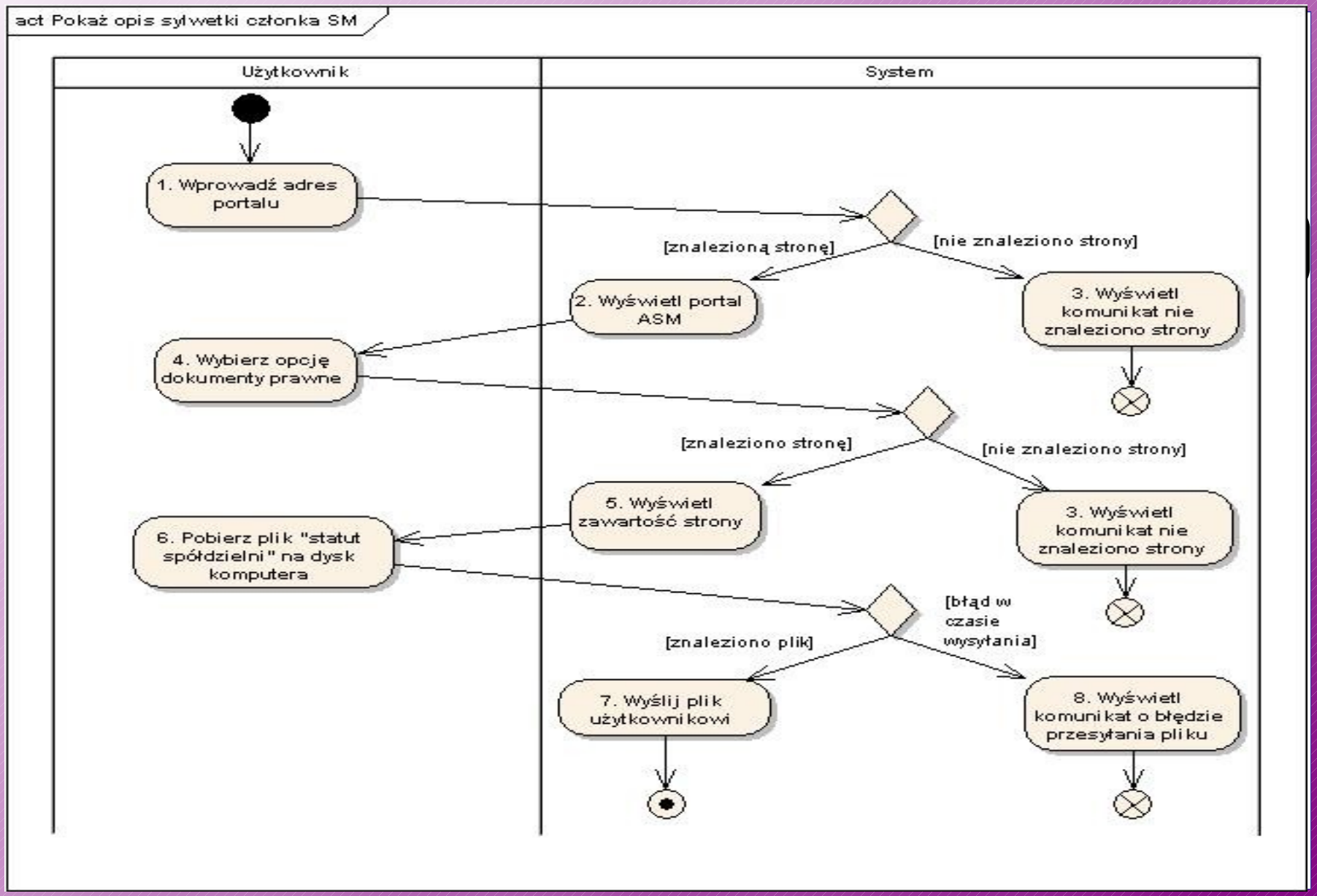
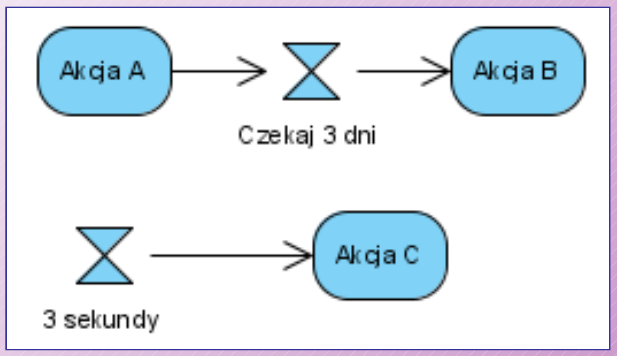
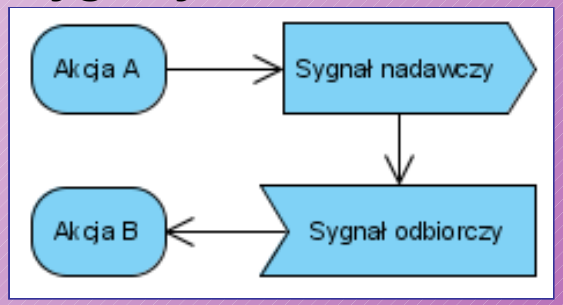

act Pokaż opis sylwetki członka SM
Użytkownik
System
1. Wprowadź adres portalu
[znalezioną stronę]
[nie znaleziono strony]
2. Wyświetl portal ASM
3. Wyświetl komunikat nie znaleziono strony
4. Wybierz opcję dokumenty prawne
[znaleziono stronę]
[nie znaleziono strony]
5. Wyświetl zawartość strony
3. Wyświetl komunikat nie znaleziono strony
6. Pobierz plik "statut spółdzielni" na dysk komputera
[znaleziono plik]
[błąd w czasie wysyłania]
7. Wyślij plik użytkownikowi
8. Wyświetl komunikat o błędzie przesyłania pliku

Diagram czynności - przykłady

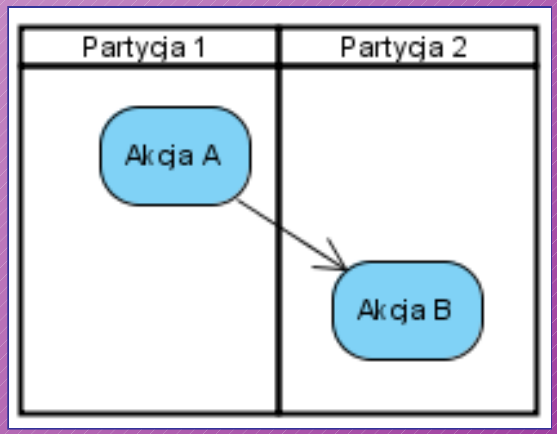
Zdarzenia czasowe



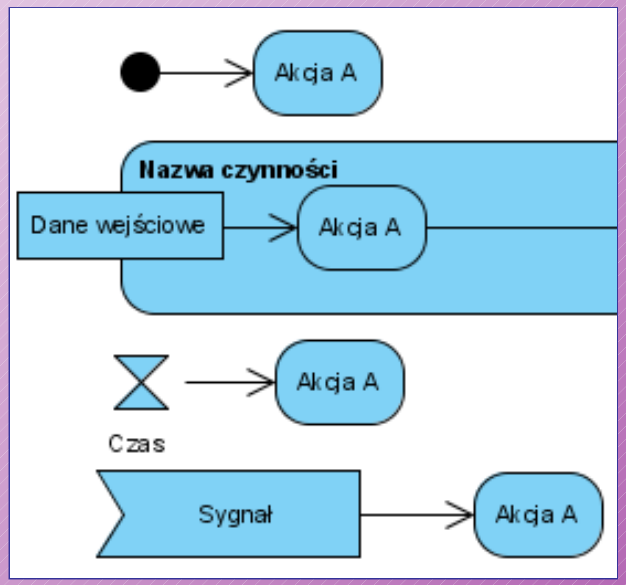
Sygnały



Partycje



Rozpoczynanie czynności



Kończenie czynności

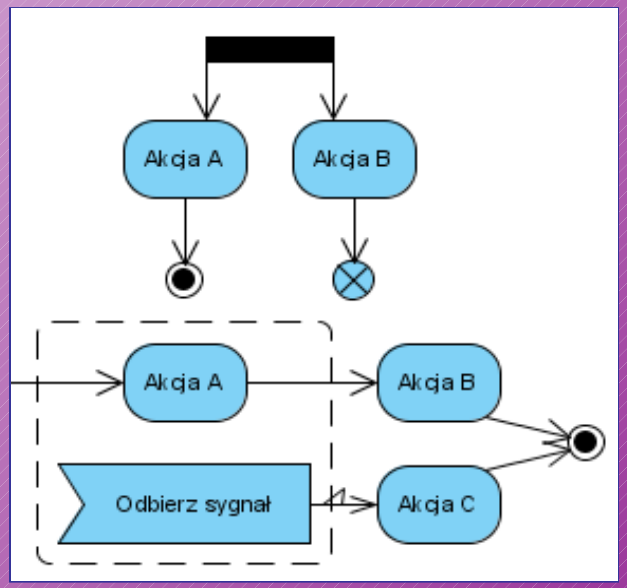


Diagram sekwencji

- **Diagramy sekwencji** posiadają dwa wymiary: wymiar pionowy reprezentuje czas, zaś wymiar poziomy reprezentuje różne obiekty.
- Generalnie, istotna jest kolejność pewnych zdarzeń, natomiast nie jest istotna rzeczywista miara czasu.
- Obiekty są zaznaczane w postaci prostokątów z wpisaną wewnątrz nazwą obiektu (klasy).
- Od każdego obiektu prowadzi linia reprezentująca „linię życia" obiektu.
- Obiekt może być aktywny lub nie - jeżeli nie jest aktywny, wówczas jego linia życia jest przedstawiona w postaci linii przerywanej; jeżeli jest aktywny, to linia życia jest wąskim, długim prostokątem

Diagram sekwencji – podstawowe elementy

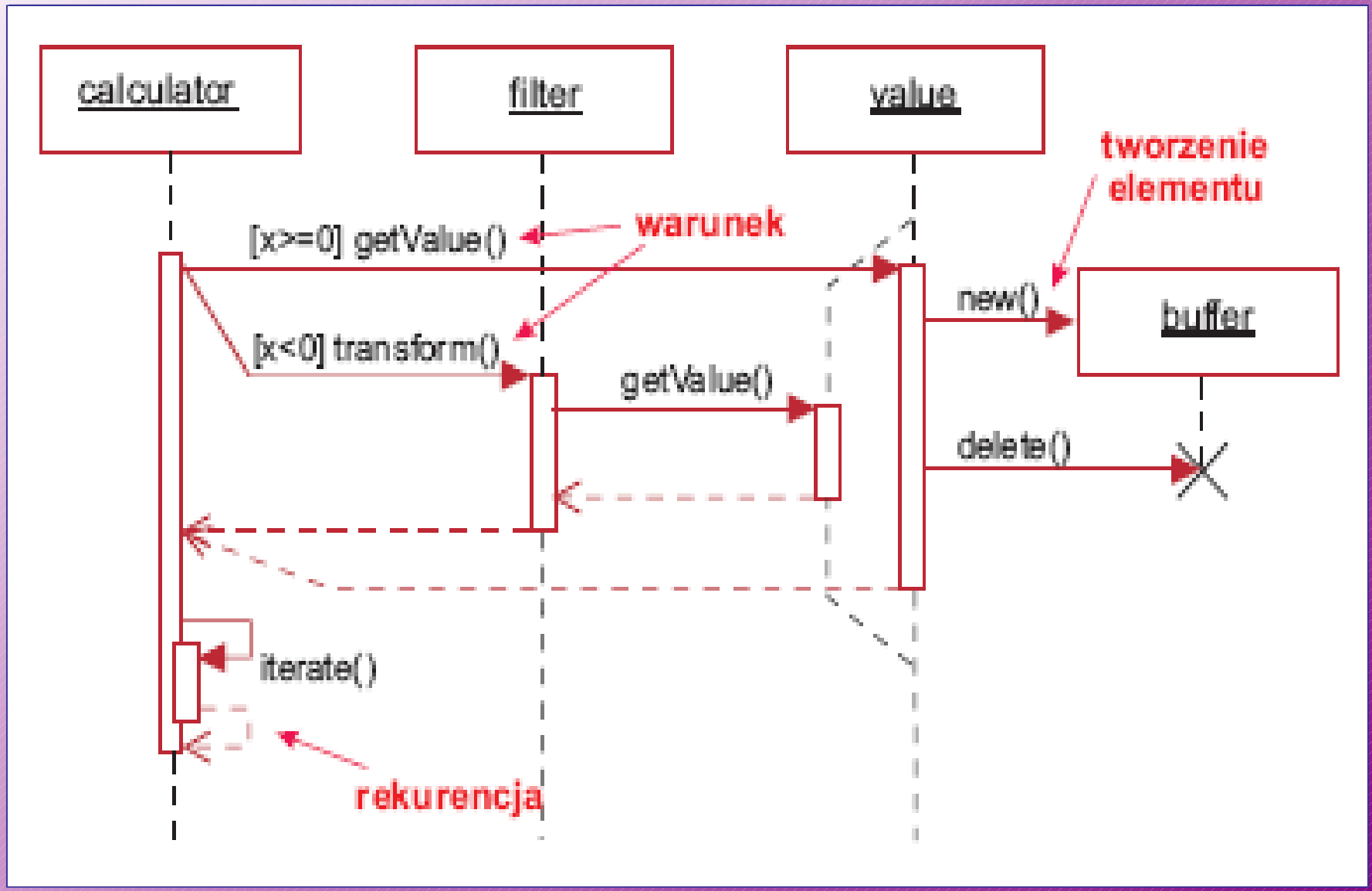
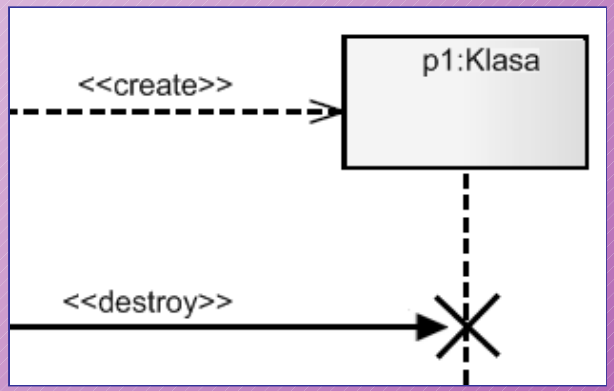


Diagram sekwencji – typy komunikatów

Składnia	Typ	Opis
	komunikat prosty	Bezpośredni przepływ sterowania z instancji do instancji
	komunikat synchroniczny	Wstrzymanie operacji do czasu uzyskania odpowiedzi
	komunikat asynchroniczny	Kontunuowanie operacji bez czekania na odpowiedź
	powrót	Powrót z wywołania podprogramu



komunikat tworzenia uczestnika

komunikat usuwania uczestnika

Diagram sekwencji – typy komunikacji

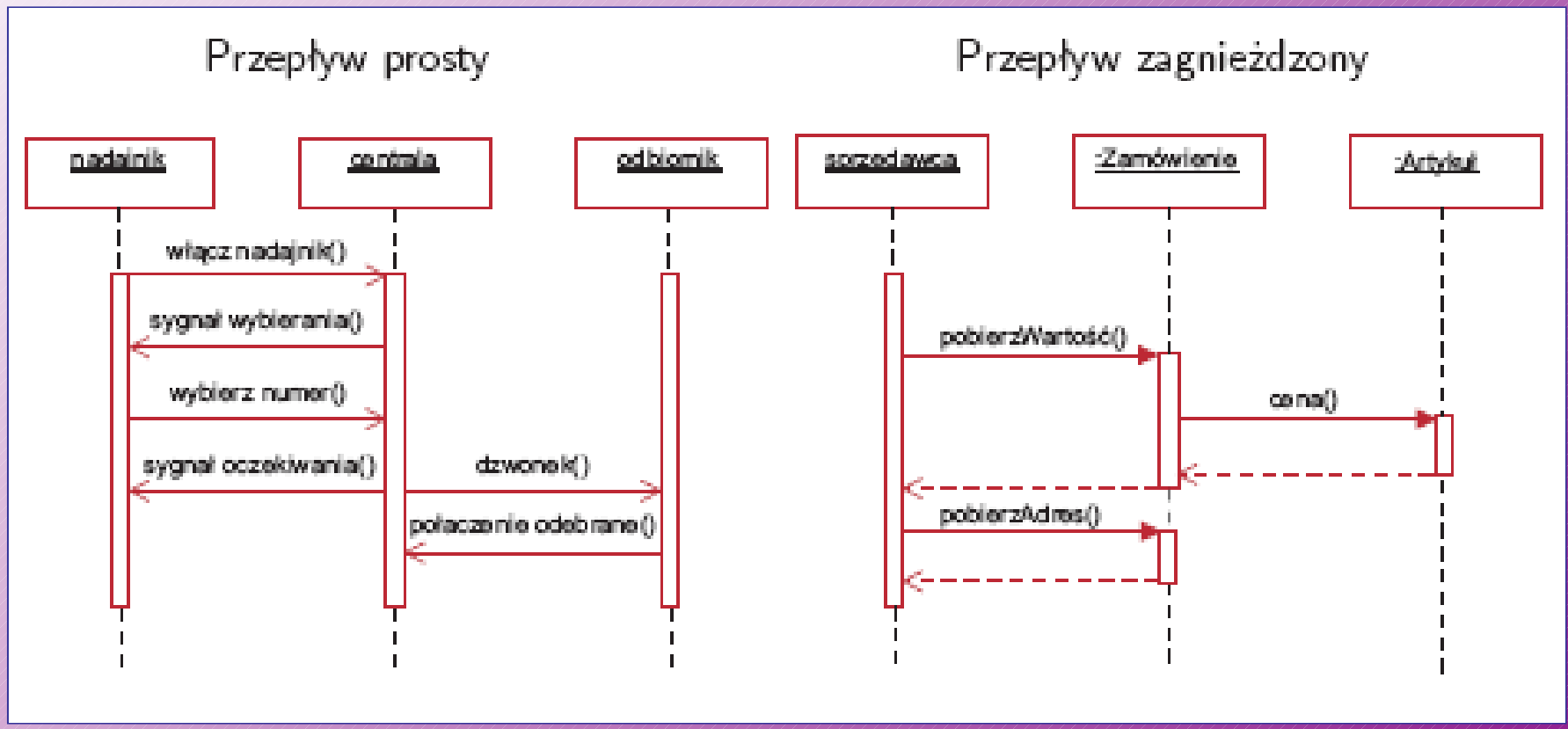


Diagram sekwencji – przykłady

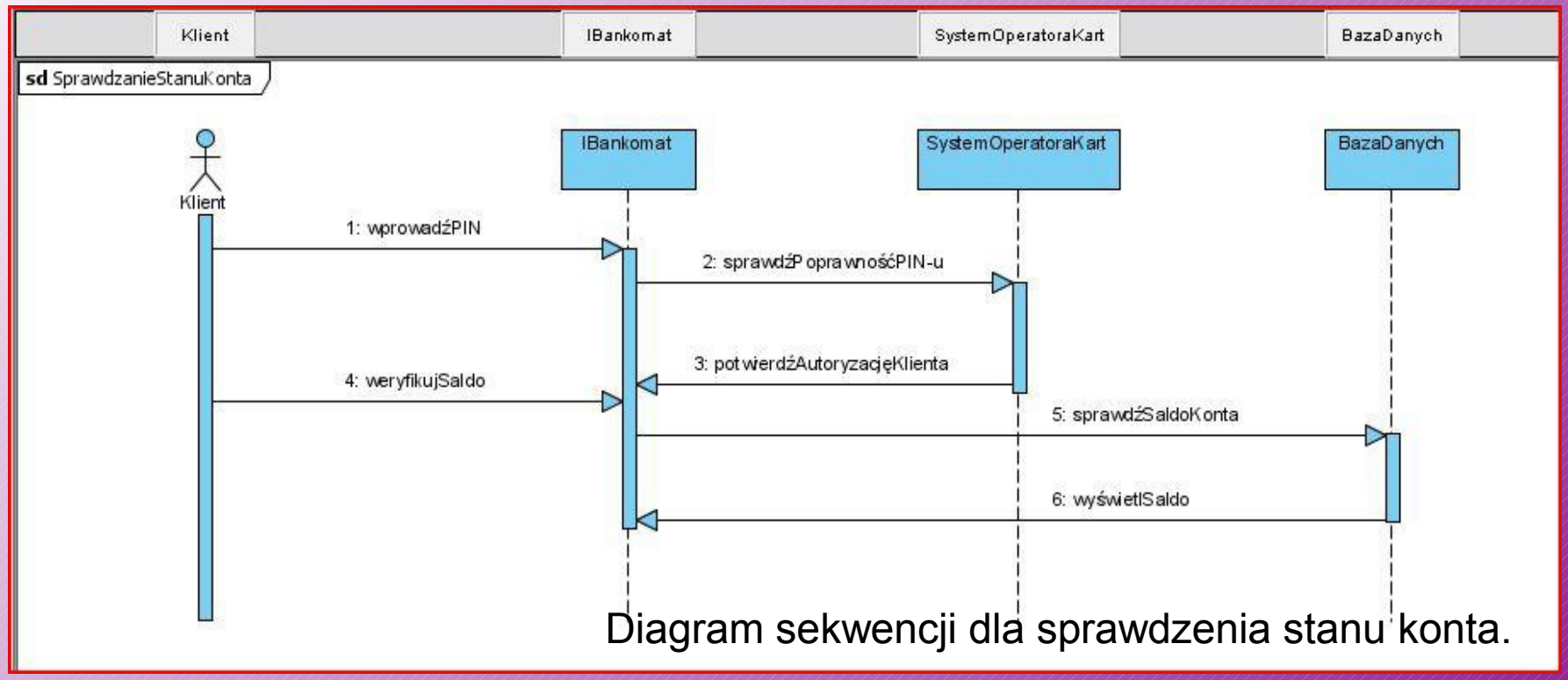


Diagram sekwencji dla sprawdzenia stanu konta.

Diagram komponentów

- **Diagramy komponentów** (component diagram) pokazują podział systemów programowych na mniejsze podsystemy.
- **Komponent** to wymienialny, wykonywalny fragment systemu, z ukrytymi szczegółami implementacyjnymi (np. plik .dll, podprogram).
- Funkcjonalność oferowana przez komponent jest dostępna przez interfejsy, które implementuje.
- Z drugiej strony, komponent może wymagać pewnych interfejsów, które muszą być dostarczone przez inne komponenty.
- Diagram komponentów służy do pokazania związków pomiędzy komponentami i interfejsami.

Diagram komponentów

- W UML zdefiniowane są następujące stereotypy komponentów:

⇨ executable określa komponent, który można wykonać na węźle;

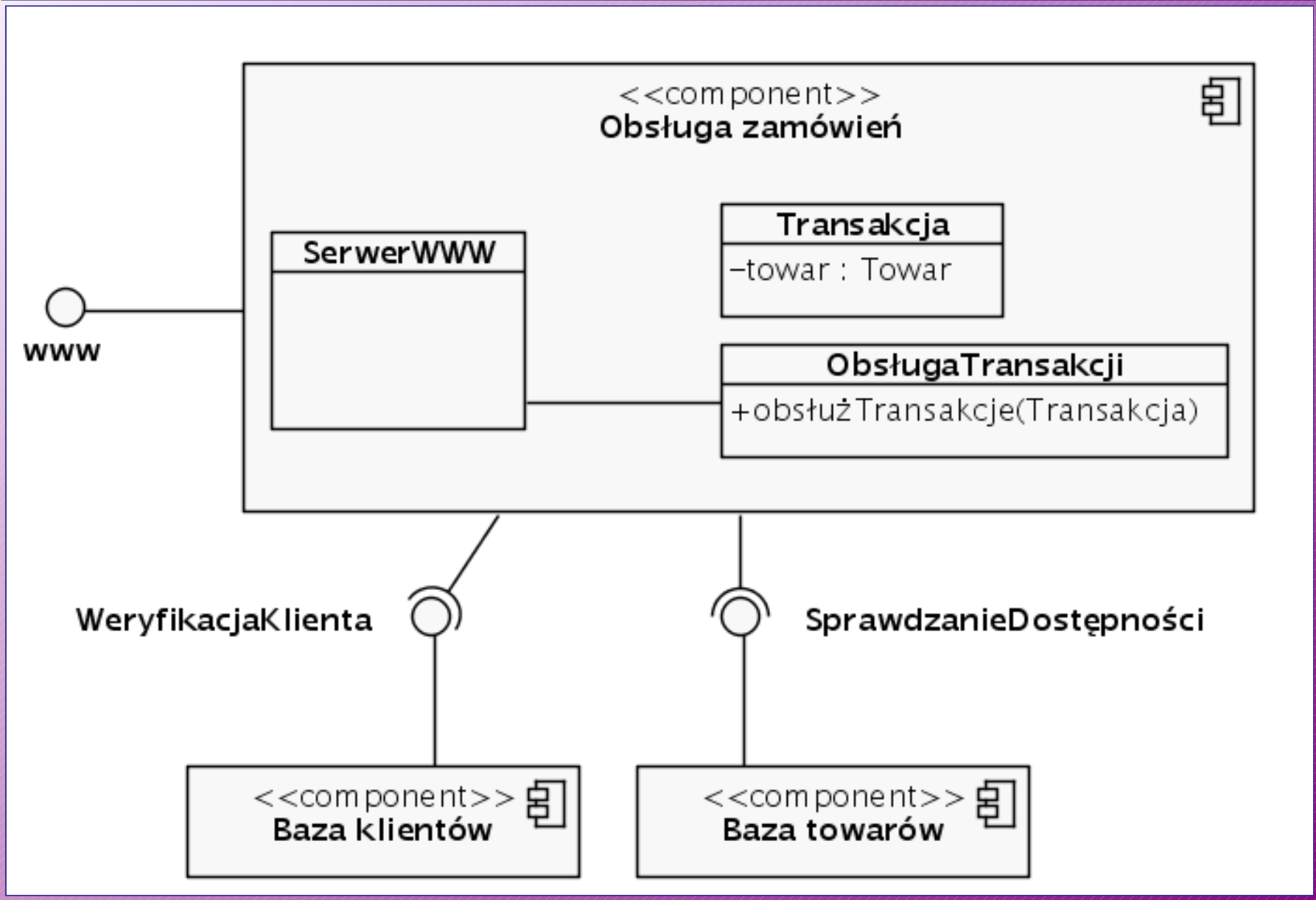
⇨ library określa dynamiczną lub statyczną bibliotekę obiektów;

⇨ table określa komponent reprezentujący tabelę bazy danych;

⇨ file określa komponent reprezentujący dokument zawierający kod źródłowy lub dane;

⇨ document określa komponent reprezentujący dokument;

Diagram komponentów - przykład



Narzędzia UML

- Narzędzia do modelowania w języku UML (*Unified Modeling Language* – *Ujednoliconym Języku Modelowania*), to oprogramowanie, które pozwala tworzyć modele pomocne przy programowaniu, ale także analizie procesów biznesowych.
- Narzędzia podzielono na wolne/otwarte i zamknięte (komercyjne).

Otwarte narzędzia UML - 1

- **Acceleo** – system generacji kodu źródłowego z modeli ML oparty na Eclipse i szablonie EMF (http://www.acceleo.org/)
- **ArgoUML** – napisany w Javie, zaawansowane generowanie kodu i podpowiedzi, ciągle rozszerzany (http://argouml.tigris.org/)
- **ATLAS Transformation Language** – narzędzie, pozwalające transformować między innymi modele UML w inne modele UML lub Java, itp. ATL jest kompletnym rozwiązaniem OpenSource (http://www.oneclipse.com/plugins/modeling/atl/view).
- **BoUml** – obejmuje UML 2.0, tworzy dokumentację HTML, nie wymaga uprawnień administratora do instalacji, dostępny na licencji GPL (http://bouml.free.fr)

Otwarte narzędzia UML - 2

- **Dia** – ogólne narzędzie do rysowania diagramów, które obsługuje modelowanie UML - licencja GNU GPL (http://www.gnome.org/projects/dia/)
- **ESS-Model** – generator diagramów projektów Delphi oraz Java (http://essmodel.sourceforge.net/)
- **Eclipse** – z platformą modelowania Eclipse (ang. Eclipse Modeling Framework, EMF) i metamodelem UML 2.0 (http://www.eclipse.org/)
- **Fujaba** – platforma developerska UML i Java; dostępna też w wersji Eclipse (http://www.fujaba.de/)
- **Gaphor** – środowisko modelowania UML 2.0 oparte na GTK+/GNOME napisane w języku Python (http://gaphor.sourceforge.net/)

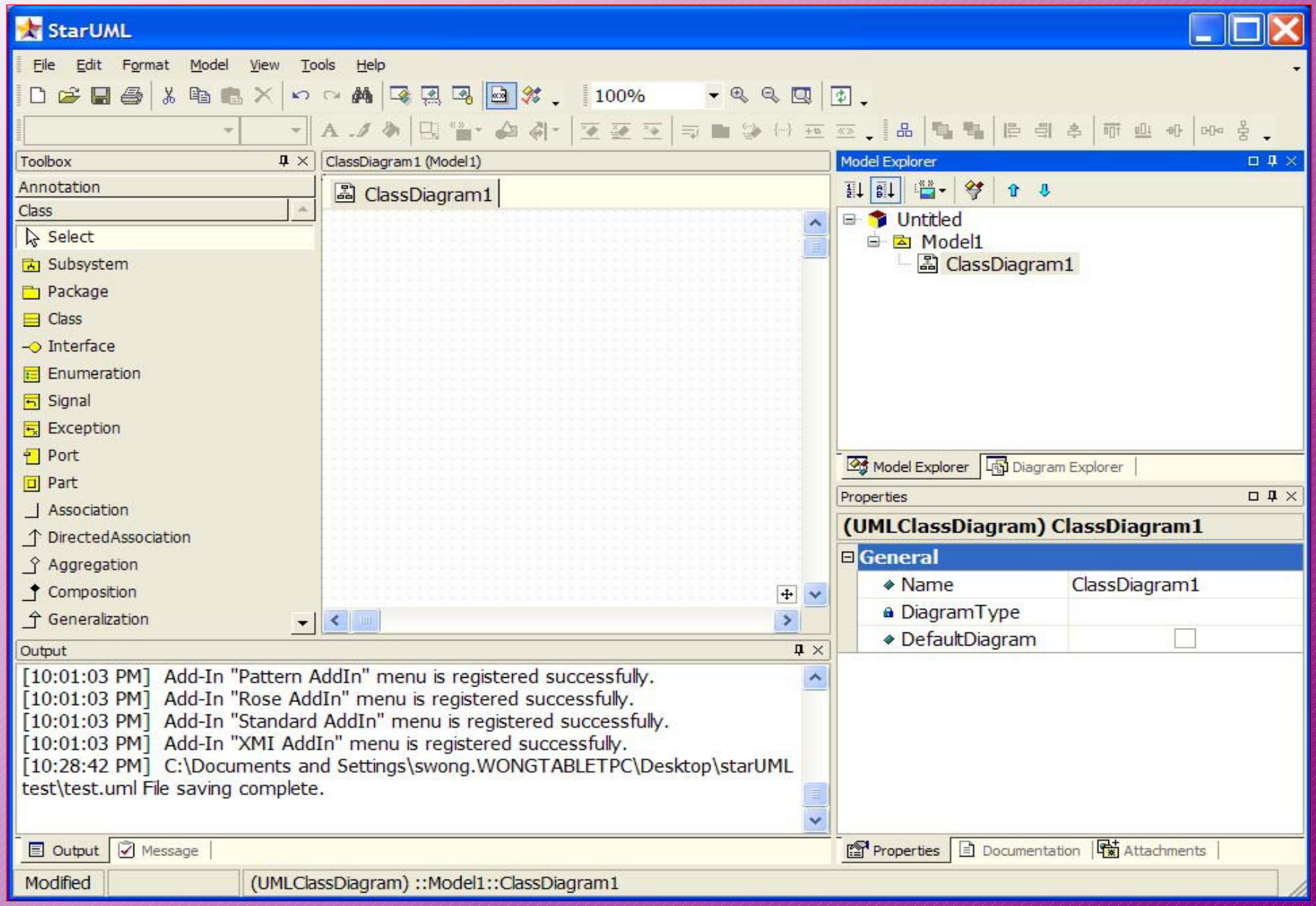
Otwarte narzędzia UML - 3

- **MetaUML** – notacja tekstowa dla UML. Renderowanie diagramów w oparciu o MetaPost, odpowiednie dla systemu składu LaTeX (http://metauml.sourceforge.net/)
- **MonoUML** – bazujące na najnowszym oprogramowaniu Mono, GTK+ i ExpertCoder.
- **NetBeans** – z "NetBeans IDE 5.5 Enterprise Pack" oraz z NetBeans IDE >= 6.0 (http://www.netbeans.org/)
- **StarUML** – (Obsługuje: C/C++, Java, Visual Basic, Delphi, JScript, VBScript, C#, VB.NET) platforma UML/MDA dla systemu Windows (2000, XP), dostępna na zmodyfikowanej licencji GNU GPL, napisana głównie w Delphi (http://staruml.tigris.org/)
- **Umbrello** – program dla Linuksa, część KDE (http://uml.sourceforge.net/index.php)

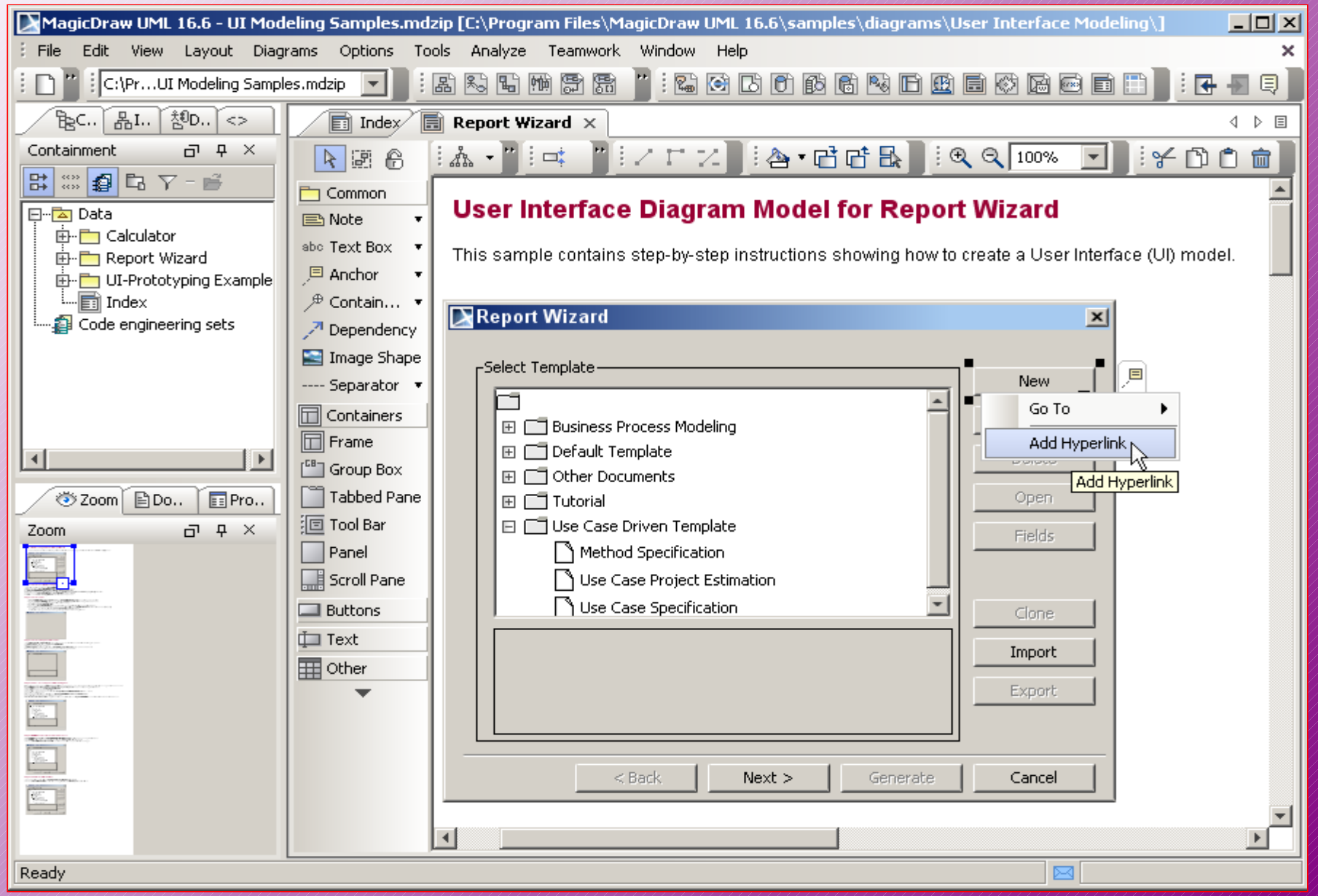
Otwarte narzędzia UML - 4

- **UMLet** – łatwe w użyciu narzędzie pozwalające tworzyć diagramy UML, stworzone w Javie, (licencja GNU GPL) (http://www.umlet.com/)
- **UML Sculptor** – prosty, łatwy w użyciu program do tworzenia diagramów klas (http://umlsculptor.sourceforge.net/)
- **Visual Paradigm SDE Community Edition -** śodowisko programistyczne (SDE) Visual Paradigm integruje się ze wszystkimi wiodącymi IDE (Visual Studio®, Eclipse/WebSphere®, Borland JBuilder®, NetBeans/Sun™ ONE, IntelliJ IDEA™, Oracle JDeveloper, BEA WebLogic Workshop™), do użytku niekomercyjnego (do nauki), dostępne za darmo (http://www.visualparadigm.com/product/sde/ec/)

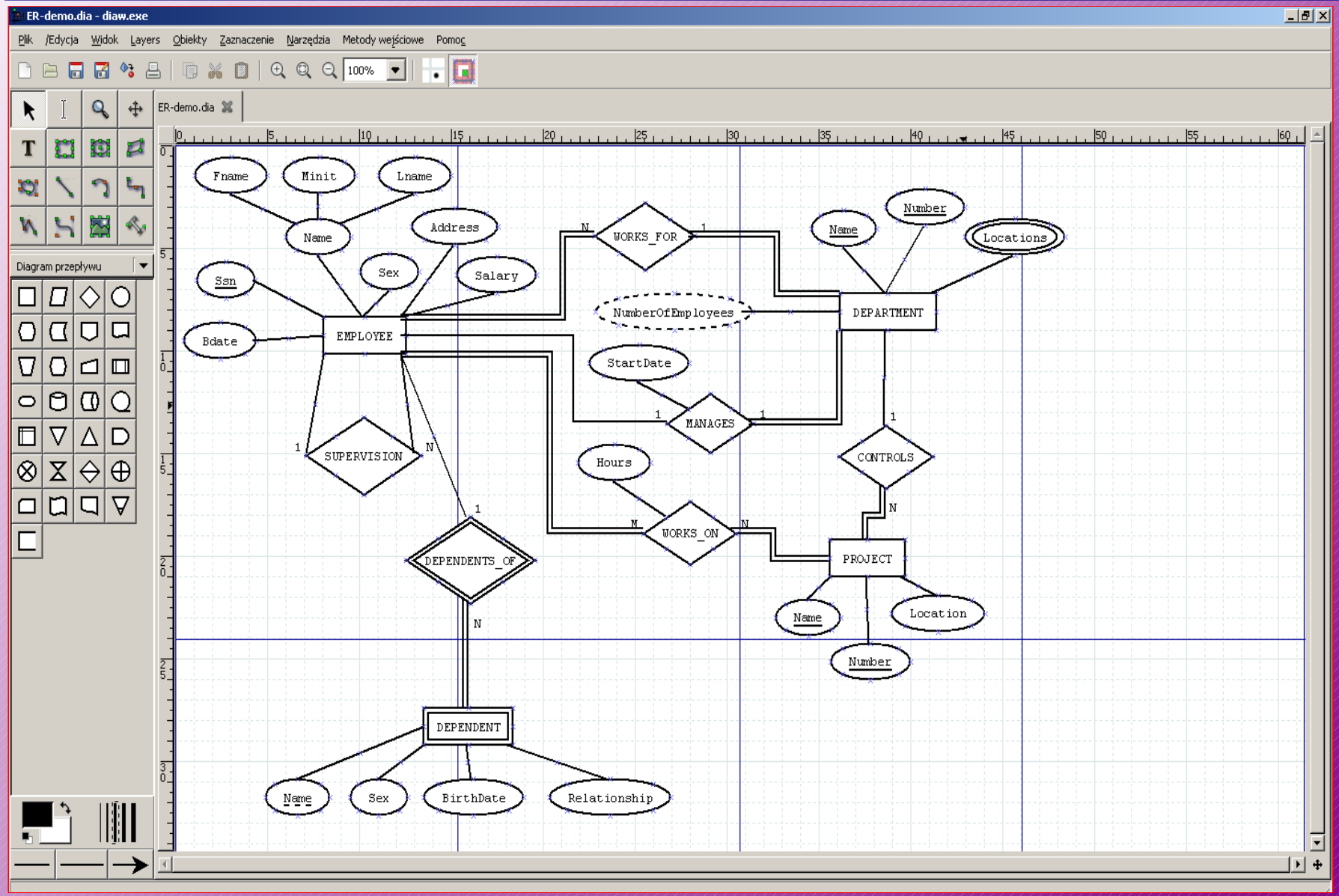
Otwarte narzędzia UML - StarUML



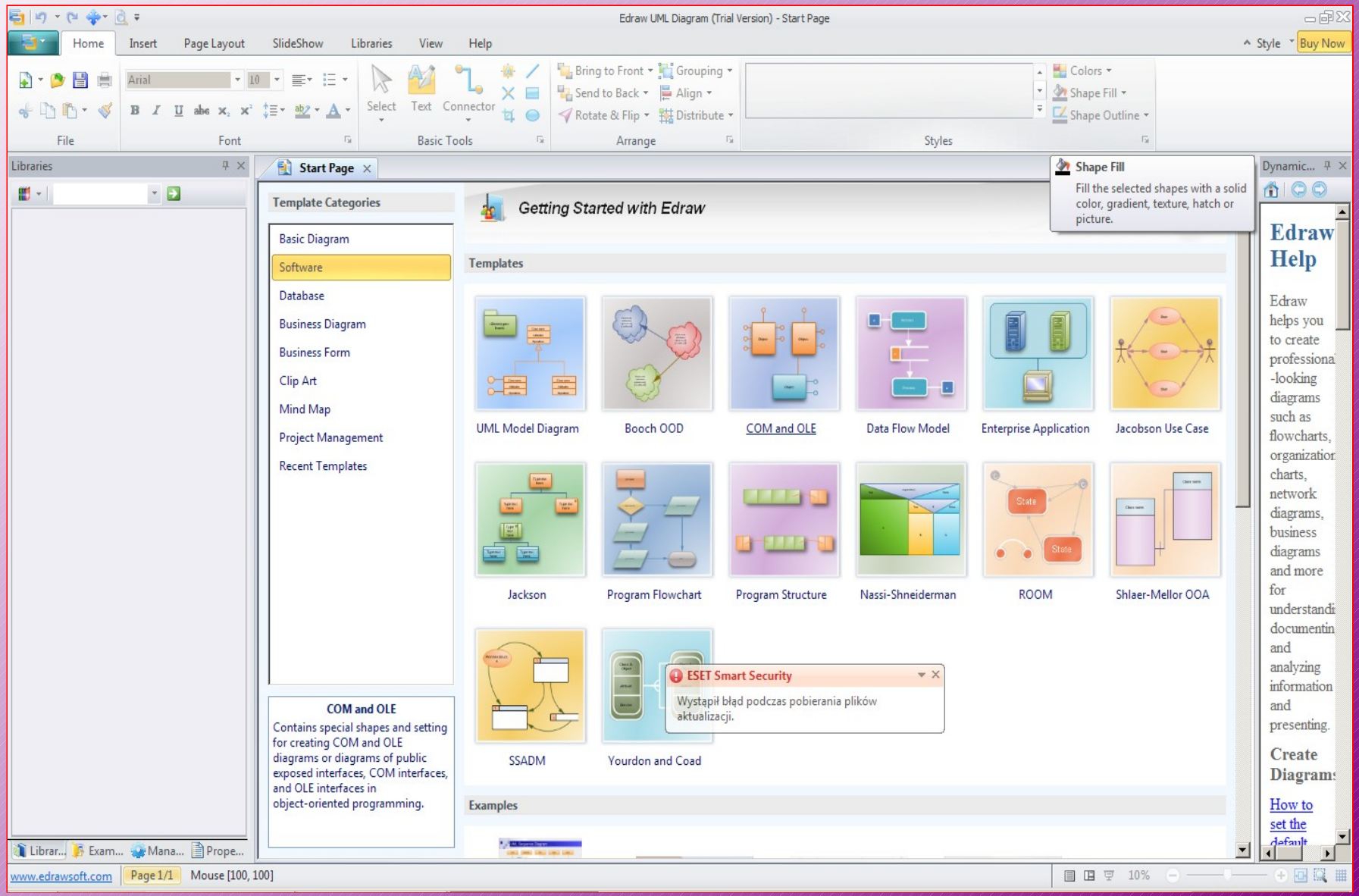
Otwarte narzędzia UML – MagicDraw UML



Otwarte narzędzia UML – Dia



Otwarte narzędzia UML – Edraw



Otwarte narzędzia UML – ArgoUML