

**PROJEKTOWANIE  
OBIEKTOWE SYSTEMÓW  
INFORMATYCZNYCH**

**Część II**

Wanda Gryglewicz-Kacerka  
Agnieszka Duraj

Włocławek 2013

REDAKCJA WYDAWNICTWA  
PAŃSTWOWEJ WYŻSZEJ SZKOŁY ZAWODOWEJ  
WE WŁOCŁAWKU

Projektowanie obiektowe systemów informatycznych. Część II

REDAKTOR NACZELNY  
dr Ernest Kuczyński

RECENZENT  
prof. dr inż. Włodzimierz Jemec  
wladzimierz.jemec@pwsz.wloclawek.pl

KOREKTA  
Adam A. Korzus.

©Copyright by Państwowa Wyższa Szkoła Zawodowa we Włocławku,  
Włocławek 2013

ISBN 978-83-60607-45-9

SKŁAD I DRUK  
Partner Poligrafia  
Białystok, ul. Zwycięstwa 10  
www.partnerpoligrafia.pl  
tel./fax: 85 653-78-04

# SPIS TREŚCI

<b>1. POJĘCIA PODSTAWOWE.....</b>	<b>4</b>
OBIEKT .....	4
METODY .....	4
ABSTRAKCJA .....	5
HERMETYZACJA .....	5
POLIMORFIZM .....	5
DZIEDZICZENIE.....	6
<b>2. PROCES PROJEKTOWANIA SYSTEMU INFORMATYCZNEGO .</b>	<b>8</b>
<b>3. CHARAKTERYSTYKA PROCESU ANALIZY SYSTEMOWEJ.....</b>	<b>10</b>
<b>4. OBIEKTOWE METODY ANALIZY SYSTEMU.....</b>	<b>13</b>
METODA JACOBSONA .....	18
METODOLOGIA ANALIZY SYSTEMÓW LYNXA.....	23
<b>5. NARZĘDZIA MODELOWANIA W ANALIZIE SYSTEMU .....</b>	<b>25</b>
MODELE OPISU SYSTEMU .....	25
<b>6. JĘZYK MODELOWANIA UML.....</b>	<b>28</b>
KLASA .....	29
OBIEKTY (EGZEMPLARZE KLAS).....	31
PRZYPADEK UŻYCIA.....	33
KOOPERACJA .....	33
STAN OBIEKTU .....	33
PRZEJŚCIA Z JEDNEGO DO DRUGIEGO STANU .....	34
KOMUNIKAT.....	34
NOTATKA.....	35
WĘZEL.....	35
ZWIĄZKI.....	36
POWIĄZANIE KLAS.....	36
AGREGACJA KLAS.....	37
WIĄZANIE OBIEKTÓW KLAS .....	37
RODZAJE ZALEŻNOŚCI POMIĘDZY KLASAMI.....	38
UOGÓLNIENIE .....	39
REALIZACJA TYPU KLASA – KOMPONENT .....	40
<b>7. DIAGRAMY ANALIZY SYSTEMÓW .....</b>	<b>41</b>
DIAGRAM OBIEKTÓW .....	41
DIAGRAM KLAS.....	41
DIAGRAM PRZYPADKÓW UŻYCIA .....	43
ZWIĄZEK.....	45

DIAGRAM CZYNNOŚCI (AKTYWNOŚCI).....	48
DIAGRAM INTERAKCJI.....	49
DIAGRAM SEKWENCJI.....	51
DIAGRAM WSPÓLPRACY (KOLABORACJI).....	51
DIAGRAM STANÓW.....	56
DIAGRAM KOMPONENTÓW.....	59
DIAGRAM WDROŻENIA.....	61
<b>8. SŁOWNIK DANYCH (<i>DATA DICTIONARY</i>).....</b>	<b>62</b>
<b>9. LITERATURA.....</b>	<b>63</b>

# 1. POJĘCIA PODSTAWOWE

Projektowanie obiektowe (*object-oriented design*) to ogół metod programowania z zastosowaniem metodologii obiektowej. Programowanie obiektowe obejmuje również różne sposoby rozwiązywania problemów.

Programowanie obiektowe to pewien wzorzec programowania (paradygmat) przyjęty w danym okresie rozwoju informatyki. Paradygmat programowania definiuje sposób patrzenia programisty na przepływ sterowania i wykonywanie programu komputerowego. Traktowany jest jako zbiór współpracujących ze sobą obiektów, podczas gdy w programowaniu strukturalnym definiowane jest, co trzeba wykonać, a nie w jaki sposób. W programowaniu obiektowym programy definiuje się za pomocą obiektów określających dane rzeczywiste i funkcji na nich operujących. Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się między sobą w celu wykonywania określonych zadań.

## Obiekt

Pojęcie „obiekt” w ujęciu metod opisu otaczającego nas świata jest bardzo pojemne. Obejmuje ono zarówno takie określenia, jak: pies, koń, samochód, ołówek, wazon, kwiat, jak i takie, które są zupełnie czymś innym: światło, czas, rozmowa, kopnięcie piłki czy dokonanie wpłaty w okienku bankowym. Jednym słowem: wszystko, co nas otacza, jest w ujęciu projektu informatycznego obiektem. Zarówno rzeczy namacalne, jak i te, które są ulotne, a także te, które nie mają jakiegokolwiek formy fizycznej. Każdy obiekt ma pewne dane opisujące go. Takie dane nazywamy atrybutami obiektu (np. kolor, wysokość, wartość, czas trwania i wiele innych).

## Metody

Z pojęciem „obiekt” ściśle wiąże się pojęcie „metoda” („metody”). Najprościej mówiąc, metody mówią nam, jak możemy danego obiektu użyć lub jak możemy wpłynąć na jego zachowanie. Sam obiekt jako taki nie ma żadnego znaczenia. Obiekt bez metod można porównać do jakiejś liczby, co do której nie wiemy, co oznacza. Samo to, że jest to np. cyfra 4, nic nam nie mówi. Natomiast jeśli wiemy, że oznacza ona pojemność

cieczy wyrażoną w litrach, to wiemy, jak jej użyć (wiemy np., ile to coś waży i jaką ma objętość). Metody są ściśle przypisane do konkretnego typu obiektu i możemy ich użyć tylko i wyłącznie do obiektów danego typu. Jeśli np. obiekt „samochód” ma metodę „Uruchom silnik”, która uruchamia silnik samochodu, to nie możemy jej użyć w stosunku do obiektu innego typu, np. do obiektu typu „zwierzę”.

## **Abstrakcja**

Każdy obiekt w systemie służy jako model abstrakcyjnego „wykonawcy”, który może wykonywać pracę, opisywać i zmieniać swój stan oraz komunikować się z innymi obiektami w systemie bez pokazywania, w jaki sposób zaimplementowano dane cechy. Procesy, funkcje lub metody mogą być również abstrahowane, a kiedy tak się dzieje, konieczne są rozmaite techniki rozszerzania abstrakcji.

## **Hermetyzacja**

Hermetyzacja to ukrywanie (implementacja, enkapsulacja). Hermetyzacja zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Do zmiany jego stanu uprawnione są tylko własne metody obiektu. Każdy typ obiektu prezentuje innym obiektom swój interfejs, który określa dopuszczalne metody współpracy. Interfejs jest definicją abstrakcyjnego typu mającego jedynie operacje, a nie dane. Interfejs określa udostępniane operacje, nie zawiera natomiast ich implementacji i danych. Z tego powodu klasy mogą implementować wiele interfejsów, bez problemów wynikających z wielokrotnego dziedziczenia. (Wszystkie metody w interfejsie z reguły muszą być publiczne).

## **Polimorfizm**

Polimorfizm w dosłownym znaczeniu oznacza wielopostaciowość. Oznacza to, że dany obiekt może się zachowywać inaczej niż jego potomek lub przodek. Inaczej mówiąc, w wyniku użycia pewnej metody w stosunku do danego obiektu możemy otrzymać coś innego niż przy użyciu tej samej metody w stosunku do jego potomka (jeśli ten odziedziczył taką metodę). Np. jeśli obiekt „rodzic” ma metodę „Czytaj książkę”, w jego wyniku obiekt czyta książkę w języku polskim. Jednak użycie tej samej metody

do obiektu potomnego „córka” może dawać wynik w postaci czytania książki w języku angielskim.

Referencja to wartość, która zawiera informacje o położeniu innej wartości w pamięci. Mechanizm referencji jest powszechnie wykorzystywany w językach programowania. Jego działanie polega na uniemożliwieniu wykonywania operacji uznawanych za potencjalnie niebezpieczne, które w wypadku błędu programistycznego mogłyby doprowadzić do awarii. Referencje zwiększają w ten sposób niezawodność oprogramowania. Przy tym nie ograniczają możliwości programisty – wszystkie operacje, które nie bazują na jawnej znajomości organizacji pamięci, mogą być zaimplementowane wyłącznie za ich pomocą.

Referencje i kolekcje obiektów mogą dotyczyć obiektów różnego typu, a wywołanie metody dla referencji spowoduje zachowanie odpowiednie dla pełnego typu obiektu wywoływanego.

## **Dziedziczenie**

Dziedziczenie porządkuje i wspomaga polimorfizm i inkapsulację dzięki umożliwieniu definiowania i tworzenia specjalizowanych obiektów na podstawie bardziej ogólnych. Dla obiektów specjalizowanych nie trzeba redefiniować całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy. W typowym przypadku powstają grupy obiektów zwane klasami oraz grupy klas zwane drzewami. Odzwierciedlają one wspólne cechy obiektów.

Każdy obiekt może (choć nie musi) mieć przodka, od którego się wywodzi. Np. każdy człowiek ma swojego przodka w postaci rodzica. W zależności od przyjętej metodologii obiekt może mieć jednego lub wielu przodków. O ile może istnieć ograniczenie w postaci jednego przodka, o tyle takiego ograniczenia nie ma co do liczby potomków, których dany obiekt jest przodkiem. Fakt posiadania przodka wiąże się ściśle z dziedziczeniem. Dziecko jako obiekt może dziedziczyć po swoich przodkach takie atrybuty, jak kolor oczu, wzrost itp.

Obiekt może oprócz dziedziczenia atrybutów odziedziczyć metody, czyli – analogicznie – w naszym przykładzie dziecko może po swoich przodkach odziedziczyć takie „metody” (zachowania), jak skłonność do palenia papierosów, talent w wybranych dziedzinach życia, wczesne wstawanie itp.

## 2. PROCES PROJEKTOWANIA SYSTEMU INFORMATYCZNEGO

Proces projektowania systemu informatycznego obejmuje następujące główne etapy:

- programowanie (tworzenie kodu programu w wybranym języku programowania),
- opracowywanie dokumentacji (opisywanie wszystkich elementów składowych systemu, zależności zachodzących między nimi, a także opracowywanie szczegółowej instrukcji użytkownika),
- wdrażanie systemu do pracy (jest to proces odpowiedzialny, złożony i długotrwały obejmujący: instalację systemu w miejscu przeznaczenia, uruchamianie, testowanie oraz szkolenie personelu obsługującego system).

Każdy z etapów procesu projektowania systemu informatycznego wykonuje określone zadanie. Proces projektowania systemu informatycznego przebiega zwykle na podstawie przyjętego schematu nazwanego Cyklem Życia Systemu (*System Development Life Cycle*).

Ten schemat to opis sposobu organizacji prac projektowych w całym okresie projektowania i eksploatacji systemu, od momentu zgłoszenia przez użytkownika potrzeby istnienia systemu aż do jego wycofania z eksploatacji. Proces projektowania wykorzystuje wybraną metodologię. Jest to nauka, która obejmuje zbiór metod, technik i zasad wykorzystywanych w procesie realizacji cyklu życia systemu.

Zespół metod organizacji prac projektowych prowadzących do otrzymania gotowego systemu informatycznego to Model Cyklu Życia systemu.

Projektowanie systemu informatycznego wymaga określenia:

- modelu danych (*Data Model*),
- interfejsu użytkownika (*User Interface, Front-End*).

Model danych określa zbiór ogólnych zasad posługiwania się danymi. Wyróżniamy tu następujące modele: hierarchiczny, sieciowy, relacyjny, obiektowy.

Interfejs użytkownika oznacza całość oprogramowania wraz z odpowiednimi urządzeniami, służącymi do wymiany informacji między czło-



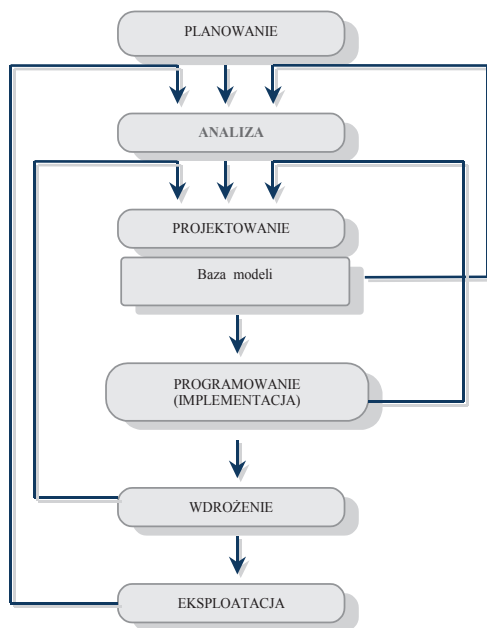
wiekiem a komputerem (czyli możliwości łatwej i przyjemnej współpracy na drodze człowiek–komputer).

Na rysunku 1 pokazane są poszczególne etapy projektowania systemu, które są ze sobą ściśle powiązane.

### 3. CHARAKTERYSTYKA PROCESU ANALIZY SYSTEMOWEJ

Analiza i projektowanie obiektowe są to pojęcia, techniki i narzędzia służące do analizy problemu będącego przedmiotem planowanego przedsięwzięcia informatycznego oraz do projektowania aplikacji lub systemu, które bazują na pojęciach obiektowości, wykorzystując metodykę obiektową.

Metodyki obiektowe to metodyki wykorzystujące pojęcia obiektowości dla celów modelowania pojęciowego oraz analizy i projektowania systemów informatycznych. Analiza i projektowanie obiektowe są wspomagane poprzez budowę modeli (zwykle w postaci diagramów graficznych) odwzorowujących klasy obiektów, ich atrybuty, metody, powiązania oraz zachowanie. Klasa przedstawia informację statyczną i stanowi wzorzec, szablon do tworzenia obiektów. Zwykle różne kategorie bytów (elementów ze świata rzeczywistego) mają podobne atrybuty i wspólne zachowania.



Rysunek 1. Cykl życia systemu informatycznego

Analiza systemowa to nowoczesny sposób projektowania systemów informatycznych, pozwalający na wprowadzenie nowoczesnych rozwiązań organizacyjnych i technicznych, wspomagających zarządzanie firmą. Celem analizy systemowej jest stworzenie nowego modelu logicznego projektowanego systemu, na podstawie którego opracowywany jest model fizyczny.

Analiza systemowa jest tylko jednym z etapów projektowania systemów informatycznych (rysunek 1). Jest to proces rozpoznawania problemu, wyjaśniania założeń i wymagań, modelowania dziedziny problemu, dokumentowania ważnych problemów będących przedmiotem projektu. Analiza systemowa stanowi studium dziedziny problemu będącego przedmiotem projektu, którego podstawowym zadaniem jest określenie (inventaryzacja) wymagań użytkownika w stosunku do projektowanego systemu.

W wyniku przeprowadzania etapu analizy systemu możliwe jest określenie wszystkich czynników lub warunków związanych z dziedziną przedmiotową, które mogłyby mieć wpływ na decyzje projektowe oraz na przebieg procesu projektowego, przy uwzględnieniu wymagań systemu i użytkownika. Analiza systemowa prowadzi do specyfikacji (określenia) zachowań systemu.

Stosowane obecnie metody analizy systemowej (oparte na strukturalnych lub obiektowych metodach projektowania) pozwalają na otrzymanie pełnej specyfikacji wymagań dla systemu (stanowi ona formalny opis wymagań dla systemu, czyli tego, co powinien wykonywać).

Specyfikacja wymagań dla systemu zawiera:

- listę zadań, jakie system powinien obsłużyć przy uwzględnieniu przyjętych założeń i ograniczeń dla projektu systemu; lista zadań systemu zawiera wyszczególnienie zadań dla systemu oraz dokładną listę potrzeb klienta;
- specyfikację procesów – funkcji, jakie system powinien wykonać, aby spełnić zadania przydzielone mu z listy zadań.

Analiza systemowa do modelowania systemu wykorzystuje narzędzia graficzne — diagramy.

Celem etapu analizy systemowej jest opracowanie:

- modelu logicznego projektowanego systemu informatycznego,
- modelu fizycznego systemu.

Model logiczny to zbiór informacji określający zachowanie się systemu.

Model fizyczny to propozycja konkretnej realizacji (implementacji) modelu logicznego, czyli projekt techniczny oprogramowania i rozwiązań sprzętowych.

Podstawowe elementy modelu logicznego to:

- lista funkcji systemu,
- określenie obiektów zewnętrznych systemu,
- określenie obiektów wewnątrz systemu i współzależności między funkcjami systemu.

Model fizyczny bazy danych (*Database Physical Model*) to model wycinka rzeczywistości wyrażony za pomocą klas, tabel, plików i struktur umożliwiających dostęp do danych. Model fizyczny bazy danych zawiera wszystkie szczegółowe informacje dotyczące organizacji danych w bazie danych (tabele, pliki, fizyczną organizację zbiorów) i jest odwzorowaniem modelu logicznego projektowanego i analizowanego systemu. Model fizyczny jest konkretną realizacją modelu logicznego na podstawie wybranego modelu danych (hierarchicznego, sieciowego, relacyjnego, obiektowego).

## 4. OBIEKTOWE METODY ANALIZY SYSTEMU

Analiza i projektowanie obiektowe (*Object-Oriented Design*) obejmują metody i sposoby rozwiązywania problemów programowania z zastosowaniem metodologii obiektowej. Metody obiektowe analizy systemów pojawiły się w latach osiemdziesiątych i są nadal intensywnie rozwijane. Ich podstawą jest wyróżnianie w systemie pewnych składowych (obiektów).

Modelowanie obiektowe charakteryzuje się warstwowością, czyli występowaniem na różnych warstwach modelu danych o różnych poziomach abstrakcji. Analiza obiektowa systemów wykorzystuje tę właściwość w celu analizy danych o różnych poziomach abstrakcji.

Cechy charakterystyczne obiektowej metodologii projektowania i analizy systemów są następujące:

- ukrywanie informacji;
- abstrakcja;
- dziedziczenie;
- hierarchia;
- wykorzystanie wtórne;
- typy definiowane przez użytkownika;
- hierarchia klas;
- klasy abstrakcyjne;
- wielopostaciowość, elastyczność.

Ukrywanie informacji (*Information Hiding*) służy bezpiecznemu programowaniu i służy oddzieleniu interfejsu użytkownika od kodu programu. Abstrakcja (*Abstraction*) jest to ogólne pojęcie oznaczające zaniebdywanie cech szczególnych, utrudniających poszukiwanie rozwiązania (konstrukcję algorytmu albo wyodrębnienie istotnych struktur w danych). Dziedziczenie jest to związek występujący pomiędzy klasami obiektów, określający przekazywanie cech (definicji atrybutów, metod itd.) z nadklasy do jej podklas. Dziedziczenie służy do budowania hierarchii klas. Hierarchia klas pomaga w utrzymaniu przejrzystości struktury programu i jest ściśle związana z dziedziczeniem pomiędzy klasami obiektów.

Wykorzystanie wtórne, ponowne użycie (*Reuse, Reusability*) jest to koncepcja powtórnego stosowania i wykorzystania składowych programowych.

W obiektowych językach programowania użytkownik może zdefiniować własne typy danych.

Klasy abstrakcyjne są ściśle związane z możliwością definiowania przez użytkownika nowych typów danych, w tym również abstrakcyjnych. Wielopostaciowość (polimorfizm) jest to możliwość określania za pomocą jednej nazwy wielu metod (działań na obiektach), których właściwy dobór odbywa się w trakcie wykonania programu. Polimorfizm jest określany jako zdolność do wykonywania określonego działania na argumentach różnych typów (np. sortowanie liczb i napisów przy użyciu tej samej procedury).

Hermetyzacja polega na ukrywaniu niektórych informacji. Najczęściej dla użytkownika widoczne są wyłącznie niektóre metody obiektu, ale niewidoczny jest jego stan (atrybuty). Hermetyzacja jest podstawową techniką abstrakcji (to znaczy ukrycia wszelkich szczegółów) danego obiektu, które na danym etapie rozpatrywania (analizy, projektowania, programowania) nie stanowią jego istotnej charakterystyki.

Warstwowość jest widoczna wyraźnie w diagramach klas, w których najwygodniejszym sposobem prezentacji modelu jest podejście „od ogółu do szczegółu”.

Technika projektowania obiektowego posługuje się pewnymi pojęciami, które niezależnie od przyjętej metodologii oznaczają te same pojęcia (choć czasami mogą mieć nieco inne nazwy).

Cechy charakterystyczne projektowania i analizy obiektowej systemów informatycznych to:

- integracja wybranego modelu i procesów występujących w systemie;
- zapewniona spójność modelowania (wynikająca z cech charakterystycznych modelowania obiektowego);
- zapewniona spójność działania danych i procedur (wynikająca z powiązania ze sobą w obiektach);
- hermetyzacja – zmiana elementów danych dotyczy tylko określonego obiektu, może to mieć wpływ na procedury i zmiany w innych obiektach;

- grupowanie obiektów w klasy – obiekty wykazujące wspólne cechy grupowane są w klasy obiektów, każdy obiekt może należeć tylko do jednej klasy;
- dziedziczenie danych i procedur w ramach hierarchii klas – tworzenie z klas już istniejących nowych klas dziedziczących pewne cechy – atrybuty;
- komunikacja między obiektami za pomocą przesyłania komunikatów.

Analiza jest jednym z ważniejszych etapów projektowania systemów informatycznych i ściśle wiąże się z fazą określania wymagań oraz następującą po niej fazą projektowania systemów:

- Celem fazy określania wymagań jest udzielenie odpowiedzi na pytanie: „co ma wykonywać i przy jakich ograniczeniach system ma działać?”. Wynikiem tej fazy jest zbiór wymagań, czyli zewnętrzny opis systemu.
- Celem fazy projektowania jest udzielenie odpowiedzi na pytanie: „jak system ma zostać zaimplementowany?”. Jej wynikiem jest projekt oprogramowania, czyli opis sposobu implementacji.
- Celem fazy analizy jest udzielenie odpowiedzi na pytanie: „jak system ma działać?”. Rezultatem jest logiczny model systemu, opisujący sposób realizacji przez system podstawowych wymagań, lecz abstrahujący od szczegółów implementacyjnych.

Ponieważ celem fazy analizy jest budowa logicznego modelu systemu, jest ona także nazywana fazą modelowania. Model systemu staje się podstawą tworzenia projektu. Zadania wykonywane w fazie analizy mogą być więc także utożsamiane z wyróżnianym tradycyjnie projektowaniem wysokiego poziomu, niebiorącym pod uwagę większości szczegółów implementacyjnych.

Metodami wykorzystywanymi w fazie analizy są:

- metody budowy modelu (metody analizy),
- notacje służące do zapisu modelu (diagramy),
- narzędzia ułatwiające stosowanie notacji i metod.

Należy podkreślić, że metody analizy nie są prostymi algorytmami, których stosowanie przez różne osoby zapewni osiągnięcie tych samych wyników. Proponowane metody są raczej zbiorem ogólnych wskazówek i rad. Analiza pozostaje w dużej mierze sztuką, której nie można się nauczyć wyłącznie na podstawie literatury czy wykładów. Dla osiągnię-

cia prawdziwego zaawansowania w modelowaniu złożonych systemów niezbędne jest doświadczenie praktyczne. Z jednej strony modele tego samego systemu stworzone przez różne osoby, stosujące te same notacje i metody, mogą się znacznie różnić. Z drugiej zaś – nie powinny być zupełnie odmienne.

Analiza obiektowa dysponuje bardziej różnorodnym wachlarzem diagramów, przez co daje możliwość przedstawienia projektowanego systemu z różnych punktów widzenia. Można przypuszczać, że dzięki temu analiza systemu jest bardziej precyzyjna. Dodatkowym atutem metodologii obiektowej jest jej ciągły rozwój, przez co staje się ona coraz elastyczniejsza (należy tu wspomnieć o notacji UML).

Z definicji analizy systemowej (modelowania) wynika, że powinna ona dać nam odpowiedź na podstawowe pytanie, jak ma działać projektowany system.

Zasadniczymi celami analizy i projektowania obiektowego są: odzielenie wymagań stawianych systemowi od projektu tego systemu, odzielenie projektu tego systemu od jego implementacji oraz utworzenie abstrakcyjnych modeli odpowiednich do problemu i nieprzekraczających możliwego do zaakceptowania stopnia złożoności.

Koncepcja przypadków użycia (*use cases*) zakłada odwzorowanie struktury systemu z punktu widzenia jego użytkownika.

Obiektowe metody projektowania i analizy najczęściej służą modelowaniu procesów biznesowych. Ich główne zadanie polega na takim zaprojektowaniu systemu informatycznego, aby wspierał wszelkie zmiany zachodzące z upływem czasu w firmie.

Modelowanie procesów biznesowych może być przeprowadzane z kilku powodów. Najważniejsze z nich to stworzenie nowej struktury i filozofii funkcjonowania firmy w ramach procesu reorganizacji oraz poprawa funkcjonowania firmy bez wprowadzania radykalnych zmian.

Proces jest jednym z ważniejszych elementów, jaki projektant powinien odzwierciedlić w projektowanym przez siebie systemie informatycznym czy bazie danych. Aby to uczynić, musimy sobie zdać sprawę, czym jest sam proces. Proces jest działaniem (lub zbiorem działań), ze ściśle określonym początkiem i końcem działania, mającym na celu dostarczenie pewnej określonej usługi lub produktu (to jedna z definicji).

Pod określeniem „początek” i „koniec działania” rozumiemy nie tylko ściśle określenie czasu, w jakim dany proces jest rozpoczęty czy



zakończony, ale również zbiór warunków, jakie muszą być spełnione, aby dany proces mógł w ogóle zaistnieć, a później się zakończyć.

Procesem więc są takie zdarzenia, jak np. zakup materiałów potrzebnych do wyprodukowania określonego produktu, naprawa urządzeń, serwowanie posiłków, sprzedaż produktu i wiele innych zachodzących w danym przedsiębiorstwie.

Należy również zauważyć, że proces, w zależności od tego, co ma dostarczyć, może angażować różne ilości środków i ludzi, jak i rozciągać się na kilka działów danego przedsiębiorstwa lub organizacji. Np. zakup materiałów może się rozciągać na trzy działy: dział sprzedaży, księgowość i magazyn. Natomiast naprawa w ograniczonej formie może się ograniczyć tylko do działu serwisu.

Z punktu widzenia przedsiębiorstwa lub organizacji procesy można podzielić na:

- procesy zewnętrzne – są to procesy, które mają na celu dostarczenie pewnych wartości (usług, produktów) na zewnątrz firmy, np. klientom firmy;
- procesy wewnętrzne – są to procesy, które służą głównie do przygotowywania lub wspomaganie procesów biznesowych zewnętrznych i zwykle zamykają się w obrębie danego przedsiębiorstwa lub organizacji.

Dla przedsiębiorstwa najważniejszymi procesami są procesy wewnętrzne. Przynoszą one bezpośrednio przedsiębiorstwu określoną wartość mierzalną lub niemierzalną. Są one nazywane procesami biznesowymi. Prawidłowe wyodrębnienie i modelowanie tych procesów jest kluczowym czynnikiem umożliwiającym prawidłowe zbudowanie systemu informatycznego wspierającego ich wykonanie.

Podczas budowy systemu informatycznego największym niebezpieczeństwem, jakie może się zdarzyć projektantom (i programistom), jest niezrozumienie się zleceńodawców i przyszłych użytkowników. Prowadzi to do zaprojektowania czegoś, co nie jest zgodne z oczekiwaniami i wymogami. Z tego powodu bardzo ważnym elementem projektu jest przygotowanie modelu wymagań przyszłych użytkowników. W zależności od precyzji zdefiniowania wymagań przyszłych użytkowników cały projekt jest skazany na sukces lub porażkę.

Zaprojektowany model systemu powinien możliwie jak najdokładniej odzwierciedlać wymagania użytkowników. Praktycznie każda wpro-

wadzana zmiana do przygotowanego już modelu jest potencjalnym źródłem zaburzeń w spójności i integralności całego projektu. Z tego wynika, że wymagania zleceniodawców i użytkowników muszą być integralną częścią projektu.

Do modelowania procesów biznesowych stosuje się głównie dwie metody analizy systemu, oparte na obiektywnym modelu projektowania i obiektywnych językach projektowania:

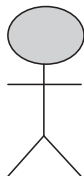
- metoda Jacobsona,
- metoda Lynx.

## Metoda Jacobsona

Metoda Jacobsona rozpatruje procesy biznesowe, opierając się na trzech podstawowych modelach (modelach wymagań użytkownika):

- modelu przypadków użycia – odzwierciedla w sposób bezpośredni wymagania użytkownika i funkcjonalność systemu;
- modelu obiektów systemu – model ten przedstawia budowę (architekturę) systemu (zawiera obiekty i powiązania między nimi);
- diagramy interakcji – przedstawiają komunikację między obiektami w trakcie realizacji przypadków użycia.

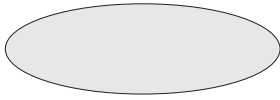
Model przypadków użycia (*Use Case Model*) opisuje procesy (zachowanie systemu), zachodzące w organizacji (firmie) z punktu widzenia zaspokojenia potrzeb klienta. Pokazuje on wszystkie powiązania pomiędzy procesami i zewnętrznym środowiskiem oraz pokazuje usługi świadczone przez organizację na rzecz środowiska zewnętrznego. Model przypadków użycia jest przedstawiany na graficznym diagramie przypadków użycia. Na diagramie tym użytkownicy systemu to aktorzy grający przypisane im role. Aktorem może być osoba, organizacja lub rzecz.



Nazwa aktora

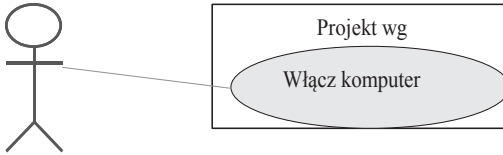
**Rysunek 2.** Przykład – aktor jest abstrakcyjnym użytkownikiem systemu

Dla przedstawienia na diagramie przypadku użycia używa się symbolu:



(nazwa przypadku użycia)

**Rysunek 3.** Przykład symbolu przypadku użycia



użytkownik komputera

**Rysunek 4.** Przykład diagramu przypadków użycia (aktor obiekt)

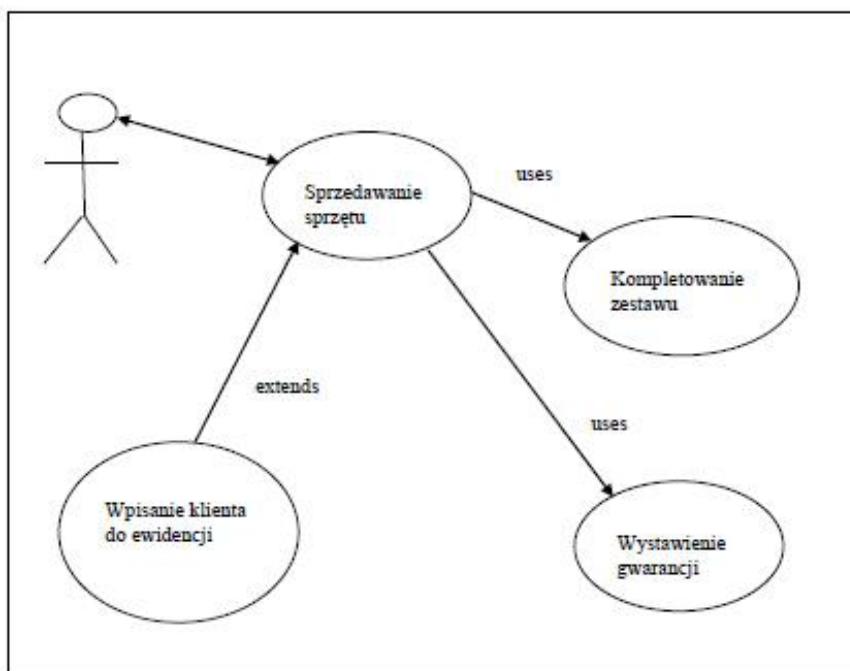
Przypadek użycia (*Use Case*) jest pewnym modelem procesu biznesowego, który określa wzajemne interaktywne powiązania aktora z systemem. Przypadek użycia zawiera opis transakcji występujących w systemie. Transakcje to operacje lub ciąg operacji w systemie, które albo są całkowicie wykonane, albo też, w przypadku przerwania ich wykonywania, skutki ich działania są całkowicie anulowane i nie pozostawiają żadnych śladów w systemie.

Model przypadków użycia nie jest modelem wystarczającym do uzyskania pełnego opisu funkcjonowania firmy. Model przypadków użycia nie zawiera przepisu, w jaki sposób są realizowane różne przypadki użycia. Na rysunku przykładowym zaznaczony jest przypadek użycia „włącz komputer”, ale nie ma możliwości podania przepisu, jak ma to być wykonane.

Model obiektów opisuje wewnętrzną strukturę każdego procesu biznesowego, zachodzącego w obrębie firmy, pokazując poszczególne elementy składowe tego procesu, miejsce oraz sposób ich wykonania. Model obiektów opisuje, w jaki sposób wykonywany jest określony przypadek użycia, oraz przedstawia sobą wewnętrzną architekturę systemu. Zbudowany jest z powiązanych różnego typu obiektów (interfejsu, encji, sterujących). Wymienione obiekty pełnią funkcję w systemie i mają swoje odpowiedniki graficzne.

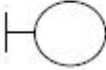


W analizie obiektowej system traktowany jest jako zbiór komunikujących się ze sobą i współpracujących obiektów. Wyróżniamy trzy typy obiektów:

- obiekty interfejsu,
- obiekty aplikacji,
- obiekty sterujące.



**Rysunek 5.** Diagram przypadków użycia – system obsługi sklepu komputerowego

Obiekty interfejsu to te obiekty, z którymi bezpośrednio komunikuje się aktor. Praktycznie wszystko, co widzi użytkownik, pracując w środowisku graficznym (przyciski, okna, listy itp.), jest przykładem obiektów interfejsu.

	<p>Obiekt interfejsu (osoby, urzędnicy znajdujące się wewnątrz firmy, z którymi komunikuje się aktor, aby zrealizować określony przypadek użycia).</p>
	<p>Obiekt encja (elementy bierne w systemie, jak produkty, dokumenty, które są potrzebne do zrealizowania przypadku użycia).</p>
	<p>Obiekty sterujące (elementy aktywne systemu, konieczne do realizacji przypadku użycia).</p>

**Rysunek 6.** Przykłady różnych obiektów

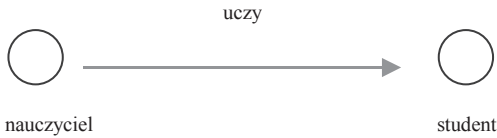
Obiekty aplikacji to takie, które z praktycznego punktu widzenia reprezentują dane w jakiś sposób zapamiętane (np. w bazie danych), a następnie udostępniające te dane do ponownego wykorzystania. Przykładem tego może być np. sprzedaż (fakt sprzedaży, dane o sprzedaży), urządzenie (dane urządzenia, pozycja w ewidencji urządzeń).

Obiekty sterujące modelują te funkcje systemu, których realizacja jest niewidoczna dla aktora, ale niezbędna do przeprowadzenia przypadku użycia. Obiektem tego typu jest np. kreator gwarancji. Obiektem sterującym można nadać szczególną funkcję reprezentowania logiki przypadku użycia. Polega to na tym, że wszelkie reguły (w tym biznesowe) i algorytmy dotyczące przeprowadzenia przypadku użycia umieszcza się w obiektach sterujących. Wtedy przebieg każdego przypadku użycia jest sterowany przez specjalnie oddelegowany obiekt.

Za pomocą obiektów sterujących modeluje się także funkcjonalność systemu, której nie można naturalnie powiązać ani z obiektami aplikacji, ani z obiektami interfejsu: zarządzanie pozostałymi obiektami, wykonywanie skomplikowanych algorytmów i obliczeń. Obiektami sterującymi mogą być: obiekt kompresujący pliki, obiekt planujący kolejne ruchy w grze, obiekt sterujący przelewami na konta itp.

Między obiektami zachodzą następujące związki (relacje):

- asocjacje (wzajemne relacje między obiektami),
- agregacje (całość–część, powiązanie obiektu z jego częścią składową),
- generalizacji–specjalizacji (dziedziczenia, relacje między obiektami dziedziczącymi wybrane cechy innych obiektów).



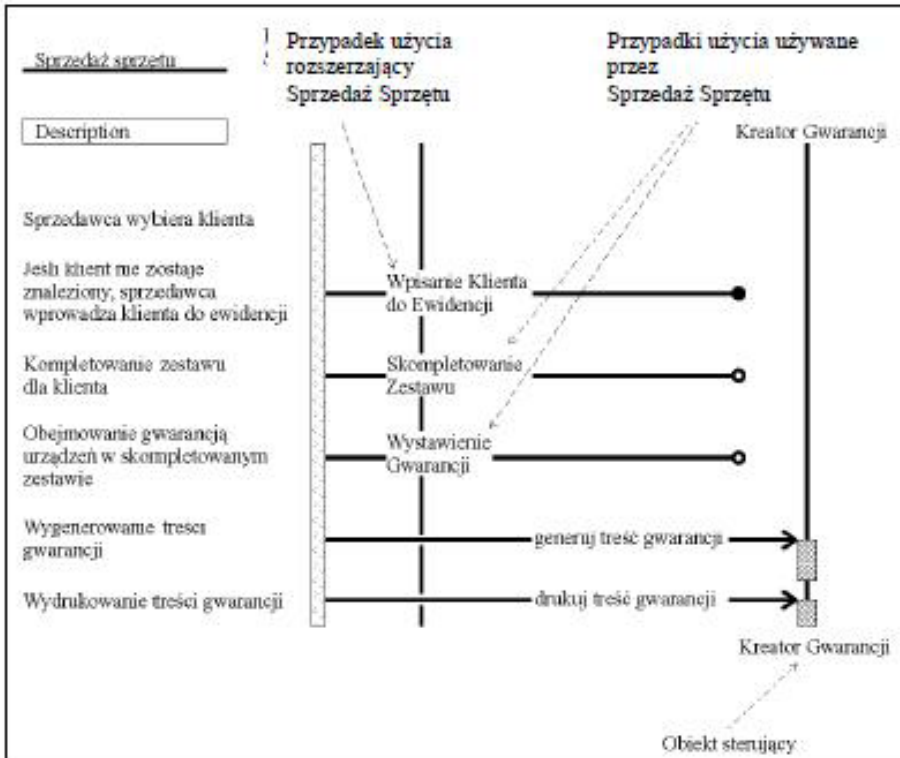
**Rysunek 7.** Przykład relacji powiązania między obiektami

Diagramy interakcji, realizując przypadek użycia, komunikują się z aktorami i ze sobą. Komunikacja odbywa się na zasadzie komunikatów (czasami zwanymi zdarzeniami).

Do przedstawienia komunikacji między obiektami w trakcie realizacji przypadku użycia stosuje się diagramy interakcji obiektów.

Na diagramie interakcji przedstawia się takie elementy, jak:

- opis przypadku użycia – szczegółowy, zawierający każdy ważny krok (transakcję) przypadku użycia;
- obiekty uczestniczące w przebiegu przypadku użycia – oznaczone są jako pionowe kreski z etykietami;
- zdarzenia i komunikaty przesyłane między obiektami – przepływ informacji między obiektami pojawia się w konsekwencji wystąpienia określonego zdarzenia, którego konsekwencją jest komunikat;
- granice systemu i architektury – oddzielają obiekty w systemie od środowiska zewnętrznego oraz obiekty interfejsu od pozostałych obiektów systemu. Granice stanowią wizualny podział pomiędzy warstwami systemu. W ten sposób widać jak poszczególne warstwy systemu komunikują się ze sobą.



Rysunek 8. Diagram interakcji obiektów

## Metodologia analizy systemów Lynxa

Metodologia analizy systemów Lynxa oparta jest na dwóch typach diagramów:

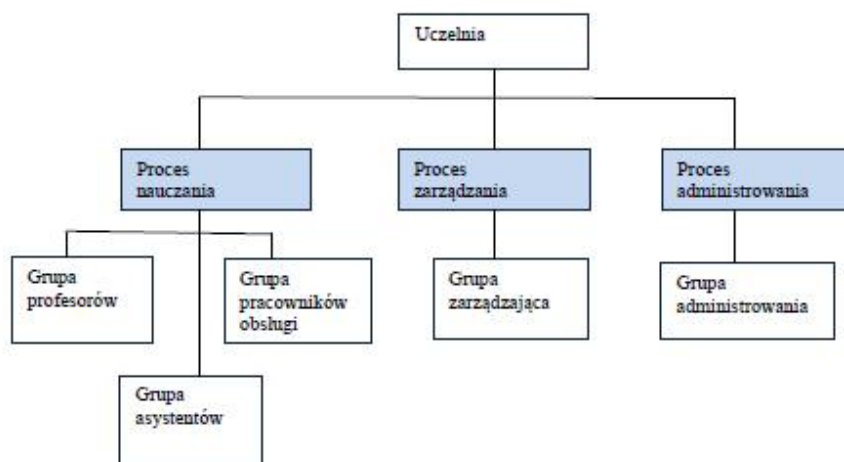
- diagram hierarchiczny procesów (*Process Hierarchy Diagram*),
- diagram wątków procesów (*Process Thread Diagram*).

Diagramy hierarchiczne procesów obrazują statyczne zmiany zachodzące między procesami. Służą one do graficznego pokazania powiązań i zależności, zachodzących w organizacji pomiędzy procesami biznesowymi, oraz ich podziałów na różnych stopniach i poziomach hierarchii.

Diagramy wątków procesów modelują dynamikę procesu. Służą do graficznego pokazania dynamicznych zależności, występujących pomiędzy poszczególnymi ogniwami, stworzonymi poprzez elementarne procesy biznesowe.

Elementarny proces biznesowy to wykonywane przez jedną osobę – w określonym miejscu i czasie – zadanie. W efekcie wykonywanego zadania (elementarnego procesu biznesowego) otrzymywana jest określona wartość. Elementarny proces biznesowy jest inicjowany przez zdarzenie biznesowe, pochodzące z wewnątrz firmy lub z jej otoczenia.

Procesem biznesowym nazywamy serię powiązanych ze sobą działań lub zadań. Proces biznesowy często jest opisywany schematem blokowym, który realizuje algorytm rozwiązania zagadnienia.



**Rysunek 9.** Przykład diagramu hierarchii procesów

Typy procesów biznesowych mogą być następujące.

- proces zarządczy, który kieruje działaniem systemu (proces zarządzania przedsiębiorstwem);
- proces operacyjny (zaopatrzenie, produkcja, marketing, sprzedaż);
- proces pomocniczy, który wspiera procesy główne (księgowość, rekrutacja).

Proces biznesowy zaspokaja potrzeby klientów. Proces biznesowy można podzielić na podprocesy o własnych atrybutach.

Wymagane cechy procesu biznesowego:

- definiowalność (określone wejście i wyjście),
- porządek (działania uporządkowane w czasie i przestrzeni),
- klient (ktoś, kto odbiera wyniki powstałe po działaniu procesu),
- zwiększanie wartości.



## 5. NARZĘDZIA MODELOWANIA W ANALIZIE SYSTEMU

Proces analizy systemu jest najczęściej wspomagany narzędziami typu CASE (*Computer Aided Software Engineering* – wspomagana komputerowo inżynieria oprogramowania).

Narzędzia typu CASE opierają się na notacji graficznej analizowanego problemu. Notacja graficzna to najczęściej diagramy. W analizie systemu wykorzystuje się diagramy graficzne do modelowania różnych aspektów systemu. W tym celu tworzy się różne modele opisu systemu. Narzędzia CASE wspomagają proces projektowania systemów informatycznych.

### Modele opisu systemu

Projektowanie systemu bardzo często zaczyna się od stworzenia modelu opisującego system. Dlaczego budujemy modele systemu? Ponieważ pozwalają w uproszczony sposób opisać projektowany system, a także uwypuklić pewne istotne cechy systemu, a inne – mniej ważne jego aspekty – w danej chwili pominąć. Model opisu systemu pozwala się komunikować z przyszłym użytkownikiem w zorganizowany sposób, bez rozpraszania się na zagadnienia i szczegóły nieistotne w danej chwili. Dodatkową zaletą analizy modelu systemu jest łatwe wprowadzanie ewentualnych zmian, jeśli w modelu zostały nieprawidłowo uwzględnione wymagania użytkownika (albo użytkownik zmienił zdanie na temat swoich potrzeb). Modele opisu systemu mogą wykorzystywać metodologię projektowania strukturalnego lub obiektowego.

W obiektowym modelu danych podstawowym pojęciem jest obiekt, który jest zbiorem:

- zmiennych,
- metod.

Zbiór zmiennych to atrybuty obiektu. Metody (funkcje) to operacje wykonywane na zmiennych.

Obiektowe bazy danych opierają się na założeniach programowania obiektowego, a więc na sposobie modelowania rzeczywistości poprzez parametry i operacje, jakie są z nimi związane. W odróżnieniu od pro-

gramowania proceduralnego, gdzie parametry stanowią niezależną część względem operacji (funkcji), w programowaniu obiektowym nie jest tworzony opis obiektu rzeczywistego, ale jego model, który obejmuje wszystkie charakteryzujące obiekt elementy. Obiekty są zgrupowane w klasie.

Klasa jest rozumiana jako opis (schemat) obiektów, które są jej wystąpieniami. Obiekty nazywamy wystąpieniami (*instances*) klasy. Obiekt jest jednostkowym wystąpieniem klasy obiektów (modelu), dla której określone są metody opisujące zachowanie się obiektu oraz jego stan (parametry).

Obiektowe bazy danych opierają się na pojęciu obiektu, który jest reprezentantem rzeczywistości. W przypadku obiektowych baz danych obiekty są scharakteryzowane przez:

- opis obiektu (wiek, nazwisko, choroba),
- zachowanie obiektu (zgłoszenie pacjenta do badania USG).

Istotna różnica pomiędzy wystąpieniem rekordu danego typu w modelu relacyjnym a wystąpieniem obiektu w modelu obiektowym polega na tym, że w przypadku obiektowej bazy danych mogą istnieć dwa obiekty posiadające dokładnie te same wartości pól, ponieważ równość pól nie oznacza równości obiektów. Cecha ta jest często niezwykle ważna, ogranicza bowiem tworzenie sztucznych kolumn (kluczy obcych) koniecznych w modelu relacyjnym.

Zalety modelu obiektowego:

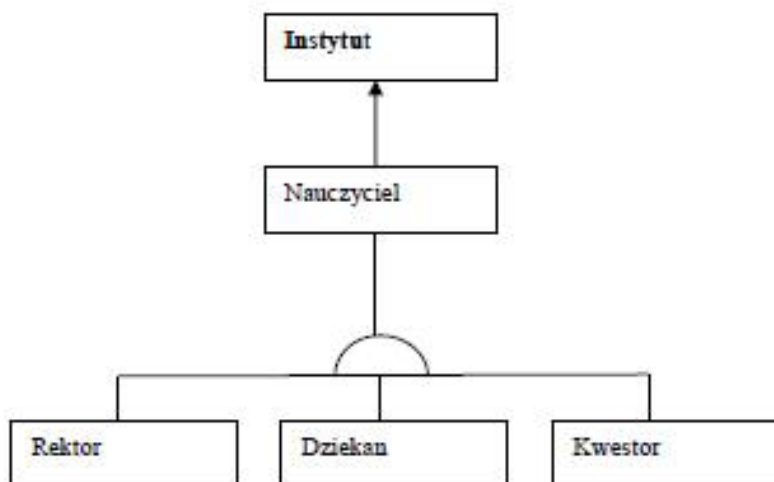
- dość naturalna reprezentacja świata,
- łatwość działania na złożonych obiektach,
- duża podatność na zmiany.

Metodyką obiektową najczęściej stosowaną do analizy systemu jest metodyka The SELECT Perspective firmy SELECT Software Tools.

Metodyka ta powstała na podstawie znanych metod:

- OMT (Object Modeling Technique) Jamesa Rumbaugh a,
- OOSE (Object-oriented Software Engineering) Ivara Jacobson a,
- Lynx firmy Computer Sciences.

Metodę tę wyróżnia przede wszystkim czterowarstwowa architektura systemu oraz iteracyjny i przyrostowy cykl życia projektu. Metodyka The SELECT Perspective jest obecnie jednym z wiodących standardów w rozwoju systemów korporacyjnych. Opisano cykl życia, wielowarstwową architekturę i techniki modelowania systemów.



**Rysunek 10.** Przykład modelu obiektowego opisu systemu

## 6. JĘZYK MODELOWANIA UML

Jezyk UML służy zapisaniu projektu systemu. Ponadto jezyk UML bardzo dobrze nadaje się do dokumentowania systemów.

W jezyku UML uwzględnia trzy rodzaje bloków konstrukcyjnych:

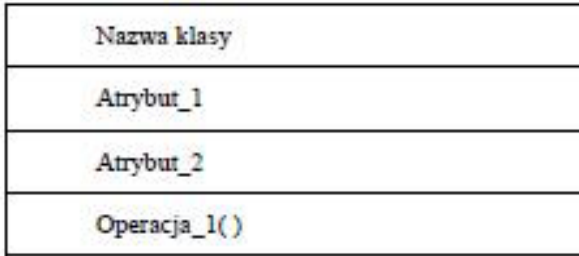
- Elementy – podstawowe obiektowe bloki konstrukcyjne stosowane do budowy modeli. Generalnie możemy podzielić je na cztery kategorie: elementy strukturalne (np. klasa, przypadek użycia), czynnościowe (np. interakcja), grupujące (pakiet) i komentujące (notatka).
- Związki – służą do łączenia elementów. Do związków należą: zależność, powiązanie i uogólnienie (w tym także realizacja).
- Diagramy – wyrażające wszystkie bloki konstrukcyjne. Diagram jest rzutem na system, przedstawiającym go schematem. Rysujemy wiele diagramów, aby przedstawić system z różnych punktów widzenia.

Jezyk UML jest używany także do modelowania procesów biznesowych, inżynierii systemów i reprezentowania struktur organizacyjnych. Systems Modeling Language UML jest jezykiem modelowania przeznaczonym dla specyficznych zagadnień inżynierii systemów. W UML do opracowywania formalnych ograniczeń można wykorzystać także jezyk Object Constraint Language (OCL) opracowany pierwotnie przez IBM. Jezyk UML służy do modelowania dziedziny problemu (opisywania-modelowania fragmentu istniejącej rzeczywistości). Na podstawie UML można zamodelować na przykład, czym zajmuje się jakiś dział w firmie. W przypadku stosowania UML do analizy oraz do modelowania wybranej rzeczywistości (przykład Szkoła Wyższa) tworzy się w nim głównie modele systemów informatycznych. Bardzo przydatna jest analiza stanu w rzeczywistości, jeśli ma dopiero powstać system informatyczny.

Jezyk UML jest głównie używany wraz z jego reprezentacją graficzną – jego elementom przypisane są symbole, które wiązane są ze sobą na diagramach. Zdefiniowane w UML bloki konstrukcyjne pozwalają w przejrzysty sposób przeanalizować interesującą nas istniejącą rzeczywistość, utworzyć model analizy (na podstawie diagramów analizy) i wyciągnąć wnioski, co tak naprawdę ma wykonywać przewidywany system. W UML przyjęto następujące oznaczenia:

## Klasa

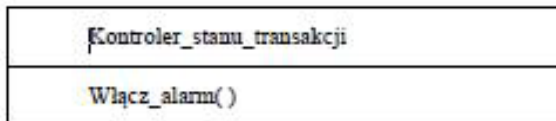
Klasa – to opis zbioru obiektów, które mają takie same atrybuty, operacje, związki i znaczenie. Klasa realizuje jeden lub więcej interfejsów. Oznaczamy ją za pomocą prostokąta.



Rysunek 11. Oznaczenie klasy

Symbol klasy (na diagramie) może zawierać jej atrybuty i operacje. Na diagramie możemy umieszczać symbole niepełne. Ogólnie powinno się ujawniać tylko te właściwości klasy, które są potrzebne do zrozumienia jej wystąpienia w danym kontekście.

Wyróżnia się wśród klas, **klasy aktywne**, czyli takie, których obiekty reprezentują elementy działające równoległe z innymi. Takie obiekty mogą samodzielnie rozpocząć przepływ sterowania. Na diagramie oznaczamy je poprzez pogrubienie obramowania.



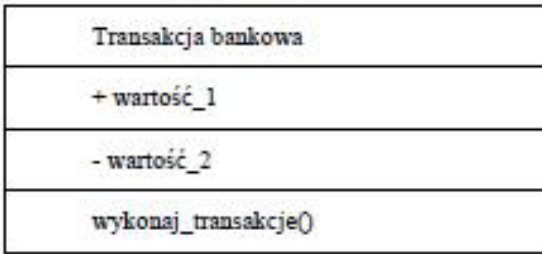
Rysunek 12. Oznaczenie klasy aktywnej

Ogólna definicja klasy:

*[widoczność] nazwa [liczebność] [:typ] [= wartPocz] [{właściwości}]*

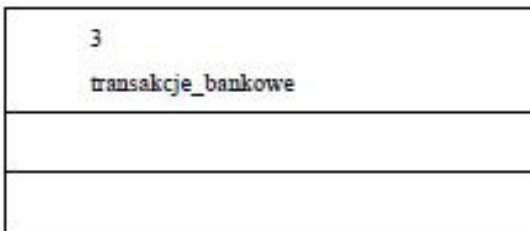
Widoczność atrybutów klasy może być:

- publiczna (+),
- prywatna (-),
- chroniona (#).



**Rysunek 13.** Przykładowa klasa (zaznaczony atrybut widoczność klasy)

Liczebność to ilość egzemplarzy klasy. Domyślnie przyjmuje się, że istnieje dowolnie wiele obiektów danej klasy. (Na rysunku określono, że istnieją trzy egzemplarze klasy Transakcje bankowe).



**Rysunek 14.** Pokazano liczebność klasy (transakcje bankowe – trzy egzemplarze)

Język UML dopuszcza trzy określenia właściwości atrybutu:

- *changeable* – (domyślnie) nie ma ograniczeń co do możliwości modyfikacji wartości atrybutu;
- *frozen* – wartość atrybutu nie może być zmieniana po zainicjowaniu atrybutu;
- *addOnly* – dotyczy atrybutów o liczebności większej od jeden. Można dodawać nowe wartości, ale raz dodana wartość nie może być już usunięta czy zmieniona.

Ponadto określamy dwa rodzaje zasięgu:

- *instancje* – każdy egzemplarz klasyfikatora (klasy, komponentu, ...) przechowuje oddzielną wartość tego składnika;
- *classifier* – istnieje tylko jedna wartość tego składnika wspólna dla wszystkich egzemplarzy klasyfikatora. Nazwę składnika o zasięgu *classifier* podkreślamy.

Ogólna definicja operacji:

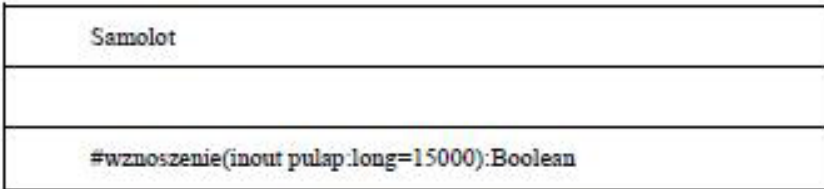
**[widoczność] nazwa [(listaParametrow)] [: typWyniku] [{właściwości}]**

Ogólna definicja parametru:

**[tryb] nazwa : typ [= wartoscDomyslna]**

Dostępne tryby to:

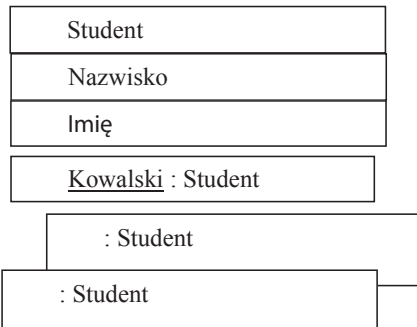
- *In* (parametr wejściowy),
- *Out* (wyjściowy),
- *Inout* (wejściowy, który może być modyfikowany).



**Rysunek 15.** Klasa ze zdefiniowanymi parametrami

### Obiekty (egzemplarze klas)

Obiekty są przedstawiane tak samo jak klasy, z tą różnicą, że ich nazwy są podkreślane. (Rysunek przedstawia klasę „Student” oraz jej dwa obiekty: „Kowalski” – jawnie przypisany do klasy klient oraz wielokrotny obiekt anonimowy klasy „Student”).



**Rysunek 16.** Klasa „Student” oraz obiekty klasy „Kowalski” i obiekt anonimowy „Student”

a: nauczyciel [akademicki]
Numer_PESEL="123456789"

**Rysunek 17.** Obiekt „nauczyciel” i zobowiązania (notatka)


Zobowiązania klasy (*Responsibilities*) są umieszczane w najniższej sekcji symbolu klasy lub obrazowane za pomocą notatki).

Wzorce klas (*template*) przedstawiane są jak klasy z dodatkowym prostokątem narysowanym linią przerywaną.

Exemplarze wzorca klas można modelować na dwa sposoby:

- jawnie (poprzez użycie zależności stereotypowanej jako <<bind>>),
- niejawnie (definiujemy klasę, której nazwa zawiera wszystkie parametry wzorca klasy).

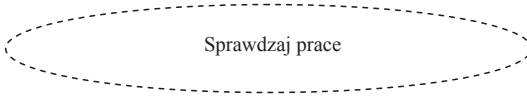
Standardowe stereotypy klasy:

- *actor*: aktorzy to specyficzny rodzaj klas oznaczający elementy zewnętrzne w stosunku do modelowanego systemu. Taka klasa określa zbiór ról odgrywanych przez użytkowników systemu). Aktor na diagramach przedstawiany jest  ;
- *enumeration*: określa typ wyliczeniowy (dopuszczalne wartości typu pochodzą z pewnego ograniczonego zbioru identyfikatorów);
- *implementationClass*: określa implementację klasy w pewnym języku programowania;
- *interface*: klasa jest interfejsem – określa zestaw operacji oferowanych przez klasę lub komponent;
- *process*: określa klasyfikator, którego egzemplarze są przepływami sterowania;
- *signal*: klasa określa sygnał – bodziec przekazywany między obiektami;
- *exception*: klasa określa zdarzenie, które może być spowodowane lub wykryte przez operację (rodzaj sygnału);
- *type*: określa klasę abstrakcyjną, używaną tylko do zdefiniowania struktury i zachowania zbioru obiektów;
- *utility*: określa klasę, której wszystkie atrybuty i operacje mają zasięg klasy.



## Przypadek użycia

Przypadek użycia to ciąg akcji wykonywanych w celu dostarczenia określonego wyniku. Na diagramie przypadek użycia przedstawiamy w postaci elipsy z nazwą pośrodku.



**Rysunek 18.** Przypadek użycia – symbol

## Kooperacja

Kooperacja to reprezentacja współdziałających ze sobą zestawów obiektów w celu wywołania pewnego zachowania się systemu. Realizację przedstawia się jako przypadek użycia. Kooperacje służą do modelowania mechanizmów. W UML przedstawiamy zestaw klas oraz ich interakcję z innymi klasami. Pojedyncza klasa może brać udział w wielu kooperacjach

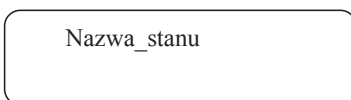
Kooperację przedstawiamy w postaci elipsy o przerywanej linii brzegowej. Jeśli chcemy dokładniej określić kooperację, to najczęściej opisujemy ją za pomocą diagramów klas i diagramów interakcji.



**Rysunek 19.** Symbol kooperacji

## Stan obiektu

Stan obiektu to stan, w jakim znajduje się obiekt (spełnia jakiś warunek lub wykonuje jakąś czynność). Stan przedstawia się jako prostokąt o zaokrąglonych rogach.



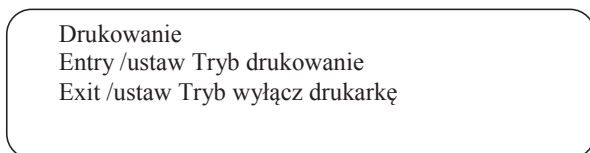
**Rysunek 20.** Stan obiektu – symbol

Stan może zawierać akcje wejściowe (wykonywane przy wejściu do tego stanu) oraz akcje wyjściowe – oznaczane *entry*, *exit*. Do oznaczania czynności wykonywanych stale w danym stanie służy słowo *do*. Dla danego stanu możemy określić przejścia wewnętrzne (notacja: zdarzenie/akcja). Jeśli zachodzi zdarzenie, wywoływana jest skojarzona z nim akcja. Wyróżniane są dwa stany:

- wejściowy,
- wyjściowy.

### Przejścia z jednego do drugiego stanu

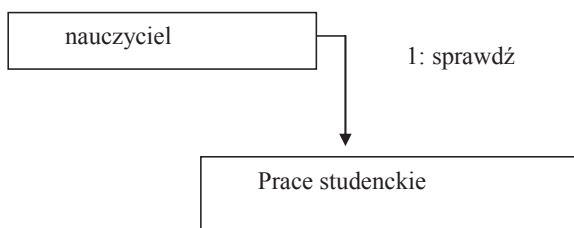
Przejścia oznaczane są strzałkami, które obrazują ścieżki przepływu sterowania pomiędzy stanami. Przejścia są wywoływane poprzez wystąpienie zdarzenia uruchamiającego (sygnału, wywołania operacji, ...) lub automatycznie (natychmiast po zakończeniu czynności w stanie źródłowym).



**Rysunek 21.** Przejścia z jednego do drugiego stanu

### Komunikat

Komunikat jest wysyłany pomiędzy obiektami w celu wywołania pewnej operacji. Ogólnie komunikat przedstawiamy w postaci strzałki z zakreskowanym grotem (opisany jest nazwą i numerem porządkowym).



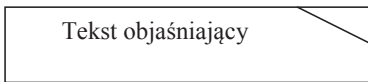
**Rysunek 22.** Przejścia z jednego do drugiego stanu

Standardowe stereotypy komunikatu:

- *copy*: wskazuje, że obiekt docelowy jest dokładną, ale niezależną kopią źródła;
- *create*: wskazuje, że komunikat tworzy obiekt docelowy;
- *destroy*: wskazuje, że komunikat niszczy obiekt docelowy;
- *become*: wskazuje, że cel jest tym samym obiektem co źródło, ale reprezentuje ten obiekt w późniejszej chwili.

## Notatka

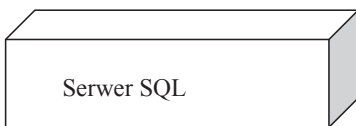
Notatka to komentarz objaśniający. Notatkę wprowadza się w następującej postaci graficznej.



**Rysunek 23.** Wygląd Notatki

## Węzeł

Węzeł to fizyczny składnik działającego systemu (np. serwer, router lub inny komputer, na którym będzie zainstalowany tworzony system). Reprezentuje zasoby obliczeniowe. Węzłów używa się do modelowania układu sprzętu komputerowego, na którym działa system. Węzły przedstawia się w postaci trójwymiarowych prostokątów z identyfikującą go nazwą.



**Rysunek 24.** Węzeł systemu

## Związki

Związki służą do łączenia elementów. Wyróżnia się następujące związki:

## Powiązanie klas

Powiązanie jest związkiem strukturalnym pomiędzy elementami. Powiązanie wskazuje, że obiekty jednego elementu są połączone z obiektami drugiego elementu. Powiązanie oznaczamy za pomocą linii ciągłej. Często do powiązania dodajemy także nazwy ról oraz oznaczenia liczebności.

Poprawne jest powiązanie, gdy:

- oba końce wskazują na tę samą klasę,
- powiązane są więcej niż dwie liczby klas,

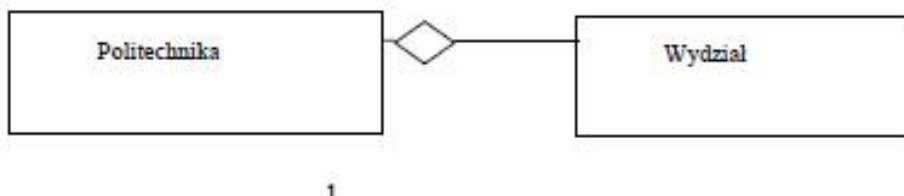
Podając liczebność na pewnym końcu powiązania (przy pewnej klasie), wskazujemy, ile obiektów tej klasy musi być połączonych z każdym obiektem klasy znajdującej się na przeciwnym końcu powiązania. Powiązanie między dwoma klasami wskazuje, że można przejść z obiektu jednej z tych klas do obiektu drugiej i odwrotnie. Można również wskazać kierunek nawigacji po obiektach.



Rysunek 25. Powiązanie klas

## Agregacja klas

Szczególnym przypadkiem powiązania jest agregacja (więź między całością i częścią). Agregacje oznaczamy za pomocą rombu po stronie całości.



Rysunek 26. Związek agregacji klas

Agregacja umożliwia oddzielenie całości od części. Nie wpływa na kierunki nawigacji ani nie wiąże czasu życia części i całości. Agregacja całkowita to agregacja charakteryzująca się relacją wyłącznej własności oraz jednością czasu życia i części. Agregację całkowitą oznaczamy poprzez wypełnienie rombu.



Rysunek 27. Związek klas – agregacja całkowita

### Wiązanie obiektów klas

Wiązanie występujące pomiędzy obiektami jest odpowiednikiem powiązania klas. Oznaczamy je za pomocą linii ciągłej

\_\_\_\_\_

Rysunek 28. Symbol powiązania



Rysunek 29. Wiązanie klas

Zwykle wiązania (między obiektami) są egzemplarzami powiązań (między klasami). Dwa obiekty, między którymi istnieje wiązanie, mogą wysyłać do siebie komunikaty. Standardowe stereotypy powiązania:

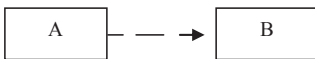
- *association* – umieszczona na jednym końcu powiązania wskazuje, że odpowiadający obiekt jest widoczny przez to powiązanie;
- *global* – umieszczone przy obiekcie na jednym końcu wiązania oznacza, że obiekt ten jest widoczny, ponieważ jest w otaczającym zasięgu;
- *local* – umieszczone przy obiekcie na jednym końcu wiązania oznacza, że obiekt ten jest widoczny, ponieważ jest w lokalnym zasięgu;

- *parameter* – umieszczone przy obiekcie na jednym końcu wiązania oznacza, że obiekt ten jest widoczny, ponieważ jest parametrem;
- *self* – umieszczone przy obiekcie na jednym końcu wiązania oznacza, że obiekt ten jest widoczny, ponieważ to właśnie on odebrał zlecenie wykonania danej operacji.

## Rodzaje zależności pomiędzy klasami

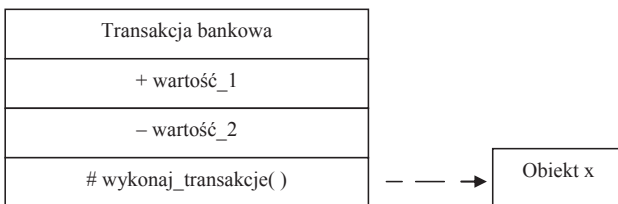
Zależność to związek użycia (używania – jeden element używa drugiego). Zmiany dokonane w definicji jednego z elementów (niezależnego) mogą mieć wpływ na znaczenie drugiego (zależnego).

(A korzysta z B)



**Rysunek 30.** Związek użycia (A korzysta z B)

Bardzo wyraźnym związkiem użycia (zależnością) jest, gdy jedna klasa używa drugiej (jako argumentu operacji). Zależność może mieć nazwę (dzięki stereotypowi).



**Rysunek 31.** Relacja zależności

Standardowe stereotypy zależności:

- *access* – pakiet źródłowy ma prawo dostępu do publicznych składników pakietu docelowego;
- *bind* – wskazuje, że źródło tworzy egzemplarz wzorca docelowego z użyciem parametrów, *call*: wskazuje, że źródło wywołuje cel (oba byty są operacjami);
- *extend* – wskazuje, że docelowy przypadek użycia rozszerza znaczenie źródłowego przypadku użycia;

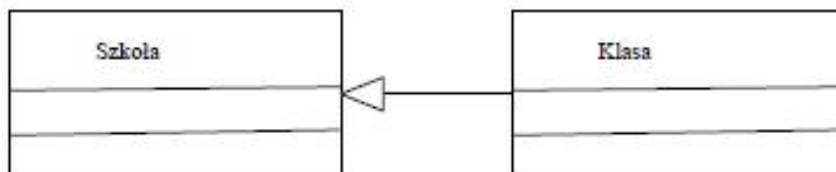
- *friend* – wskazuje, że źródło ma szczególny dostęp do wnętrza celu (zależny od języka programowania);
- *import* – dostęp, w którym zawartość publiczna pakietu docelowego jest dołączana do obszaru nazw źródła;
- *include* – wskazuje, że źródłowy przypadek użycia jawnie przyłącza docelowy przypadek użycia;
- *use* – podkreśla, że zależność jest związkiem użycia.

## Uogólnienie

Uogólnienie to związek między przodkiem i potomkiem. Grot strzałki wskazuje przodka. Uogólnień używa się względem klas, interfejsów, pakietów w celu przedstawienia dziedziczenia. Potomek może wystąpić wszędzie tam, gdzie jest spodziewany przodek. Potomek dziedziczy wszystkie właściwości przodka.



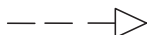
**Rysunek 32.** Związek uogólnienie



**Rysunek 33.** Przykład związku uogólnienie

## Realizacja typu klasa – komponent

Realizacja jest związkiem występującym pomiędzy elementami, na przykład interfejs i realizujące je klasy i komponenty.



**Rysunek 34.** Związek realizacja zależności

## 7. DIAGRAMY ANALIZY SYSTEMÓW

Na diagramach można wyrazić wszystkie bloki konstrukcyjne. Te bloki są najczęściej grafami, w których wierzchołkami są wybrane elementy, a krawędziami związki pomiędzy nimi. Diagram jest schematem przedstawiającym rozpatrywany system. Zwykle wykorzystujemy wiele diagramów, aby przedstawić system z różnych punktów widzenia. Język UML udostępnia kilka podstawowych rodzajów diagramów, które można dowolnie mieszać i łączyć. Każdy tworzony diagram powinien mieć unikatową nazwę.

### Diagram obiektów

Diagramy obiektów pokazują obiekty występujące w systemie oraz związki między nimi zachodzące. Obiekt jest opisany za pomocą trzech elementów: tożsamości, stanu i zachowania. Tożsamość jest cechą wyróżniającą obiekt, którą określa się za pomocą unikatowej nazwy.

Stan obiektu jest zbiorem wszystkich wartości i właściwości obiektu, które każdy z nich posiada. Obiekt ma zawsze pewien zestaw niezmiennych właściwości. Ich wartości przez cały okres istnienia obiektu mogą się zmieniać. Zachowanie obiektu to zbiór usług, które mogą być wykonane przez analizowany obiekt na rzecz innych obiektów. Zachowanie obiektu określa dynamikę występującą w modelu. Dzięki znajomości zachowania się obiektu można przedstawić relacje występujące pomiędzy obiektami lub określić czynności, które mogą zostać wykonane.

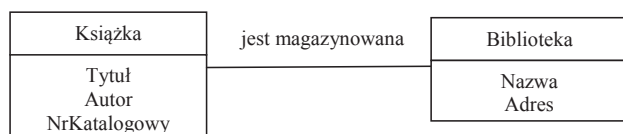
Dokładny opis obiektu za pomocą tych trzech elementów pozwala na dokładną identyfikację oraz na przypisywanie mu dużej liczby informacji. Dzięki temu jesteśmy w stanie rozróżnić obiekty w dużym stopniu podobne do siebie.

### Diagram klas

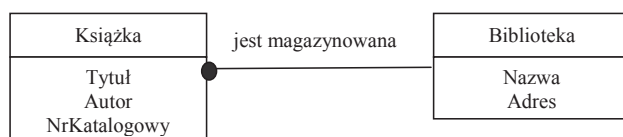
Diagram klas w modelach obiektowych jest diagramem spotykanym najczęściej. Zawiera klasy, interfejsy i związki między nimi. Modeluje statyczne aspekty systemu. We wszystkich tych związkach trawersowanie związku odbywa się w obydwu kierunkach. Oznacza to, że dany związek można czytać w obydwie strony. Np. związek na rysunku 36 oznacza, że



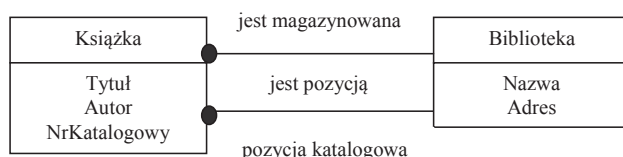
dana książka jest magazynowana tylko w jednej bibliotece, a także, że w bibliotece jest wiele książek.



**Rysunek 35.** Związek 1:1



**Rysunek 36.** Związek wiele do jeden



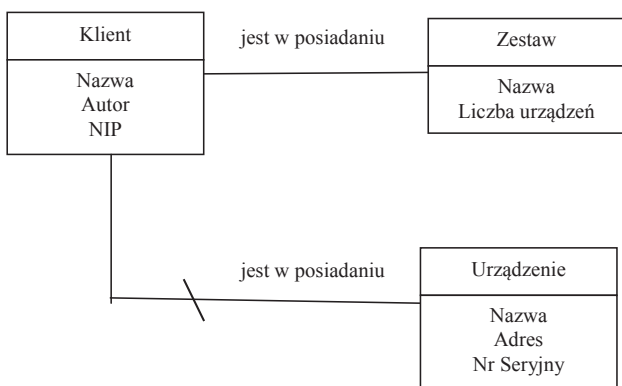
**Rysunek 37.** Związek wiele do wielu

Czasami jednak zdarza się, że związek może być trawersowany tylko w jednym kierunku. W takim przypadku kierunek związku oznacza się zamkniętą strzałką umieszczoną na końcu odcinka, zgodnie z kierunkiem jego trawersowania.

Związek wnioskowany przedstawia sytuację, w której klient zakupuje zestawy, w skład których wchodzi konkretne urządzenia. Przeglądanie zakupionych urządzeń odbywa się za pomocą przeglądania zestawów oraz urządzeń wchodzących w ich skład. Aby przyspieszyć przegląd wszystkich zakupionych urządzeń, wprowadzono dodatkowy związek pomiędzy klasą klienta a klasą urządzenia.

Rysunek 37 przedstawia sytuację, w której klient zakupuje zestawy, w skład których wchodzi konkretne urządzenia. Przeglądanie zakupionych urządzeń odbywa się za pomocą przeglądania zestawów oraz urządzeń wchodzących w ich skład. Aby przyspieszyć przegląd wszystkich zakupionych urządzeń, wprowadzono dodatkowy związek pomiędzy klasą klienta a klasą urządzenia. Jest to typ związku wnioskowanego.

Istotę związków pomiędzy klasami, a następnie między poszczególnymi obiektami można porównać do relacji między krotkami w relacyjnej bazie danych. Podobnie jak w przypadku relacji, związki pomiędzy klasami określają liczbę obiektów, z którymi może być związany dany obiekt.



**Rysunek 38.** Związek wnioskowanie

Uzupełnieniem diagramu klas i obiektów są następujące diagramy:

- diagramy dynamiczne, uwzględniające stany i przejścia pomiędzy tymi stanami;
- diagramy interakcji ustalające zależności pomiędzy wywołaniami metod;
- diagramy funkcjonalne (będące zwykle pewną mutacją diagramów przepływu danych).

## Diagram przypadków użycia

Diagram przypadków użycia (*Use Case Diagram*) pokazuje statyczne aspekty w stosunku perspektywy przypadków użycia. Pokazuje system taki, jaki jest widziany oczami użytkownika. Diagram przypadków użycia jest podstawą do wyznaczania i modelowania zachowania systemu.

Przypadki użycia określają wymagane zachowanie systemu, ale nie narzucają sposobu jego implementacji. Diagramy przypadków użycia stanowią główne narzędzie służące do modelowania zachowania się systemu, podsystemu lub klasy. Każdy z tych przypadków użycia przedstawia się na diagramie przypadków użycia oraz aktorów.

Każdy przypadek użycia może mieć warianty wiązania z innymi przypadkami użycia lub obiektami ze świata rzeczywistego. Są te powiązania następujące (zależności między przypadkami użycia):

- *extend* (rozszerza)

Powiązanie <<extend>>: strzałka prowadzi od przypadku użycia, który czasami rozszerza inny przypadek użycia – wykorzystywane w przebiegach opcjonalnych (operacje nie zawsze wykonywane).

- *include* (włącza)

Powiązanie <<include>>: wskazuje na wspólny fragment wielu przypadków użycia (wyłączony „przed nawias”); wykorzystywane w przebiegach podstawowych (operacje zawsze wykonywane).

Cele stosowania diagramu przypadków użycia jest są następujące: pozwalają na identyfikację oraz dokumentację wymagań;

- umożliwiają analizę obszaru zastosowań, dziedziny przedmiotowej;
- pozwalają na opracowanie projektu przyszłego systemu;
- stanowią przystępną i zrozumiałą platformę współpracy i komunikacji twórców systemu, inwestorów i właścicieli;
- są rodzajem umowy, kontraktu pomiędzy udziałowcami co do zakresu i funkcjonalności przyszłego systemu;
- stanowią podstawę testowania funkcji systemu na dalszych etapach jego cyklu życia.

Diagram przypadków użycia składa się z następujących kategorii pojęciowych:

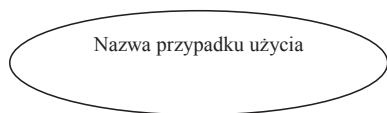
- przypadków użycia,
- aktorów,
- związków.

Przypadek użycia (*use case*) obejmuje specyfikację ciągu akcji i ich wariantów, które system (lub inna jednostka) może wykonać poprzez interakcję z aktorami tego systemu.

Można przyjąć, że przypadek użycia jest kompleksowym działaniem realizowanym w projektowanym systemie i pozwala na wykazanie działalności i aktywności aktora. Zakres danego działania wynika ze wszystkich wzajemnie powiązanych ze sobą przypadków użycia.

Przypadek użycia ma swoją niepowtarzającą się nazwę. Nazwa przypadku użycia to zwięzłe polecenie wykonania funkcji w projektowanym

systemie. Najczęściej jest to sformułowanie w trybie rozkazującym. Według standardu UML reprezentowany jest przez elipsę z etykietą wewnątrz.



**Rysunek 39.** Przypadek użycia

Aktor (*actor*) – to użytkownik przypadku użycia (często spójny zbiór ról odgrywanych przez użytkowników przypadków użycia w czasie interakcji z tym przypadkiem użycia).

Można wyróżnić aktorów:

- osobowych – aktorem osobowym mogą być: osoba, zespół, dział, instytucja, organizacja, zrzeszenie organizacji lub organizacja wirtualna. Nazwy aktorów osobowych często pokrywają się z nazwami funkcji, jakie pełnią w organizacji, projekcie lub przedsięwzięciu, bądź nazwą stanowiska, jakie piastują.
- nieosobowych – aktorem bezosobowym może być system zewnętrzny (podsystemy, bazy danych), urządzenie.

Nazwa aktora jest rzeczownikiem w liczbie pojedynczej. Należy pamiętać, że przyjęte dla aktora nazwy odzwierciedlają role odgrywane przez te obiekty, a nie indywidualne obiekty ze świata rzeczywistego.

Aktor (nieosobowy) – fortepian, baza danych, czas spektaklu, szpital, termin płatności, system rezerwacji pokoi, kiosk multimedialny itp.

Aktor użytkuje jeden lub wiele przypadków użycia w projektowanym systemie, natomiast przypadek użycia jest użytkowany przez jednego lub więcej aktorów. Interakcja aktorów z przypadkami użycia składa się z ich inicjowania, dostarczania danych, otrzymywaniu danych oraz użytkowania realizowanej przez przypadek użycia funkcjonalności.

## Związek

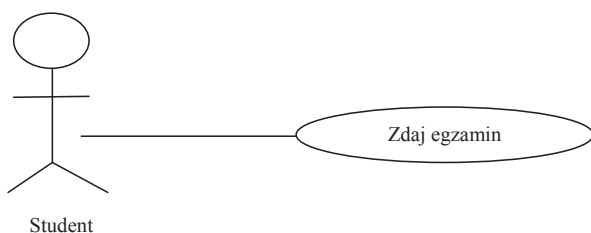
Związek (*relationship*) to określenie powiązania pomiędzy elementami modelu przypadków użycia. Jest to powiązanie pomiędzy aktorami i przypadkami użycia. Związki między aktorami a przypadkami użycia mogą być jedynie powiązaniem. Takie powiązanie oznacza, że aktor i przypadek użycia porozumiewają się ze sobą, wysyłając i odbierając

komunikaty. Każdy aktor, który jest na diagramie przypadków użycia, musi być bezpośrednio powiązany z co najmniej jednym przypadkiem użycia. Podobnie każdy przypadek użycia musi być użytkowany przez co najmniej jednego aktora (niejednokrotnie są to powiązania pośrednie).

Asocjacja (*association*) jest związkiem pomiędzy dwoma lub więcej klasyfikatorami, opisującym powiązanie pomiędzy ich instancjami. W diagramach przypadków użycia asocjacja wskazuje na komunikację dwukierunkową pomiędzy przypadkiem użycia a aktorem.

Uogólnienie (*generalization*) polega na tym, że pewien przypadek użycia może być szczególną odmianą innego, już istniejącego przypadku użycia. W praktyce może to być zarówno wygodny, jak i skuteczny sposób na przeniesienie wspólnych zachowań, ograniczeń i założeń szczególnych przypadków użycia do ogólnego przypadku użycia.

Aktorzy (czyli użytkownicy) systemu mogą być ludźmi lub zewnętrznymi systemami. W tym drugim przypadku warto je oznaczać, używając notacji prostokątnej. Każdy przypadek użycia jest powiązany z jednym lub wieloma aktorami. Ciągła linia ze strzałką oznacza, że pewien aktor jest szczególnym przypadkiem innego (szef sprzedaży jest także sprzedawcą).



**Rysunek 40.** Przykład komunikacji dwukierunkowej (asocjacja)

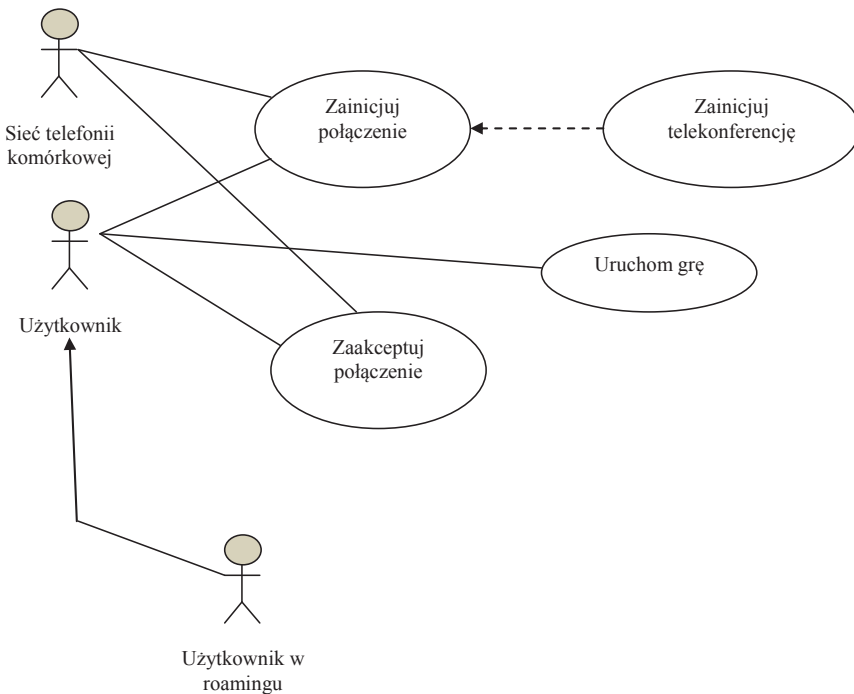
Relacja *include* między przypadkami użycia informuje nas o tym, że jeden przypadek użycia zawiera w sobie inny.

Modelując statyczne aspekty perspektywy przypadków użycia systemu, diagramy przypadków użycia będą wykorzystywane do dwóch celów.

- modelowanie otoczenia systemu – to zadanie polega między innymi na wyznaczeniu granicy wokół całego systemu i na wskazaniu leżących poza nią aktorów, którzy wchodzą w interakcję z systemem. Diagramy przypadków użycia służą w tym wypadku do zdefiniowania aktorów i znaczenia ich ról.

- modelowanie wymagań stawianych systemowi – to zadanie polega na określeniu, co system powinien robić (z punktu widzenia jego otoczenia) – niezależnie od tego, jak ma to zrobić. Diagramy przypadków użycia służą w tym wypadku do zdefiniowania oczekiwanego działania systemu. Można do systemu podejść jak do czarnej skrzynki – znane jest otoczenie i sposób porozumienia się z bytami leżącymi poza nim, ale nie to, co dzieje się w jego wnętrzu.

Model przypadków użycia dostarcza bardzo abstrakcyjnego spojrzenia na system – spojrzenia z pozycji aktorów, którzy go używają. Nie włącza zbyt wielu szczegółów, co pozwala wnioskować o funkcjonalności systemu na odpowiednio wysokim poziomie. Podstawowym (choć nie jedynym) zastosowaniem jest tu dialog z przyszłymi użytkownikami, zmierzający do sformułowania poprawnych wymagań na system.



**Rysunek 41.** Przykład diagramu przypadków użycia

## Diagram czynności (aktywności)

Diagram czynności służy do modelowania czynności i zakresu odpowiedzialności elementów bądź użytkowników systemu. Opisuje on działania związane z wieloma obiektami, pomiędzy którymi może występować komunikacja przy wykonywaniu czynności.

Elementy na diagramie czynności są przedstawiane następująco: początkowy i końcowy stan akcji – odpowiednio jako wypełnione koło oraz wypełnione koło w okręgu.



**Rysunek 42.** Początkowy i końcowy stan akcji

Stan akcji zawiera etykietę ją opisującą. Obrazowany jest za pomocą prostokąta z zaokrąglonymi narożnikami.

Przejście przepływu sterowania (ciągła strzałka) występuje pomiędzy czynnościami. Zakończenie jednej czynności powoduje rozpoczęcie drugiej.

Przejście przepływu obiektów (przerywana strzałka) – obiekt występuje pomiędzy aktywnościami, co oznacza, że jest otrzymywany na wyjściu pierwszej z nich, a pobierany na wejściu drugiej.

Tory pływackie rysowane są za pomocą linii ciągłych. Służą do określania, który element systemu wykonuje dane akcje.

Decyzje (obrazowane za pomocą rombów) służą do wyboru jednego przejścia przepływu sterowania. Odpowiednie wyjścia opisywane są warunkami, które muszą zostać spełnione, by dane przejście mogło zajść.

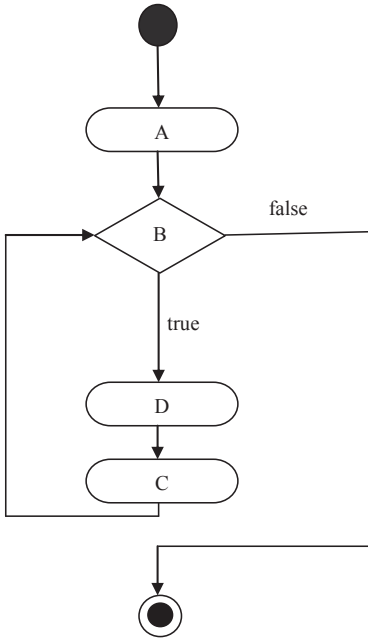
Współbieżność obrazowana jest za pomocą pogrubionej kreski i dzieli się na dwa elementy:

- synchronizacja sterowania – aby nastąpiło przejście (lub przejścia) wychodzące, muszą wystąpić wszystkie przejścia przychodzące;
- rozdzielenie sterowania – po zajściu przejść przychodzących występują jednocześnie wszystkie przejścia wychodzące.

Opcjonalnie możemy wyznaczyć elementy rozproszone, nadając im symbol :R przy przejściu w kolejny stan.

## Diagram interakcji

W języku UML diagram interakcji służy do opisu zależności przy przesyłaniu komunikatów pomiędzy obiektami, czyli zależności w przepływie sterowania pomiędzy obiektami.



**Rysunek 43.** Diagram czynności dla pętli for

Diagram interakcji ułatwia zrozumienie zależności w przepływie sterowania. Metody realizujące to sterowanie są rozproszone w wielu klasach, co powoduje trudności ze zrozumieniem ich wzajemnej zależności i interakcji. Jest to powód sporządzania diagramów interakcji. Diagramy interakcji służą do opisu zależności przy przesyłaniu komunikatów dla pewnej grupy obiektów. (Bardzo często stanowią one precyzyjny opis pojedynczego przypadku użycia).

Diagramy interakcji są jednym z rodzajów diagramów dynamicznych. Diagramy interakcji przyjmują jako bazę istniejący diagram klas i na jego podstawie opisują sposób współpracy ze sobą obiektów, których celem jest zrealizowanie konkretnej funkcji systemu lub nawet scenariusza danego przypadku użycia. Takie rozwiązanie pozwala na lepsze zrozumienie zdarzeń zachodzących pomiędzy obiektami. W ten sposób



na diagramie interakcji przedstawione jest zazwyczaj zachowanie systemu dotyczące jednego przypadku użycia.

Rodzaje diagramów interakcji są następujące:

- diagram sekwencji (diagram przebiegu) – przedstawia zależności czasowe pomiędzy obiektami, które służą do modelowania systemów czasu rzeczywistego oraz złożonych scenariuszy.
- diagram interakcji modeluje interakcje zachodzące w czasie działania systemu pomiędzy jego częściami, które to części wchodzi w skład widoku logicznego modelu.
- diagramy komunikacji – specyfikują strukturalne związki pomiędzy biorącymi udział w interakcji częściami oraz wymianę komunikatów pomiędzy tymi instancjami.

### **Diagram sekwencji**

Diagramy sekwencji opisują interakcje pomiędzy częściami systemu w postaci sekwencji komunikatów wymienianych między nimi. Obrazuje kolejność w czasie wysyłania komunikatów pomiędzy różnymi obiektami w systemie

### **Diagram współpracy (kolaboracji)**

Diagram współpracy (diagram współdziałania, diagram kolaboracji) odwzorowuje powiązania pomiędzy obiektami, służy do odwzorowywania efektów oddziaływania na pojedynczy obiekt (nie uwzględnia wpływu czasu,).

Diagram współpracy, komunikacji (*Communication diagram*) nazywany jest również diagramem kolaboracji. Można przyjąć, że jest to rozszerzona wersja diagramu współpracy, która przedstawia sposób wymiany informacji pomiędzy obiektami (aktorami, klasami), które wchodzi ze sobą w interakcję.

Oba typy ilustrują ten sam problem, jednak przedstawiają go w innym kontekście, dlatego wybór odpowiedniego diagramu zależy od sposobu, w jaki chcemy daną sytuację opisać. Diagram komunikacji jest jednym z diagramów interakcji. Określa związki strukturalne pomiędzy obiektami

biorącymi udział w interakcji zgodnie ze scenariuszem przypadków użycia.

Elementami diagramu komunikacji są:

- obiekty – aktorzy, moduły, klasy biorące udział w interakcji;
- asocjacje – określają strukturę organizacji obiektów, reprezentowane przez linie łączące obiekty;
- komunikaty – realizacje interakcji, opisywane etykietowanymi strzałkami

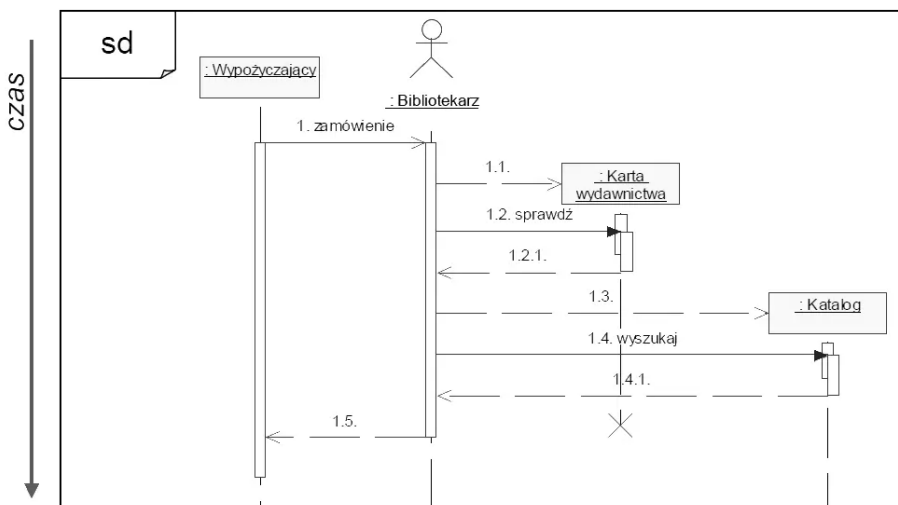
Każda etykieta komunikatu ma postać liczb oddzielonych kropkami, połączonych z krótkim opisem komunikatu. Liczby określają kolejność wysyłania komunikatów. Rozdzielenie sterowania oznaczone jest kropką i ponowioną numeracją komunikatów zgodnie z kolejnością w danym bloku sterowania. Diagram komunikacji używa etykiet do ustalania kolejności komunikatów oraz w zdecydowanie mniejszym stopniu niż diagram sekwencji określa aspekt czasu.

Cele stosowania diagramów komunikacji są następujące:

- określenie komunikatów wymienianych pomiędzy obiektami,
- wskazanie relacji pomiędzy obiektami,
- ustalenie kolejności wysyłania komunikatów,
- ułatwienie zrozumienia interakcji.

W pierwszym kroku tworzenia diagramu komunikacji należy określić, jakie obiekty wejdą w jego skład, a następnie zdefiniować związki pomiędzy nimi. W kolejnym kroku wprowadzane są symbole (opisy wraz ze strzałkami) komunikatów zapisywane równoległe do linii asocjacji bądź w poziomie. Strzałka powinna wskazywać kierunek przepływu komunikatu. Jako ostatni element na diagram nanoszona jest numeracja kolejności komunikatów. Po zakończeniu tworzenia diagramu każdy komunikat musi mieć numer swojej kolejności, nazwę operacji oraz strzałkę ustalającą kierunek przepływu. W przypadku dużej liczby komunikatów płynących w tym samym kierunku i pomiędzy tymi samymi obiektami dozwolone jest używanie jednej strzałki z kilkoma etykietami. Dodatkowo można oznaczać opisem bądź typem strzałki, z jakim rodzajem komunikatu mamy do czynienia

Diagramy sekwencji w sposób intuicyjny prezentują kolejność wywołań operacji, przepływ sterowania pomiędzy obiektami oraz szablon realizowanego algorytmu. Pomijają natomiast całkowicie aspekt dostępu i operacji na danych, związany z komunikacją.



**Rysunek 44.** Diagram sekwencji

Białe prostokąty umieszczone na linii życia obiektu oznaczają, że obiekt jest zajęty wykonywaniem pewnej czynności (natomiast nie mają bezpośredniego związku z istnieniem obiektu).

Uczestnikami diagramów sekwencji są obiekty, opisane nazwą obiektu i jego klasą, które wymieniają między sobą komunikaty. Diagram sekwencji jest zapisany w prostokącie oznaczonym operatorem sd (od angielskiej nazwy diagramu) i składa się z pionowych linii życia (*lifelines*) poszczególnych obiektów uczestniczących w interakcji oraz wymieniających między nimi komunikatów (wywołań operacji).

Czas jest reprezentowany w postaci pionowej osi diagramu. Linia życia obiektu to czas, w którym konkretna instancja obiektu jest w stanie przyjmować komunikaty i wysyłać je. Obejmuje ona czas istnienia obiektu w systemie.

Obiekt jest tworzony przez wysłanie do niego komunikatu-konstruktor (Bibliotekarz tworzy obiekt klasy Karta Wydawnictwa), natomiast niekoniecznie jest fizycznie usuwany na końcu linii życia (przestaje być istotny). Fizyczne usunięcie obiektu można wprost oznaczyć jako znak X na linii życia (na przykład obiekt Karta wydawnictwa).

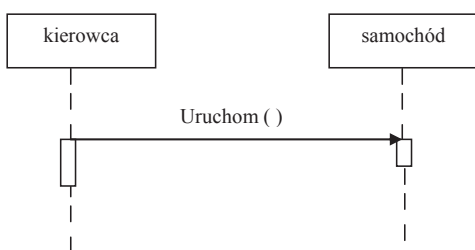
Przykładowe rodzaje komunikatów zwyczajowo przesyłanych na diagramie:

- wywołanie procedury,

- powrót z wywołania,
- wywołanie asynchroniczne,
- komunikat tworzenia uczestnika,
- komunikat usuwania uczestnika.

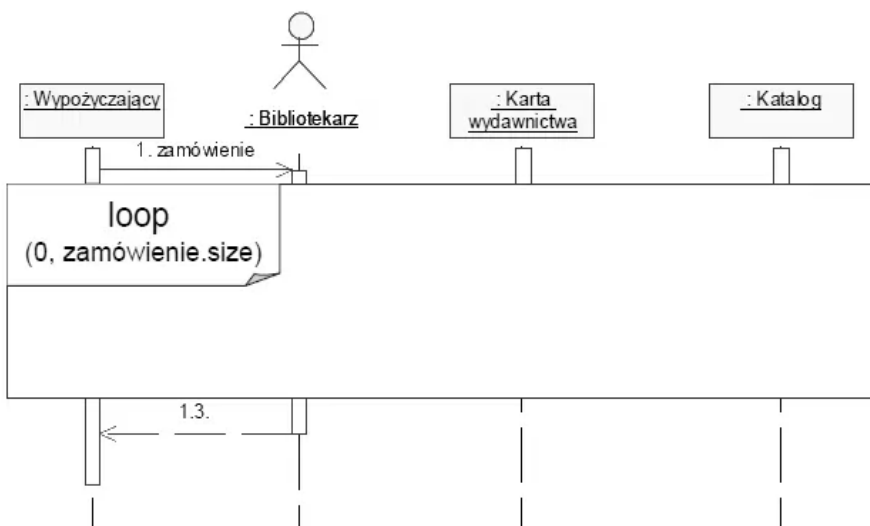
Komunikat jest formą kontaktu pomiędzy obiektami, którego efektem ma być podjęcie przez docelowy obiekt pewnej akcji. Otrzymanie komunikatu przez obiekt wiąże się z wykonaniem przez niego jego własnego kodu lub wysłaniem kolejnego komunikatu do innego obiektu w celu wykonania przez niego pewnej akcji.

Komunikaty w UML są reprezentowane przez strzałki łączące linie życia poszczególnych obiektów. Każdy komunikat wewnątrz interakcji opatrzony jest kolejnym numerem, co pozwala na łatwe śledzenie jej przebiegu. Istnieją trzy podstawowe komunikaty, jakie mogą zostać wymienione pomiędzy obiektami: wywołanie procedury, powrót z niej oraz wywołanie asynchroniczne.



**Rysunek 45.** Blok – fragmenty wyodrębnione

Blok definiuje grupę komunikatów wspólnie mającą pewną właściwość. Bardzo często zachodzi konieczność wskazania specjalnej własności pewnej części interakcji, np. oznaczenie sekcji krytycznej czy zwyczajnej pętli. Na diagramach sekwencji taką grupę operacji obejmuje się prostokątem, w którego lewym górnym narożniku, w pięciokącie umieszcza się słowo kluczowe lub opis określający znaczenie danego bloku (tzw. operator interakcji).

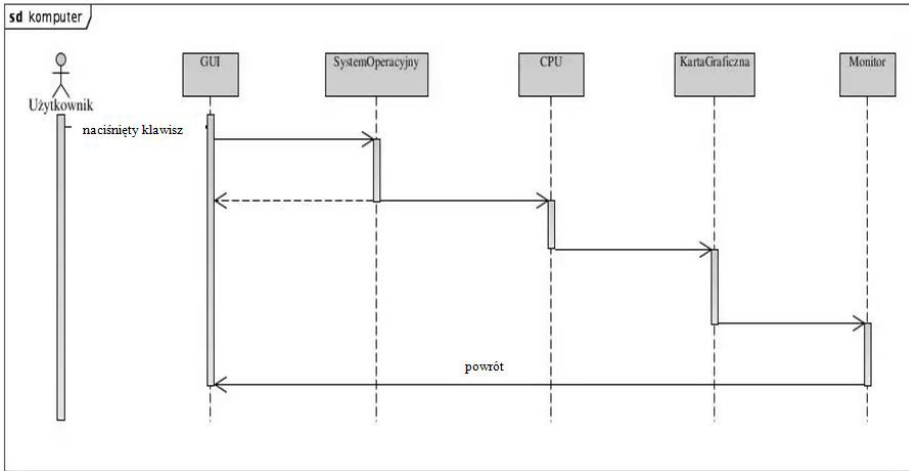


**Rysunek 46.** Grupa komunikatów – blok

Operatory interakcji najczęściej używane:

- alt – określający warunek wykonania bloku operacji, odpowiadający instrukcji if-else; warunek umieszcza się wówczas wewnątrz bloku w nawiasach kwadratowych;
- opt – reprezentujący instrukcję if (bez else );
- par – nakazujący wykonać operacje równolegle;
- critical – blok atomowy, oznaczający obszar krytyczny;
- loop – definiujący pętlę typu for (o określonej z góry liczbie iteracji) lub while (wykonywanej dopóty, dopóki pewien warunek jest prawdziwy);
- break – wykonanie fragmentu i zakończenie interakcji;
- seq – słaba sekwencja (podobnie do współbieżności) dotyczy zdarzeń z kilku linii;
- ignore/consider – ignore(komunikat1, komunikat2, ...) oznacza, że na diagramie nie pokazano wymienionych komunikatów, choć mogą wystąpić. Consider = odwrotnie.

Aktorzy na diagramach interakcji – po lewej stronie diagramu można dodawać aktora inicjującego dany ciąg działań. Można również narysować jego linię życia.



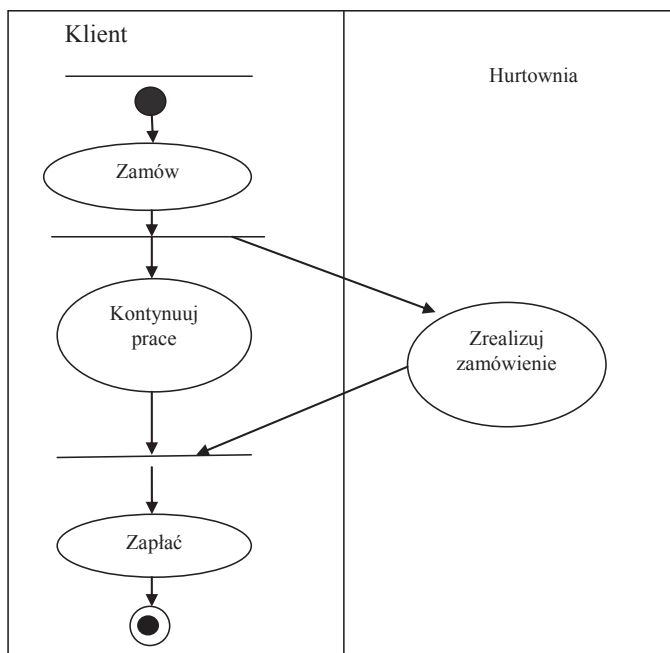
**Rysunek 47.** Aktor na diagramie inicjujący ciąg działań

## Diagram stanów

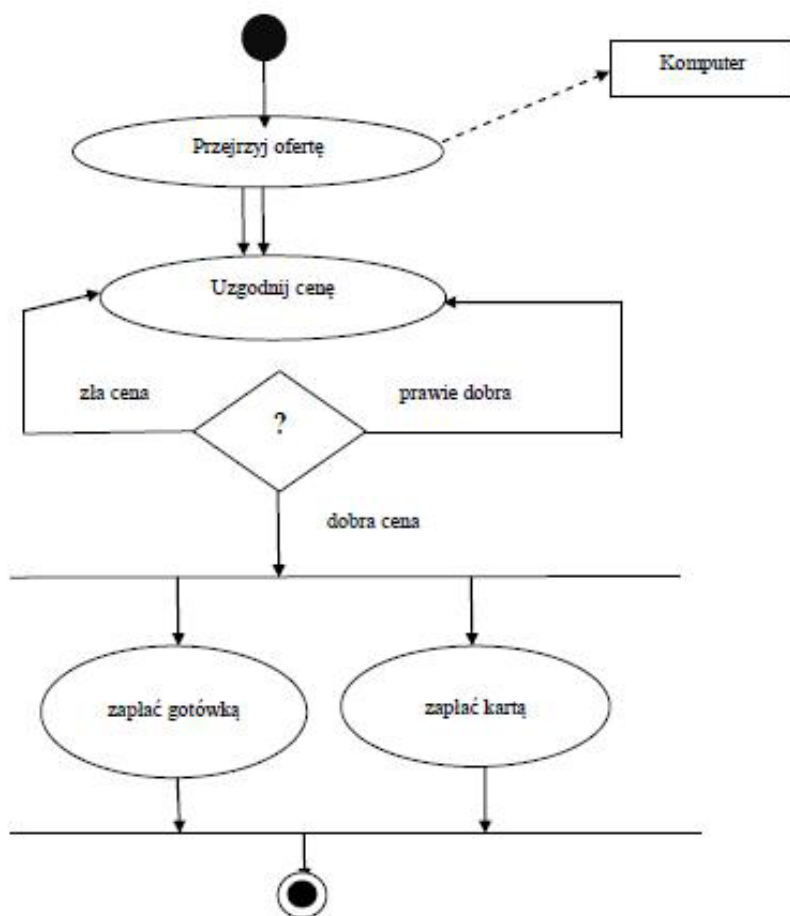
Diagram stanów (*Statechart Diagram*) to diagram, który jest używany na etapie projektowanego systemu. Diagram stanów pokazuje, jakie możliwe stany może przyjąć wybrany obiekt. Na tym diagramie pokazywane są również elementy (przejścia), które powodują zmianę tego stanu. Na diagramie stanów znajdują się sekwencje sygnałów (danych) wejściowych, które powodują przejście systemu w dany stan. Przejściu systemu w inny stan muszą towarzyszyć jakieś działania (akcje), które są podejmowane w odpowiedzi na działanie określonych stanów wejściowych.

Można przyjąć, że w ramach diagramu stanów tworzony jest cykl życia wybranego obiektu. Diagram stanów jest przydatny w modelowaniu zachowania interfejsów, klas i kooperacji. Diagram stanów nazywany jest w literaturze jak maszyna stanowa, która obrazuje reakcje obiektów na ciągu zdarzeń. Interakcje służą do modelowania zachowania zestawów obiektów. Maszyna stanowa służy do modelowania zachowania jednego obiektu (tym obiektem może być klasa, ale może być też cały system). Maszynę stanową można wykorzystać do pokazania na diagramie stanów – dopuszczalne stany i przejścia między nimi lub (na diagramie czynności) pokazania przepływu sterowania między czynnościami.

Tory wskazują umiejscowienie czynności. Występujące na diagramie czynności są podzielane pionowymi, ciągłymi liniami. Każdy tor ma unikatową nazwę.

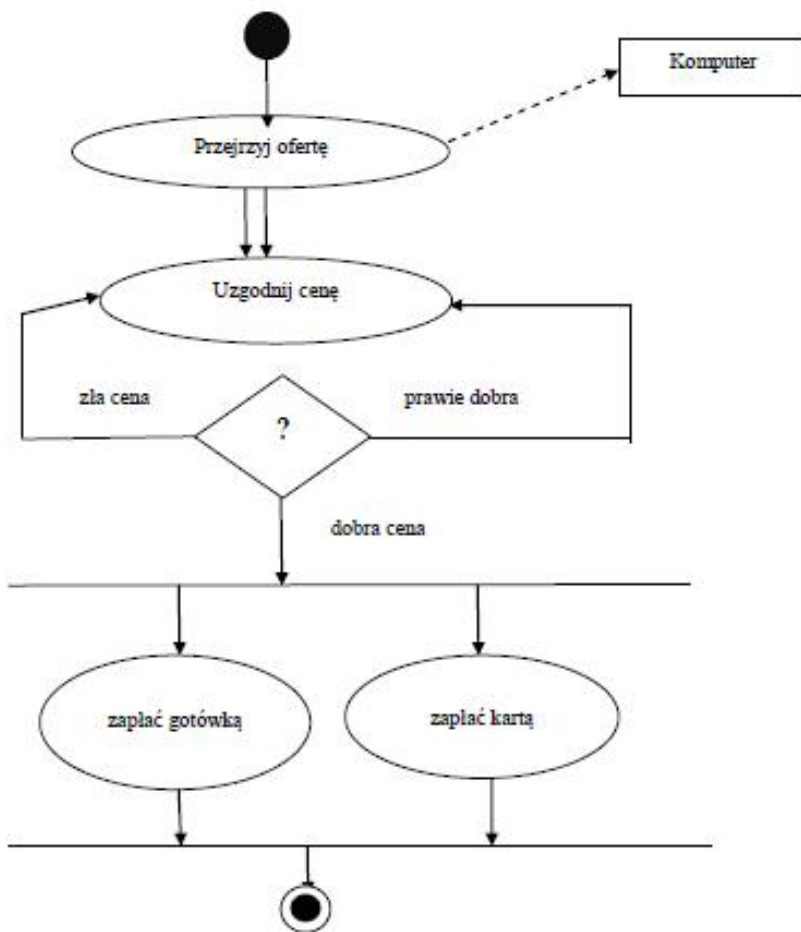


**Rysunek 48.** Diagram czynności – przykład z torami



Rysunek 49. Przykładowy diagram stanów – zakup komputera



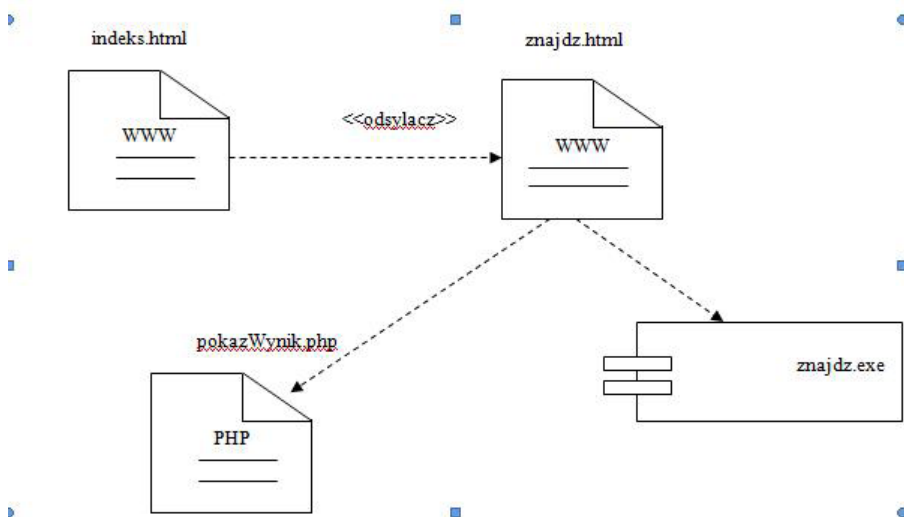


Rysunek 50. Przykładowy diagram stanów – sterownik grzania

### Diagram komponentów

Diagram komponentów (*Component Diagram*) pokazuje uporządkowane komponenty w systemie. Diagram komponentów wiąże się z diagramem klas (zwykle każdemu komponentowi są przyporządkowane pewne klasy, interfejsy i kooperacje). Komponent to wymienialny, wykonywalny fragment systemu, z ukrytymi szczegółami implementacyjnymi (np. plik, podprogram itp.). Przedstawia fizyczne elementy wchodzące w skład systemu i połączenia między nimi.

Węzły są to zasoby sprzętowe dostępne podczas działania systemu. Obrazowane są za pomocą prostopadłościów.



**Rysunek 51.** Diagram komponentów strony WWW

Na diagramach komponentów uwzględnia się elementy fizyczne (programy, biblioteki, tabele) instalowane na węzłach.

Diagramy komponentów służą do obrazowania statycznych aspektów perspektywy implementacyjnej. Diagramy komponentów (*component diagram*) pokazują podział systemów programowych na mniejsze podsystemy.

Komponent udostępnia zestaw interfejsów, może też wymagać pewnych interfejsów do funkcjonowania.

Funkcjonalność oferowana przez komponent jest dostępna przez interfejsy, które implementuje. Podsumowując, diagram komponentów służy do pokazania związków pomiędzy komponentami i interfejsami.

W języku 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;

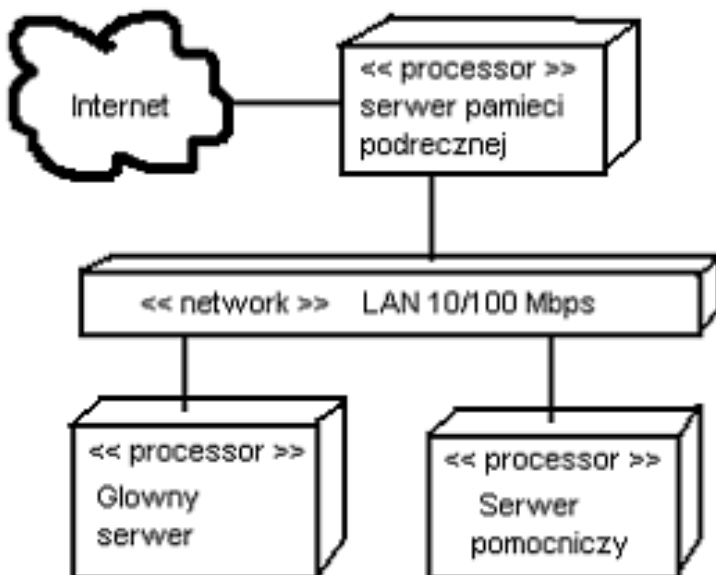
- *document* – określa komponent reprezentujący dokument;

## Diagram wdrożenia

Diagram wdrożenia (*Deployment Diagram*) pokazuje połączenie i konfigurację węzłów działających w czasie wykonania i zainstalowane na nich komponenty.

Diagram wdrożenia odnosi się do statycznych aspektów wdrożeń. Ten diagram jest powiązany z diagramem komponentów (zwykle każdy węzeł zawiera chociaż jeden komponent).

Diagramy wdrożenia wykorzystywane są do modelowania systemów rozproszonych i typu klient-serwer.



Rysunek 52. Przykładowy diagram wdrożenia

## 8. SŁOWNIK DANYCH (*DATA DICTIONARY*)

Słownik danych to repozytorium (magazyn danych, składnica danych) wszystkich pojęć zdefiniowanych w projekcie. Słownik danych jest zorganizowaną listą elementów systemu, obejmującą ich definicje. Zawiera dokładne definicje wszystkich elementów każdego z diagramów, a także wzajemne powiązania pomiędzy diagramami (modelami).

Każdy element ma unikatową nazwę i krótki opis. Każda dana elementarna ma zdefiniowane nazwy stosowane zamiennie (aliasy), typ, format, zakres wartości oraz prawa dostępu. Każda dana złożona ma określoną listę atrybutów elementarnych. Magazyny danych mają zdefiniowaną zawartość, określone są ich źródła i przeznaczenie.

Zalety stosowania słowników:

- zbieranie i przechowywanie informacji o procesach i danych w jednym miejscu;
- uzgadnianie przez analityków, projektantów oraz użytkowników znaczenia pozycji w słowniku;
- ułatwienie komunikacji z użytkownikiem;
- sprawdzenie redundancji i niezgodności nazw dla poszczególnych danych (elementów słownika);
- przeprowadzanie modyfikacji w poszczególnych elementach i badanie wpływu tych zmian na pozostałe;
- określenie praw dostępu do poszczególnych elementów.

Słownik danych tworzy się po to, by użytkownik miał pełną dokumentację zawartych w projekcie obiektów (wraz z ich definicjami) oraz opisy wszystkich wejść, wyjść, elementów składów danych, elementów obiektów. W słowniku danych zawiera się również wszystkie pośrednie formuły obliczeń zastosowane w projekcie.

W skład słownika danych wchodzi również:

- opisy złożonych agregatów danych zawartych w składach danych,
- szczegółowe opisy relacji pomiędzy obiektami zawartych na diagramach.

W notacji dla słownika danych używa się następujących symboli:

=	[składa się z]
---	----------------

+	i
()	opcja
{ }	iteracja
	wybranie jednej z kilku możliwości
**	komentarz (tekst komentarza zawarty jest pomiędzy tymi znakami)
@	identyfikator (pole kluczowe – dla opisu składu lub obiektu)
	oddziela alternatywne wybory w konstrukcji [ ]

## Przykład

samochód = marka + kolor + pojemność + liczba\_drzwi

autor = [kowalski | zieleński | myszkorowski]  
 wydawnictwo = [mikom | WNT | Nasza\_księgarnia]  
 rok\_wydania = [1997 | 2000 | 2004]  
 nakład = [1000 | 500 ]  
 autor = {dowolny\_znak\_autor}  
 wydawnictwo = {dowolny\_znak\_wydawnictwo}  
 rok\_wydania = {dowolny\_znak\_rok\_wydania}  
 nakład = {dowolny\_znak\_nakład}  
 dowolny\_znak\_autor = [A-Z | a-z | \* ]  
 dowolny\_znak\_wydawnictwo = [A-Z | a-z | 0-9]  
 dowolny\_znak\_rok\_wydania = [0.0 -9.0 ]  
 dowolny\_znak\_nakład = [D-K | 1000-5000 ]

Definicje elementu danych wprowadza się poprzez symbol znaku równości lub przez komentarz tekstowy. Dla komentarza tekstowego należy podać dziedzinę wartości.

W słowniku danych powinny się znaleźć opisy wszystkich elementów użytych na diagramach.

## 9. LITERATURA

Bliźniuk G., *Metody szacowania wartości wybranych miar jakości oprogramowania*, Warszawa 1999.

Dumnicki R., Kasprzyk A., Kozłowski M., *Analiza i projektowanie obiektowe*, Gliwice 2010.

Gamma E. i in., *Wzorce projektowe: elementy oprogramowania obiektowego wielokrotnego użytku*, Warszawa 2005.

<http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-komponentow,1,17.html>, data dostępu: 2013.

Larman C., *UML i wzorce projektowe. Analiza i projektowanie obiektowe oraz iteracyjny model wytwarzania aplikacji*, Gliwice 2011.

Shalloway A., Trott J. R., *Programowanie zorientowane obiektowo. Wzorce projektowe*, Gliwice 2005.

Subieta K., *Perspektywy obiektowych baz danych*, cz. I i cz. II, <http://www.ipipan.waw.pl/~subieta> 1998, data dostępu: 2013.

Subieta K., Wprowadzenie do obiektowych metodyk projektowania i notacji UML, <http://www.ipipan.waw.pl/~subieta>, UML PTI Szczyrk 1999

Wirfs-Brock R., McKean A., *Projektowanie obiektowe. Role, odpowiedzialność i współpraca*, Gliwice 2012.

[www.agh.edu.pl/uczelnia/tad/PSI/Obiektowe.ppt](http://www.agh.edu.pl/uczelnia/tad/PSI/Obiektowe.ppt), data dostępu: sierpień 2013

## 10. SPIS RYSUNKÓW

<b>RYSUNEK 1.</b>	CYKL ŻYCIA SYSTEMU INFORMATYCZNEGO .....	10
<b>RYSUNEK 2.</b>	PRZYKŁAD – AKTOR JEST ABSTRAKCYJNYM UŻYTKOWNIKIEM SYSTEMU.....	18
<b>RYSUNEK 3.</b>	PRZYKŁAD SYMBOLU PRZYPADKU UŻYCIA .....	19
<b>RYSUNEK 4.</b>	PRZYKŁAD DIAGRAMU PRZYPADKÓW UŻYCIA (AKTOR OBIEKT).....	19
<b>RYSUNEK 5.</b>	DIAGRAM PRZYPADKÓW UŻYCIA – SYSTEM OBSŁUGI SKLEPU KOMPUTEROWEGO .....	20
<b>RYSUNEK 6.</b>	PRZYKŁADY RÓŻNYCH OBIEKTÓW .....	21
<b>RYSUNEK 7.</b>	PRZYKŁAD RELACJI POWIĄZANIA MIĘDZY OBIEKTAMI.....	22
<b>RYSUNEK 8.</b>	DIAGRAM INTERAKCJI OBIEKTÓW .....	23
<b>RYSUNEK 9.</b>	PRZYKŁAD DIAGRAMU HIERARCHII PROCESÓW .....	24
<b>RYSUNEK 10.</b>	PRZYKŁAD MODELU OBIEKTOWEGO OPISU SYSTEMU .....	27
<b>RYSUNEK 11.</b>	OZNACZENIE KLASY .....	29
<b>RYSUNEK 12.</b>	OZNACZENIE KLASY AKTYWNEJ.....	29
<b>RYSUNEK 13.</b>	PRZYKŁADOWA KLASA (ZAZNACZONY ATRYBUT WIDOCZNOŚĆ KLASY)	30
<b>RYSUNEK 14.</b>	POKAZANO LICZEBNOŚĆ KLASY (TRANSAKCJE BANKOWE – TRZY EGZEMPLARZE).....	30
<b>RYSUNEK 15.</b>	KLASA ZE ZDEFINIOWANYMI PARAMETRAMI .....	31
<b>RYSUNEK 16.</b>	KLASA „STUDENT” ORAZ OBIEKTY KLASY „KOWALSKI” I OBIEKT ANONIMOWY „STUDENT”.....	31
<b>RYSUNEK 17.</b>	OBIEKT „NAUCZYCIEL” I ZOBOWIĄZANIA (NOTATKA).....	32
<b>RYSUNEK 18.</b>	PRZYPADEK UŻYCIA – SYMBOL.....	33
<b>RYSUNEK 19.</b>	SYMBOL KOOPERACJI .....	33
<b>RYSUNEK 20.</b>	STAN OBIEKTU – SYMBOL .....	33
<b>RYSUNEK 21.</b>	PRZEJŚCIA Z JEDNEGO DO DRUGIEGO STANU.....	34
<b>RYSUNEK 22.</b>	PRZEJŚCIA Z JEDNEGO DO DRUGIEGO STANU.....	34
<b>RYSUNEK 23.</b>	WYGLĄD NOTATKI .....	35
<b>RYSUNEK 24.</b>	WĘZEL SYSTEMU .....	35
<b>RYSUNEK 25.</b>	POWIĄZANIE KLAS .....	36
<b>RYSUNEK 26.</b>	ZWIĄZEK AGREGACJI KLAS.....	36
<b>RYSUNEK 27.</b>	ZWIĄZEK KLAS – AGREGACJA CAŁKOWITA.....	37
<b>RYSUNEK 28.</b>	SYMBOL POWIĄZANIA.....	37
<b>RYSUNEK 29.</b>	WIĄZANIE KLAS .....	37
<b>RYSUNEK 30.</b>	ZWIĄZEK UŻYCIA (A KORZYSTA Z B) .....	38
<b>RYSUNEK 31.</b>	RELACJA ZALEŻNOŚCI.....	38
<b>RYSUNEK 32.</b>	ZWIĄZEK UOGÓLNIENIE .....	39
<b>RYSUNEK 34.</b>	ZWIĄZEK REALIZACJA ZALEŻNOŚCI .....	39

<b>RYSUNEK 35.</b> ZWIĄZEK 1:1 .....	41
<b>RYSUNEK 36.</b> ZWIĄZEK WIELE DO JEDEN .....	41
<b>RYSUNEK 37.</b> ZWIĄZEK WIELE DO WIELU .....	41
<b>RYSUNEK 38.</b> ZWIĄZEK WNIOSKOWANIE .....	42
<b>RYSUNEK 39.</b> PRZYPADEK UŻYCIA .....	44
<b>RYSUNEK 40.</b> PRZYKŁAD KOMUNIKACJI DWUKIERUNKOWEJ (ASOCJACJA).....	45
<b>RYSUNEK 41.</b> PRZYKŁAD DIAGRAMU PRZYPADKÓW UŻYCIA .....	46
<b>RYSUNEK 42.</b> POCZĄTKOWY I KOŃCOWY STAN AKCJI.....	47
<b>RYSUNEK 43.</b> DIAGRAM CZYNNOŚCI DLA PĘTLI FOR.....	48
<b>RYSUNEK 44.</b> DIAGRAM SEKWENCJI .....	51
<b>RYSUNEK 45.</b> BŁOK – FRAGMENTY WYODRĘBNIONE .....	52
<b>RYSUNEK 46.</b> GRUPA KOMUNIKATÓW – BŁOK .....	53
<b>RYSUNEK 47.</b> AKTOR NA DIAGRAMIE INICJUJĄCY CIĄG DZIAŁAŃ .....	54
<b>RYSUNEK 48.</b> DIAGRAM CZYNNOŚCI – PRZYKŁAD Z TORAMI.....	55
<b>RYSUNEK 49.</b> PRZYKŁADOWY DIAGRAM STANÓW – ZAKUP KOMPUTERA.....	56
<b>RYSUNEK 50.</b> PRZYKŁADOWY DIAGRAM STANÓW – STEROWNIK GRZANIA .....	57
<b>RYSUNEK 51.</b> DIAGRAM KOMPONENTÓW STRONY WWW .....	58
<b>RYSUNEK 52.</b> PRZYKŁADOWY DIAGRAM WDROŻENIA .....	59





