Sprawozdanie

Architektura i projektowanie systemów komputerowych

Temat: Diagram przypadków użycia - modelowanie w UML

Adam Oporski, Kajetan Lach

Uniwersytet Gdański

Kierunek: Informatyka Praktyczna

Gdańsk, 13 listopada 2025

Spis treści

Spis rysunków

Rozdział 1

Diagram klas

Modelowanie klas jest fundamentem projektowania obiektowego. Pozwala ono na statyczne przedstawienie struktury systemu poprzez zdefiniowanie bytów (klas), za które system jest odpowiedzialny, oraz relacji (powiązań) zachodzących między nimi. Diagram klas jest centralnym artefaktem w języku UML, służącym jako plan architektoniczny dla kodu źródłowego.

1.1 Diagram klas - notacja i semantyka

Diagram klas (Class Diagram) to statyczny diagram strukturalny, który opisuje budowę systemu poprzez pokazanie jego klas, ich atrybutów, operacji (metod) oraz relacji (powiązań) między tymi klasami. Jest to najczęściej używany i prawdopodobnie najważniejszy diagram UML w kontekście programowania obiektowego (OOP).

Graficznie klasa jest reprezentowana jako prostokąt, zazwyczaj podzielony na trzy sekcje:

- **Nazwa klasy:** Górna sekcja, zawiera nazwę (np. "KontoBankowe").
- **Atrybuty (Attributes):** Środkowa sekcja, zawiera pola lub właściwości klasy (np. "saldo : Pieniadze").
- **Operacje (Operations):** Dolna sekcja, zawiera metody lub funkcje, jakie klasa udostępnia (np. "wplac(kwota : Pieniadze)").

Diagram klas pokazuje nie tylko pojedyncze klasy, ale przede wszystkim sposób, w jaki łączą się one w spójny system za pomocą różnych typów relacji (asocjacji, agregacji, kompozycji, dziedziczenia).

1.2 Zastosowanie diagramu klas

Diagram klas jest wszechstronnym narzędziem wykorzystywanym na różnych etapach cyklu życia oprogramowania i do różnych celów:

- **Analiza domeny (Domain Modeling):** We wczesnej fazie projektu diagram klas służy do modelowania kluczowych pojęć (bytów) z dziedziny problemu (np. w systemie bankowym będą to "Klient", "Konto", "Transakcja"). Pomaga to analitykom i deweloperom zrozumieć świat biznesu.
- **Projektowanie systemu (System Design):** Jest to podstawowe zastosowanie. Deweloperzy używają diagramów klas do projektowania architektury oprogramowania. Decydują, jakie klasy będą potrzebne, jakie będą miały atrybuty i operacje oraz jak będą ze sobą współpracować.
- **Generowanie kodu (Code Generation):** Wiele narzędzi typu CASE (Computer-Aided Software Engineering) potrafi automatycznie wygenerować szkielety kodu (np. w Javie, C#, C++) bezpośrednio z diagramu klas.
- **Inżynieria wsteczna (Reverse Engineering):** Narzędzia potrafią również analizować istniejący kod źródłowy i generować z niego diagramy klas. Jest to niezwykle przydatne do zrozumienia i dokumentowania starszych (legacy) systemów.
- **Dokumentacja techniczna:** Diagram klas stanowi precyzyjny i jednoznaczny "plan"systemu, który jest łatwiejszy do zrozumienia niż przeglądanie tysięcy linii kodu. Służy jako kluczowy element dokumentacji architektonicznej.
- **Komunikacja w zespole:** Diagramy te stanowią wspólny język wizualny dla programistów, projektantów i analityków, ułatwiając dyskusje na temat struktury i odpowiedzialności poszczególnych modułów systemu.

1.3 Atrybuty diagramu klas

Atrybuty diagramu klas to elementy, które opisują klasę. Są one reprezentowane jako prostokąty, zazwyczaj podzielone na trzy sekcje:

- **Nazwa atrybutu:** Górna sekcja, zawiera nazwę atrybutu (np. "saldo").
- **Typ atrybutu:** Środkowa sekcja, zawiera typ atrybutu (np. "Pieniadze").
- **Dostępność:** Dolna sekcja, zawiera informację o dostępności atrybutu (np. "public").

1.4 Podstawowe relacje stosowane w diagramie klas

Podstawowe relacje stosowane w diagramie klas to:

- **Asocjacja (Association):** Relacja między dwiema klasami, która wskazuje, że obiekty jednej klasy są powiązane z obiektami drugiej klasy.
- **Agregacja (Aggregation):** Relacja między dwiema klasami, która wskazuje, że obiekty jednej klasy są składowymi obiektów drugiej klasy.
- **Kompozycja (Composition):** Relacja między dwiema klasami, która wskazuje, że obiekty jednej klasy są składowymi obiektów drugiej klasy i że obiekt drugiej klasy nie może istnieć bez obiektu pierwszej klasy.
- **Dziedziczenie (Inheritance):** Relacja między dwiema klasami, która wskazuje, że klasa potomna dziedziczy atrybuty i operacje klasy nadrzędnej.

1.5 Dziedziczenie

Dziedziczenie (Inheritance) to relacja między dwiema klasami, która wskazuje, że klasa potomna dziedziczy atrybuty i operacje klasy nadrzędnej.

Graficznie dziedziczenie jest reprezentowane jako strzałka skierowana od klasy potomnej do klasy nadrzędnej.

Rozdział 2

System aukcyjny: wymagania, struktura i działanie

2.1 Opis problemu i kontekst

System aukcyjny to platforma internetowa umożliwiająca użytkownikom wystawianie, licytowanie i kupowanie przedmiotów w ramach aukcji online. Głównym celem systemu jest zapewnienie bezpiecznej i przejrzystej przestrzeni transakcyjnej pomiędzy sprzedającymi i kupującymi. System wspiera różne typy użytkowników – obserwatorów, uczestników aukcji oraz administratorów – oferując im odpowiedni zakres funkcjonalności i poziom uprawnień.

2.2 Wymagania funkcjonalne

Obserwator

- System umożliwia założenie konta w serwisie aukcyjnym.
- System umożliwia przeglądanie aktywnych aukcji.

Uczestnik aukcji

- System umożliwia wystawienie towaru na aukcję.
- System umożliwia przeglądanie historii zawartych transakcji.
- System umożliwia licytowanie towarów w ramach aktywnych aukcji.
- System umożliwia finalizację transakcji po zakończeniu aukcji i wyłonieniu zwycięzcy.

Administrator

- System umożliwia zarządzanie serwisem.
- System umożliwia zarządzanie kontami użytkowników.
- System umożliwia zarządzanie aukcjami.
- System umożliwia zarządzanie kategoriami towarów (CRUD).
- System umożliwia finalizację transakcji w przypadku, gdy użytkownik wygra aukcję.

2.3 Wymagania niefunkcjonalne

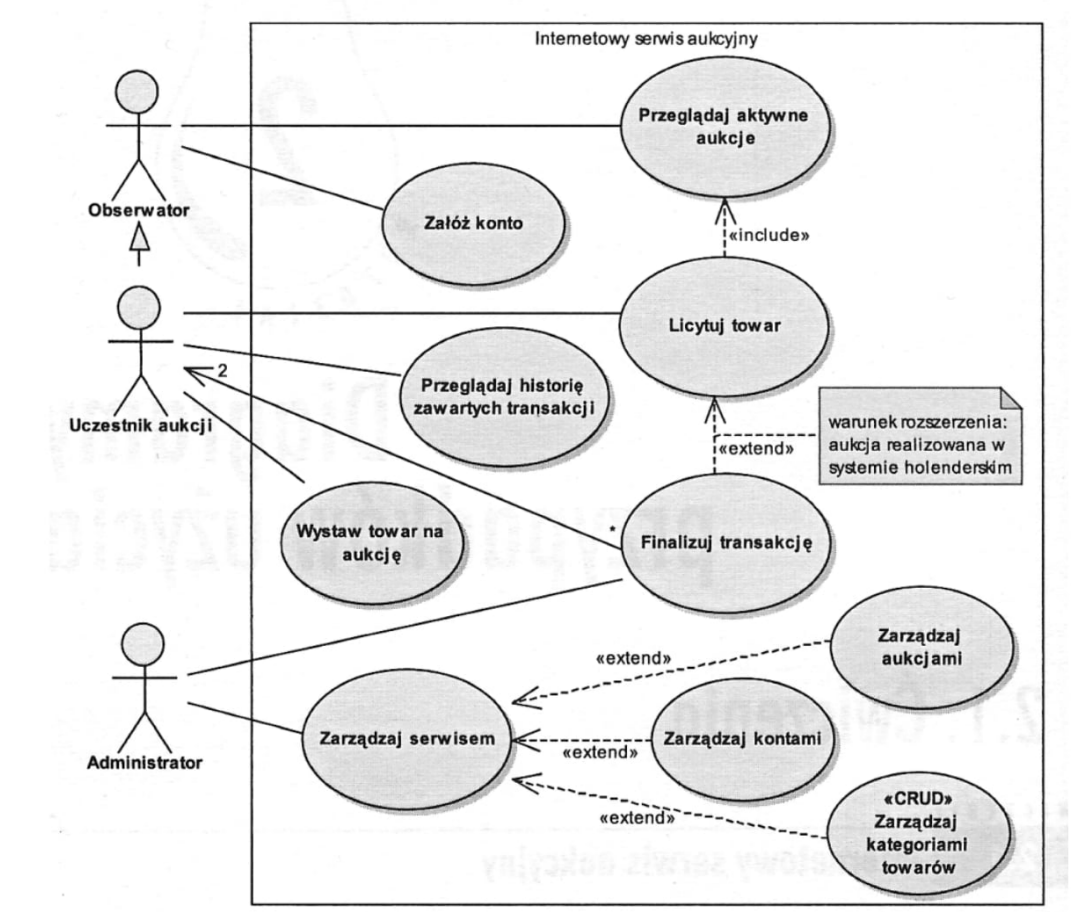
- System powinien umożliwiać obsługę dużej liczby jednoczesnych użytkowników.
- Czas odpowiedzi interfejsu użytkownika nie powinien przekraczać 2 sekund.
- System musi zapewniać bezpieczeństwo danych użytkowników (szyfrowanie haseł, certyfikaty SSL).
- Architektura systemu powinna umożliwiać łatwe skalowanie oraz aktualizację komponentów.
- System powinien być dostępny 24/7 z gwarancją dostępności na poziomie minimum 99,9%.

2.4 Struktura systemu (architektura wysokiego poziomu)

System aukcyjny został zaprojektowany w architekturze wielowarstwowej, obejmującej:

- **Warstwę prezentacji** – interfejs webowy oraz API REST dla klientów mobilnych.
- **Warstwę logiki biznesowej** – serwisy odpowiedzialne za obsługę aukcji, kont użytkowników, płatności i powiadomień.
- **Warstwę danych** – relacyjną bazę danych przechowującą informacje o użytkownikach, aukcjach, ofertach i transakcjach.
- **Warstwę bezpieczeństwa** – system autoryzacji i uwierzytelniania użytkowników.

DIAGRAMY PRZYPADKÓW UŻYCIA



Rys. 1. Internetowy serwis aukcyjny

Rysunek 2.1: Diagram przypadków użycia dla systemu aukcyjnego.

2.5 Opis działania (przepływy)

- **Rejestracja użytkownika:** Obserwator wypełnia formularz rejestracyjny, a system tworzy nowe konto użytkownika.
- **Wystawienie aukcji:** Uczestnik aukcji dodaje nowy przedmiot poprzez formularz, wprowadza cenę minimalną, zdjęcie i opis. Aukcja jest zapisywana w bazie danych i publikowana w systemie.
- **Licytacja towaru:** Uczestnik aukcji wybiera interesujący przedmiot i składa ofertę. System porównuje kwoty i aktualizuje najwyższą ofertę.
- **Finalizacja transakcji:** Po zakończeniu aukcji system automatycznie identyfikuje zwycięzcę i rozpoczyna proces płatności oraz potwierdzenia transakcji.

- **Zarządzanie serwisem:** Administrator nadzoruje działanie systemu, może usuwać nieaktywne aukcje, zarządzać kontami użytkowników i kategoriami towarów.

Rozdział 3

Projekt *Czytelnia*: funkcjonalności i diagram

3.1 Opis systemu

System *Czytelnia* (system biblioteczny) umożliwia użytkownikom dostęp do katalogu materiałów bibliotecznych w formie papierowej i cyfrowej, wspiera proces rezerwacji, wypożyczania i zarządzania zbiorami, a także automatyczne powiadomienia o terminach zwrotów. System rozróżnia dwa typy użytkowników: czytelników oraz administratorów.

3.2 Wymagania funkcjonalne

Użytkownik niezalogowany

- Zarejestrowanie się i stworzenie konta, podając imię, nazwisko, unikalny adres e-mail oraz hasło spełniające wymogi bezpieczeństwa.
- Zalogowanie się do systemu za pomocą adresu e-mail oraz hasła.

Użytkownik zalogowany (czytelnik)

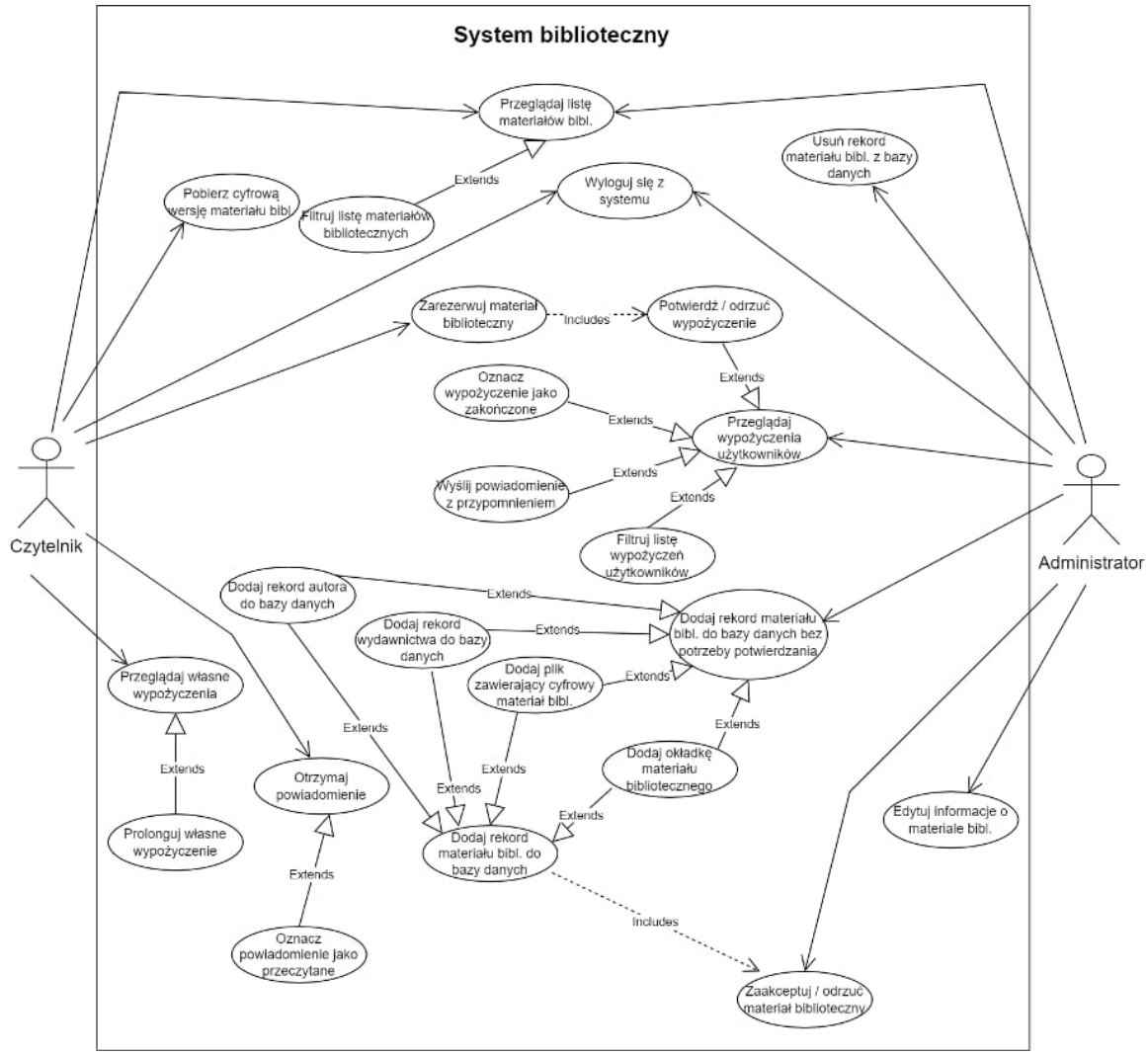
- Przeglądanie materiałów bibliotecznych w formie papierowej udostępnionych do wypożyczenia.
- Przeglądanie materiałów bibliotecznych w formie cyfrowej udostępnionych do pobrania.
- Przeglądanie materiałów papierowych aktualnie niedostępnych (wypożyczonych przez innych użytkowników).
- Filtrowanie katalogu według autora, tytułu, wydawcy lub kategorii.

- Zarezerwowanie materiałów dostępnych do wypożyczenia.
- Prolongowanie aktualnego wypożyczenia.
- Pobranie materiału w formie cyfrowej.
- Dodanie nowych rekordów materiałów bibliotecznych wymagających akceptacji administratora.
- Dodanie autora i wydawnictwa do bazy danych.
- Otrzymywanie powiadomień o zdarzeniach (zaakceptowanie/odrzucenie materiału, wypożyczenia, przypomnienie o terminie zwrotu).
- Oznaczanie powiadomień jako odczytane.
- Przeglądanie trwających, oczekujących, zakończonych oraz odrzuconych wypożyczeń.
- Wylogowanie się z systemu.

Administrator

- Przeglądanie materiałów oczekujących na zatwierdzenie.
- Akceptowanie lub odrzucanie materiałów bibliotecznych.
- Przeglądanie trwających lub zakończonych wypożyczeń wszystkich użytkowników.
- Filtrowanie wypożyczeń według imienia i nazwiska użytkownika.
- Wysyłanie powiadomień przypominających o terminie zwrotu.
- Oznaczanie wypożyczeń jako zakończone.
- Edytowanie informacji o materiałach bibliotecznych.
- Usuwanie materiałów z bazy danych.

3.3 Diagram przypadków użycia



Rysunek 3.1: Diagram przypadków użycia dla projektu *Czytelnia*.

3.4 Podsumowanie

System *Czytelnia* stanowi kompleksowe rozwiązanie dla nowoczesnych bibliotek cyfrowych i tradycyjnych. Dzięki rozbudowanym funkcjonalnościom filtrowania, zarządzania zbiorami oraz automatycznym powiadomieniom o terminach zwrotów, wspiera zarówno użytkowników końcowych, jak i administratorów.

Rozdział 4

Własny przykład UML – System wypożyczania samochodów

4.1 Opis systemu

System wypożyczania samochodów jest prostą aplikacją umożliwiającą klientom rezerwację oraz wynajem pojazdów na określony czas. System obsługuje dwóch głównych aktorów: **Klienta** oraz **Administratora**. Klient może przeglądać dostępne pojazdy, dokonywać rezerwacji oraz zwrotu samochodu. Administrator zarządza flotą pojazdów oraz kontroluje proces rezerwacji.

4.2 Wymagania funkcjonalne

Klient

- Przeglądanie listy dostępnych samochodów.
- Filtrowanie pojazdów po typie, cenie lub dostępności.
- Dokonywanie rezerwacji pojazdu na wybrany okres.
- Anulowanie rezerwacji przed jej rozpoczęciem.
- Zwrot samochodu po zakończeniu okresu wypożyczenia.

Administrator

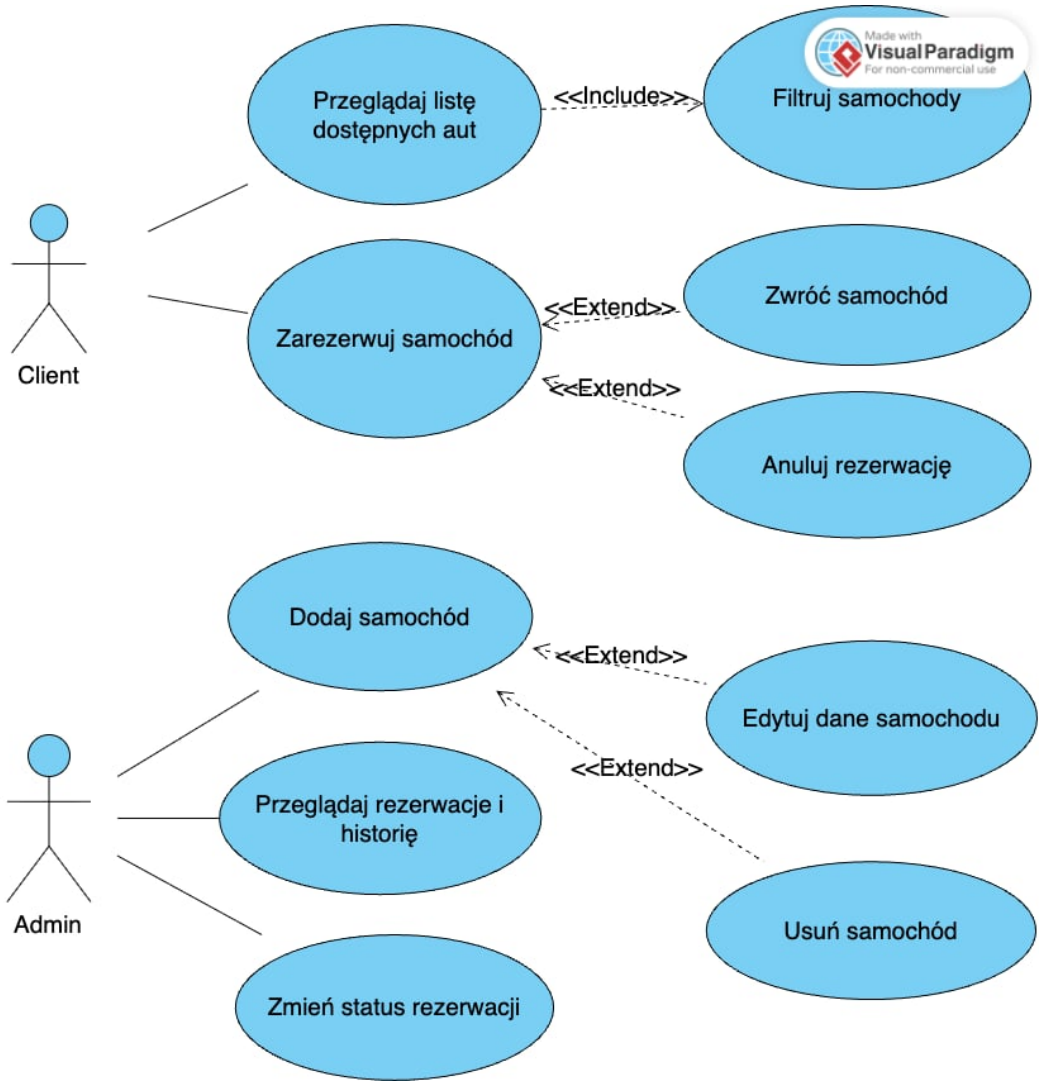
- Dodawanie nowych samochodów do floty.
- Edytowanie informacji o samochodach.
- Usuwanie samochodów z systemu.

- Przeglądanie wszystkich rezerwacji i historii wypożyczeń.
- Zmiana statusu rezerwacji (zatwierdzenie, anulowanie, zakończenie).

4.3 Wymagania niefunkcjonalne

- System powinien umożliwiać obsługę wielu użytkowników jednocześnie.
- Czas odpowiedzi na żądanie nie może przekraczać 3 sekund.
- System powinien przechowywać dane w relacyjnej bazie danych z kopią zapasową wykonywaną co 24 godziny.
- Interfejs użytkownika powinien być intuicyjny i dostępny w przeglądarce.

4.4 Diagram przypadków użycia



Rysunek 4.1: Diagram przypadków użycia – System wypożyczania samochodów.

4.5 Opis działania (przykładowy scenariusz)

- Klient loguje się do systemu i przegląda listę dostępnych samochodów.
- Wybiera interesujący go pojazd i dokonuje rezerwacji.
- Administrator otrzymuje informację o nowej rezerwacji i może ją zatwierdzić lub odrzucić.
- Po zakończeniu okresu wypożyczenia klient zwraca samochód, a system aktualizuje jego status w bazie danych.

4.6 Komentarz projektowy

System wypożyczania samochodów to przykład prostego rozwiązania typu klient–serwer. Logika biznesowa jest rozdzielona od interfejsu użytkownika, co umożliwia łatwą rozbudowę o dodatkowe funkcje, takie jak płatności online, historia transakcji czy integracja z systemem GPS.

Rozdział 5

Wiele diagramów klas w złożonych projektach - przykładowa struktura

W przypadku systemów o dużej skali, próba przedstawienia całej struktury klas na jednym, monolicznym diagramie jest niepraktyczna i prowadzi do jego całkowitej nieczytelności. Diagram taki staje się "plątaniną"(tzw. "big ball of mud"), która zaciemnia architekturę, zamiast ją wyjaśniać. Kluczową strategią jest dekompozycja modelu na wiele mniejszych, wyspecjalizowanych diagramów klas, z których każdy przedstawia system z innej perspektywy lub koncentruje się na konkretnym jego fragmencie.

5.1 Problem złożoności w monolicznym diagramie klas

Monolityczny diagram klas w dużym projekcie cierpi na kilka podstawowych problemów:

- **Przeciążenie informacyjne:** Zbyt wiele klas i relacji na jednym diagramie uniemożliwia ich jednoczesne zrozumienie.
- **Nieczytelność graficzna:** Linie relacji krzyżują się w chaotyczny sposób, a same klasy muszą być mocno pomniejszone, aby zmieścić się na diagramie.
- **Brak separacji odpowiedzialności (Separation of Concerns):** Diagram miesza ze sobą różne aspekty systemu (np. logikę biznesową, dostęp do danych, interfejs użytkownika), utrudniając zrozumienie poszczególnych warstw.
- **Trudności w utrzymaniu:** Każda, nawet mała zmiana w kodzie, może wymagać skomplikowanej aktualizacji centralnego diagramu, co często prowadzi do jego deaktualizacji.

5.2 Strategie podziału modelu klas

Aby zarządzać złożonością, model dzieli się na wiele diagramów, stosując różne kryteria podziału. Ważne jest, aby pamiętać, że jest to **wiele widoków (views) na ten sam, jeden model** – klasa "Użytkownik"jest tą samą klasą, nawet jeśli pojawi się na trzech różnych diagramach.

5.2.1 Podział ze względu na pakiety (widok logiczny)

Jest to najbardziej naturalny i powszechny sposób podziału. Model jest organizowany w pakiety (np. odpowiadające przestrzeniom nazw lub modułom), a następnie tworzone są diagramy klas:

- **Diagram pakietów:** Pokazuje widok "z lotu ptaka- same pakiety i zależności między nimi (np. pakiet `LogikaBiznesowa` zależy od `DostepDoDanych`).
- **Diagram klas dla każdego pakietu:** Dla każdego istotnego pakietu tworzony jest osobny diagram, pokazujący *tylko* klasy zawarte w tym pakiecie oraz (opcjonalnie) klasy spoza pakietu, z którymi się one bezpośrednio komunikują.

Przykład: Osobny diagram dla pakietu "Zarządzanie Zamówieniami", osobny dla "Płatności"i osobny dla "Magazyn".

5.2.2 Podział ze względu na realizację przypadku użycia

Ta strategia koncentruje się na aspekcie behawioralnym. Dla kluczowych lub złożonych przypadków użycia tworzy się diagramy klas, które pokazują *tylko te klasy*, które biorą udział w realizacji danego przypadku.

- Jest to często widok uproszczony, pomijający wiele atrybutów i metod klas, które nie są istotne dla danego scenariusza.
- Taki diagram jest doskonałym uzupełnieniem diagramu sekwencji dla tego samego przypadku użycia.

Przykład: Diagram dla przypadku "Złóż zamówienie"pokazywałby klasy `KontrolerZamowien`, `Zamowienie`, `Koszyk`, `PozycjaZamowienia` oraz interfejs `SerwisPlatnosci`.

5.2.3 Podział ze względu na wzorce projektowe lub architektoniczne

Model można również podzielić, aby pokazać implementację konkretnych wzorców lub warstw architektury.

- **Wzorzec MVC (Model-View-Controller):** Można stworzyć trzy diagramy – jeden dla klas Modelu (logika biznesowa, byty), drugi dla Widoku (GUI) i trzeci dla Kontrolerów (obsługa żądań).
- **Architektura warstwowa:** Osobne diagramy dla warstwy prezentacji, warstwy aplikacji (serwisów), warstwy domeny i warstwy infrastruktury (np. repozytoriów).

5.3 Zarządzanie spójnością między diagramami

Używanie wielu diagramów rodzi ryzyko niespójności. Aby temu zapobiec, kluczowe jest używanie profesjonalnych narzędzi CASE (np. Visual Paradigm, Enterprise Architect, StarUML):

- **Centralne repozytorium modelu (Central Repository):** Narzędzie przechowuje jeden, centralny model (zbiór wszystkich klas, interfejsów itp.). Diagramy są tylko różnymi "widokami"na ten model.
- **Spójność zmian:** Jeśli zmienimy nazwę operacji w klasie `Użytkownik` na jednym diagramie, narzędzie automatycznie zaktualizuje tę nazwę na wszystkich innych diagramach, na których ta klasa występuje.
- **Nawigacja:** Dobre narzędzia pozwalają na łatwą nawigację (np. kliknięcie na klasę `Zamowienie` na diagramie przypadku użycia i przejście do jej pełnej definicji na diagramie pakietu "Zamówienia").

Podsumowanie

W ramach niniejszego sprawozdania przeanalizowano trzy różne systemy informatyczne, modelując ich działanie za pomocą diagramów przypadków użycia UML. Każdy z przedstawionych przykładów reprezentuje odmienny obszar zastosowania technologii informatycznych, jednak wszystkie łączy wspólny cel – zrozumienie relacji pomiędzy użytkownikami systemu a jego funkcjonalnościami.

Pierwszy przykład – **system aukcyjny** – ukazuje typową architekturę aplikacji e-commerce, w której kluczową rolę odgrywa interakcja pomiędzy sprzedawcami, kupującymi i administratorem systemu. Drugi projekt – **system biblioteczny „Czytelnia”** – przedstawia bardziej złożony model, w którym istotna jest kontrola dostępu do zasobów, obsługa powiadomień oraz proces akceptacji operacji przez administratora. Trzeci przykład – **system wypożyczania samochodów** – zaprezentowano jako prosty, lecz przejrzysty model klient–serwer, pozwalający na łatwe rozszerzanie o kolejne moduły biznesowe.

Przeprowadzone modelowanie pozwala zauważyć, że diagramy przypadków użycia stanowią efektywne narzędzie do wizualizacji wymagań funkcjonalnych systemów. Ułatwiają one komunikację pomiędzy analitykami, projektantami i interesariuszami oraz stanowią pierwszy krok w kierunku tworzenia bardziej szczegółowych modeli projektowych – takich jak diagramy klas, sekwencji czy aktywności.

Opracowanie to pokazuje, że umiejętność analizy wymagań i ich odwzorowania w notacji UML jest kluczowym elementem w procesie projektowania systemów informatycznych.