

**POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT ELEKTROTECHNIKI TEORETYCZNEJ I
MIERNICTWA ELEKTRYCZNEGO
ZAKŁAD ELEKTROTECHNIKI TEORETYCZNEJ**

Język UML – opis notacji

**Paweł Gryczon
Piotr Stańczuk**

Fragment pracy dyplomowej magisterskiej pt. „Obiektowy system
konstrukcji scenariuszy przypadków użycia” wykonanej pod opieką dr inż.
Michała Śmiałka

WARSZAWA, grudzień 2002 roku

Spis treści

| | | |
|----------|---|-----------|
| 1 | Wstęp | 3 |
| 2 | UML – ogólne spojrzenie..... | 4 |
| 3 | Modelowanie architektury systemu | 6 |
| 4 | Elementy w UML..... | 8 |
| 4.1 | Elementy strukturalne | 8 |
| 4.1.1 | Klasa..... | 8 |
| 4.1.2 | Interfejs..... | 9 |
| 4.1.3 | Kooperacja..... | 10 |
| 4.1.4 | Przypadek użycia | 11 |
| 4.1.5 | Klasa aktywna..... | 13 |
| 4.1.6 | Komponent | 14 |
| 4.1.7 | Węzeł..... | 15 |
| 4.2 | Elementy czynnościowe..... | 15 |
| 4.3 | Elementy grupujące | 16 |
| 4.4 | Elementy komentujące | 17 |
| 5 | Związki w UML | 18 |
| 5.1 | Zależność..... | 18 |
| 5.2 | Uogólnienie | 18 |
| 5.3 | Powiązanie..... | 19 |
| 5.4 | Realizacja | 21 |
| 6 | Diagramy w UML | 22 |
| 6.1 | Diagram klas..... | 22 |
| 6.2 | Diagram obiektów..... | 23 |
| 6.3 | Diagram przypadków użycia | 24 |
| 6.4 | Diagram interakcji | 25 |
| 6.5 | Diagram stanów | 28 |
| 6.6 | Diagram czynności..... | 29 |
| 6.7 | Diagram komponentów | 30 |
| 6.8 | Diagram wdrożenia | 31 |
| 7 | Podstawowe mechanizmy językowe w UML | 33 |
| 7.1 | Specyfikacje..... | 33 |
| 7.2 | Dodatki | 33 |
| 7.3 | Zasadnicze rozgraniczenia | 34 |
| 7.4 | Mechanizmy rozszerzania | 35 |

1 Wstęp

Unified Modeling Language (UML) to graficzny język do obrazowania, specyfikowania, tworzenia i dokumentowania elementów systemów informatycznych. Umożliwia standaryzację sposobu opracowywania przekrojów systemu, obejmujących obiekty pojęciowe, takie jak procesy przedsiębiorstwa i funkcje systemowe, a także obiekty konkretne, takie jak klasy zaprogramowane w ustalonym języku, schematy baz danych i komponenty programowe nadające się do ponownego użycia.

Języki modelowania obiektowego pojawiły się między połową lat siedemdziesiątych a końcem lat osiemdziesiątych. Z chwilą powstania nowego rodzaju języków programowania obiektowego zaczęto szukać innych rozwiązań dotyczących analizy i projektowania. Opracowanych zostało wiele metod obiektowych, z których jednymi z najważniejszych były: metoda Boocha, metoda OOSE Jacobsona (Object-Oriented Software Engineering) i metoda OMT Rumbaugh (Object Modeling Technique). Każda z nich stanowiła zamkniętą całość. Metoda Boocha sprawdzała się podczas projektowania i implementacji, OOSE stanowiła znakomite wsparcie przy spełnianiu wymagań i wysokopoziomowym projektowaniu, zaś OMT-2 była bardzo przydatna do analizy oraz rozwijania systemów przetwarzających duże ilości danych. W połowie lat dziewięćdziesiątych Grady Booch, Ivan Jacobson i James Rumbaugh postanowili opracować zunifikowany język modelowania.

Oficjalny początek prac nad UML datuje się na październik 1994 roku. W pierwszej kolejności ujednolicono metody Boocha i OMT. Roboczą wersję 0.8 Unified Method (tak została wtedy nazwana) opublikowano w październiku 1995 roku. W tym okresie rozszerzono prace nad UML o uwzględnienie w nim metody OOSE. W czerwcu 1996 roku powstała dokumentacja wersji 0.9. Przez cały rok zbierano uwagi środowiska inżynierów oprogramowania na temat nowego języka. Wynikiem współpracy kilku firm, m.in. Rational, IBM, Hewlett-Packard, Texas Instruments, Unisys, Microsoft, był UML 1.0 – precyzyjnie zdefiniowany język modelowania. W styczniu 1997 roku UML 1.0 został przekazany Object Management Group (OMG) w odpowiedzi na zapotrzebowanie na propozycję standardu języka modelowania obiektowego. Do współpracy włączyły się kolejne firmy, w wyniku czego powstała poprawiona wersja UML (numer 1.1), przyjęta ostatecznie przez OMG 14 listopada 1997.

OMG Revision Task Force (RTF) przejął zadanie pielęgnacji standardu UML. Dokonał redakcyjnych poprawek i w czerwcu 1998 roku przedstawił wersję UML 1.2, a jesienią 1999 roku opublikował UML 1.3.

2 UML – ogólne spojrzenie

Unified Modeling Language (UML) jest językiem znormalizowanym, służącym do zapisywania projektu systemu. Może być stosowany do obrazowania, specyfikowania, tworzenia i dokumentowania elementów powstałych podczas procesu budowy systemu informatycznego. UML wspomaga specyfikowanie wszystkich ważnych decyzji analitycznych, projektowych i implementacyjnych, które muszą być podejmowane w trakcie wytwarzania i wdrażania systemu informatycznego.

UML nie jest językiem programowania graficznego, jednak modele w nim zapisane mogą być wprost powiązane z wieloma językami programowania. Model utworzony w języku UML można przekształcić w taki język, jak Java, C++ czy Visual Basic, albo w tabele relacyjnej bazy danych. To przekształcenie umożliwia inżynierię do przodu, to znaczy generowanie kodu w języku programowania na podstawie modelu UML. Możliwe jest także odwrotne przekształcenie, czyli rekonstrukcja modelu na podstawie implementacji (inżynieria wstecz). Przy przejściu od modelu do kodu każda informacja niezakodowana w implementacji jest tracona, dlatego inżynieria wstecz wymaga odpowiednich narzędzi i udziału człowieka. UML jest na tyle wyrazisty i jednoznaczny, że umożliwia nie tylko bezpośrednie przekształcanie modeli, ale także symulację systemów oraz dostrajanie elementów wdrożonych systemów.

W procesie tworzenia oprogramowania oprócz kodu wykonywalnego powstaje wiele elementów. Są to:

- wymagania,
- architektura,
- projekt,
- kod źródłowy,
- plany projektu,
- testy,
- prototypy,
- kolejne wersje.

Wszystkie te elementy odgrywają istotną rolę w kontrolowaniu, ocenianiu i przekazywaniu informacji o systemie podczas procesu tworzenia go i po jego wdrożeniu i są przedstawiane na zakończenie kolejnych etapów prac. UML obejmuje dokumentowanie architektury systemu i wszystkich jego szczegółów. Składa się na to nie tylko język do zapisywania wymagań i testów, ale także język modelowania czynności, które wykonywane są podczas planowania danego przedsięwzięcia i zarządzania wersjami systemu.

Głównym przeznaczeniem UML jest budowa systemów informatycznych. Z powodzeniem stosowano go już w:

- tworzeniu systemów informacyjnych przedsiębiorstw,
- usługach bankowych i finansowych,
- przemyśle obronnym i lotniczym,
- rozproszonych usługach internetowych,
- telekomunikacji,
- transporcie,
- sprzedaży detalicznej,
- elektronice w medycynie,
- nauce.

Język UML jest na tyle bogaty, że oprócz oprogramowania można modelować w nim systemy nie związane z oprogramowaniem (np. przepływ pracy w ministerstwie,

struktura i zachowanie systemu opieki zdrowotnej oraz projektowanie sprzętu komputerowego).

3 Modelowanie architektury systemu

Modelowanie systemu informatycznego wymaga podejścia do niego z kilku punktów widzenia. Każdy rodzaj użytkownika (użytkownicy końcowi, programiści i analitycy, specjaliści od integracji systemu, osoby wykonujące testy, autorzy dokumentacji technicznej i kierownicy projektu) ma inne spojrzenie na system. Każdy patrzy na zadanie z innej perspektywy i na innym etapie jego życia. Architektura systemu umożliwia kontrolowanie iteracyjnego i przyrostowego procesu tworzenia systemu. Pozwala ogarnąć i opanować wszystkie punkty widzenia, dlatego jest jednym z najistotniejszych elementów systemu.

Architektura jest zbiorem decyzji dotyczących:

- organizacji systemu komputerowego,
- wyboru elementów strukturalnych i ich interfejsów, z których system jest zbudowany,
- zachowania tych elementów opisanego w kooperacjach,
- składania elementów strukturalnych i czynnościowych w coraz większe podsystemy,
- charakterystycznych elementów statycznych i dynamicznych oraz ich interfejsów.

Architektura oprogramowania oprócz jego struktury i zachowania, dotyczy także jego funkcjonalności, efektywności i możliwości ponownego użycia. Obejmuje także ograniczenia ekonomiczne i technologiczne oraz elementy estetyki.

Najlepszym sposobem zobrazowania architektury systemu komputerowego jest przedstawienie go w postaci pięciu powiązanych ze sobą perspektyw. Wszystkie perspektywy dotyczą struktury i organizacji systemu, jednak każda z nich przedstawia inny aspekt tego systemu.



Rys. 1 Obrazowanie architektury systemu

W perspektywie przypadków użycia najważniejsze jest przedstawienie zachowania systemu z punktu widzenia użytkowników, analityków i osób wykonujących testy. Perspektywa ta opisuje czynniki wpływające na kształt systemu. W UML aspekty statyczne tej perspektywy wyraża się za pomocą diagramów przypadków użycia, a dynamiczne za pomocą diagramów interakcji, diagramów stanów i diagramów czynności.

W perspektywie projektowej umożliwia zapisywanie wymagań funkcjonalnych stawianych systemowi. Opisuje usługi, które system powinien udostępniać użytkownikom. Największy nacisk położony jest na klasy, interfejsy i kooperacje, które stanowią słownictwo danego zagadnienia. W UML aspekty statyczne tej perspektywy wyraża się za pomocą diagramów klas i diagramów obiektów, a dynamiczne za pomocą diagramów interakcji, diagramów stanów i diagramów czynności.

Perspektywa procesowa dotyczy głównie efektywności systemu, jego skalowalności i przepustowości. W perspektywie tej brane są pod uwagę wątki i procesy, które kształtują metody współbieżności i synchronizacji w systemie. W UML aspekty statyczne tej perspektywy wyraża się za pomocą diagramów klas (ze szczególnym uwzględnieniem klas aktywnych, które reprezentują procesy i wątki) i diagramów obiektów, a dynamiczne za pomocą diagramów interakcji, diagramów stanów i diagramów czynności.

Perspektywie implementacyjna związana jest z zarządzaniem konfiguracją poszczególnych wersji systemu. Każda wersja systemu składa się z niezależnych komponentów i plików, które mogą być zespolone na wiele sposobów, tworząc działający system. W perspektywie tej największe znaczenie mają komponenty i pliki, użyte do scalenia i udostępnienia systemu fizycznego. W UML aspekty statyczne tej perspektywy wyraża się za pomocą diagramów komponentów, a dynamiczne za pomocą diagramów interakcji, diagramów stanów i diagramów czynności.

Perspektywa wdrożeniowa dotyczy sprzętu, na którym system będzie uruchamiany. Obejmuje zagadnienia związane z rozmieszczeniem, dostarczeniem i instalacją części systemu fizycznego. W UML aspekty statyczne tej perspektywy wyraża się za pomocą diagramów wdrożenia, a dynamiczne za pomocą diagramów interakcji, diagramów stanów i diagramów czynności.

Każda z tych pięciu perspektyw stanowi niezależną całość. Każda osoba biorąca udział w tworzeniu systemu może skoncentrować się na tym fragmencie architektury systemu, który ją najbardziej interesuje. Przedstawione perspektywy ściśle są ze sobą powiązane. Węzły w perspektywie wdrożeniowej zawierają komponenty z perspektywy implementacyjnej, które z kolei reprezentują fizyczną realizację klas, interfejsów, kooperacji i klas aktywnych z perspektywy projektowej i procesowej. UML pozwala na przedstawienie każdej z tych perspektyw i ich wzajemnych oddziaływań.

4 Elementy w UML

W UML istnieją cztery rodzaje elementów:

- 1) strukturalne,
- 2) czynnościowe,
- 3) grupujące,
- 4) komentujące.

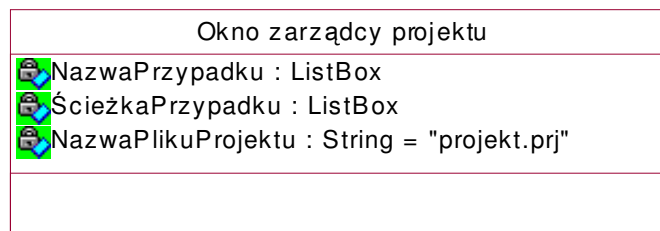
Są to podstawowe obiektowe bloki konstrukcyjne UML. Stosuje się je do budowy modeli.

4.1 Elementy strukturalne

Elementy strukturalne w modelach UML wyrażone są rzeczownikami. Reprezentują składniki pojęciowe albo fizyczne, są statycznymi częściami modelu. Niżej przedstawiamy podstawowe rodzaje elementów strukturalnych.

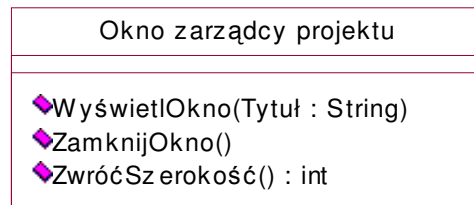
4.1.1 Klasa

Klasa jest opisem zbioru obiektów o takich samych atrybutach, związkach i znaczeniu. Symbolem graficznym klasy jest prostokąt. Każda klasa ma przypisaną nazwę, wyróżniającą ją spośród innych klas. Nazwy mogą być proste lub ścieżkowe, czyli poprzedzone nazwą pakietu. Atrybut stanowi nazwaną właściwość klasy. Określa on zbiór wartości, jakie można przypisać do poszczególnych obiektów tej klasy. Liczba atrybutów klasy nie jest określona, klasa może mieć dowolną liczbę atrybutów lub może nie mieć ich wcale. Atrybut reprezentuje właściwość pewnego modelowanego bytu, która jest określona dla wszystkich jego wystąpień, np. każdy klient sklepu ma nazwisko, adres i datę urodzenia. Atrybut jest abstrakcją pewnego rodzaju danych lub stanu, jakie obiekt klasy może obejmować. W graficznym symbolu klasy atrybuty przedstawiane są w pierwszej sekcji poniżej nazwy klasy. Bardziej szczegółowym określeniem atrybutu jest podanie jego klasy i domyślnej wartości początkowej (Rys. 2).

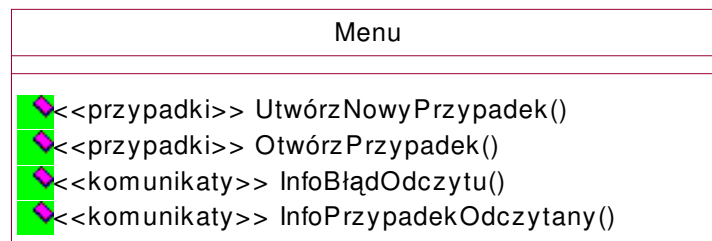


Rys. 2 Klasa i atrybuty

Operacja jest implementacją usługi, którą może wykonać każdy obiekt tej klasy. Klasa może mieć dowolną liczbę operacji lub nie mieć ich wcale. W graficznym symbolu klasy operacje przedstawiane są w sekcji pod atrybutami. Bardziej szczegółowym opisem operacji jest podanie jej sygnatury, która zawiera domyślne wartości początkowe parametrów i w przypadku funkcji typ zwracanej wartości (Rys. 3).

**Rys. 3 Operacje i ich sygnatury**

Symbol klasy nie musi zawierać wszystkich atrybutów i operacji związanych z tą klasą. Najczęściej jest ich na tyle dużo, że nie mieszczą się wszystkie na rysunku. Ponadto nie wszystkie są niezbędne do zrozumienia modelu. Dlatego na diagramie można umieścić niepełny symbol klasy, który pokazuje tylko część atrybutów lub operacji. Niekiedy symbol klasy w ogóle nie zawiera atrybutów lub operacji. Istnieje możliwość zaznaczenia na diagramie, że klasa ma więcej atrybutów lub operacji, niż zostało to przedstawione w symbolu. Wystarczy na końcu każdej listy elementów umieścić wielokropek. Listy atrybutów i operacji można podzielić na grupy wykorzystując stereotypy (Rys. 4).

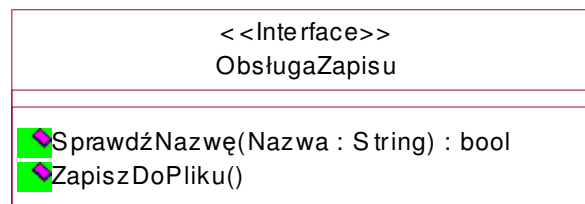
**Rys. 4 Stereotypy**

4.1.2 Interfejs

Interfejs jest zestawem operacji, które wyznaczają usługi oferowane przez klasę lub komponent, ale takich, które dotyczą zewnętrznie obserwowalnych zachowań elementu. Może reprezentować pełne zachowanie klasy lub komponentu lub jedynie część zachowania. Interfejs zawsze określa zbiór deklaracji operacji, czyli ich sygnatur – nie dotyczy implementacji operacji. Podstawowym graficznym symbolem interfejsu jest okrąg z nazwą zapisaną niżej. Na diagramie interfejs najczęściej powiązany jest z realizującą go klasą lub komponentem. Przykład interfejsu widoczny jest na rysunku (Rys. 5).

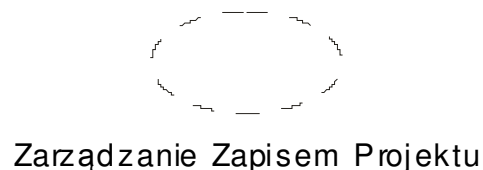
**Rys. 5 Interfejs**

Graficzny symbol interfejsu w postaci okręgu z nazwą jest tzw. postacią normalną. W takim przypadku nie są uwidocznione operacje interfejsu. Niekiedy jednak jest niezbędne do zrozumienia modelu. Wówczas interfejs przedstawiany jest jako stereotypowana klasa i operacje, tak jak w przypadku symbolu zwykłej klasy, podawane są pod sekcją atrybutów (Rys. 6).

**Rys. 6 Operacje**

4.1.3 Kooperacja

Kooperacja związana jest z architekturą systemu, służy do specyfikowania realizacji przypadków użycia oraz do modelowania mechanizmów systemowych, które są istotne z punktu widzenia architektury systemu. Z każdą kooperacją wiążą się dwa aspekty: struktura i zachowanie. Część dotycząca struktury najczęściej przedstawiana jest w postaci diagramów klas, natomiast część dotycząca zachowania obrazowana jest za pomocą diagramów interakcji. Pojedyncza klasa może brać udział w wielu kooperacjach. Kooperacje mogą także modelować realizację operacji. Jest to szczególnie przydatne gdy implementacja operacji wymaga współpracy wielu obiektów. Na diagramie kooperacje przedstawione są w postaci elipsy z przerywaną linią brzegową i nazwą (Rys. 7)

**Rys. 7 Kooperacja**

4.1.4 Przypadek użycia

Przypadek użycia jest opisem zbioru sekwencji czynności, które są wykonywane przez gotowy system, aby dostarczyć każdemu aktorowi żadanego wyniku. Służy do określania w modelu struktury zachowania systemu. Przypadek użycia realizowany jest przez kooperację. Symbolem graficznym przypadku użycia jest elipsa narysowana ciągłą linią i nazwa (Rys. 8)



Utworzenie słownika

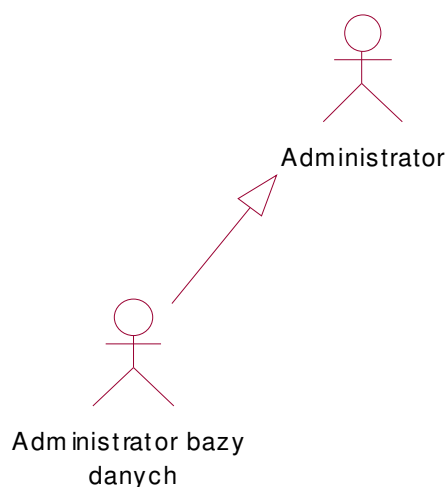
Rys. 8 Przypadek użycia

Zadaniem przypadków użycia jest modelowanie oczekiwanego zachowania systemu. Modelując zachowanie za pomocą przypadków użycia nie ma konieczności zagłębiania się w zagadnienia związane z implementacją tego zachowania. Przypadki użycia pozwalają osiągnąć porozumienie między programistami a użytkownikami i specjalistami z danej dziedziny. Umożliwiają weryfikację struktury i architektury systemu w trakcie jego tworzenia. Poprawne przypadki użycia skupiają się tylko na zasadniczych zachowaniach systemu. Każdy ciąg czynności, opisywany przez przypadek użycia, reprezentuje oddziaływanie elementów stanowiących otoczenie systemu, czyli aktorów, z samym systemem. Oddziaływania są w zasadzie funkcjami systemowymi, używanymi do określania żądanych zachowań budowanego systemu jeszcze na etapie analizy zachowania systemu i określania wymagań funkcjonalnych. Przypadek użycia reprezentuje wymaganie funkcjonalne dla systemu jako całości.

Bardziej rozbudowane systemy zawierają przypadki użycia, które stanowią uszczegółowienie innych przypadków użycia lub są ich częściami. Są także takie, które stanowią rozszerzenie głównych przypadków użycia. Z punktu widzenia konkretnego aktora przypadek użycia opisuje działanie dające rezultat, którego ten aktor oczekuje, na przykład obliczenie wyniku, utworzenie nowego obiektu lub zmianę stanu danego obiektu.

Przypadki użycia pozwalają analizować cały system, jednak mogą mieć także zastosowanie podczas analizy części systemu (np. podsystemów) lub pojedynczych klas i interfejsów. Przypadki użycia opisują oczekiwane zachowanie tych elementów i stanowią podstawę do opracowania dla nich przypadków testowych w miarę ich rozbudowywania.

Elementami nie będącymi częścią systemu, ale ściśle z nim związanymi, są aktorzy. Aktorzy Reprezentują role odgrywane przez użytkowników przypadku użycia w czasie interakcji z tym przypadkiem. Stanowią kontekst przypadku użycia. Dobre opracowanie modelu aktorów pozwala na zrozumienie kto lub co będzie wchodziło w interakcję z systemem. Aktor może być zwykłym klasycznym użytkownikiem (człowiekiem), może to być też program a nawet urządzenie sprzętowe. Na rysunku (Rys. 9) przedstawiono graficzny symbol aktora. Stosując uogólnienia można definiować ogólne rodzaje aktorów (np. Administrator) i uszczegółowienia (np. Administrator bazy danych).



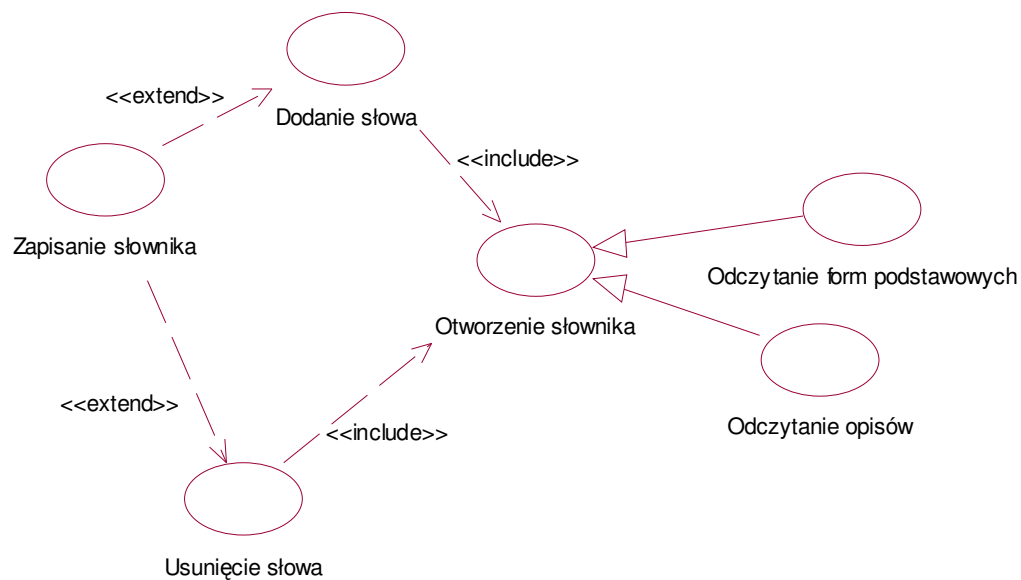
Rys. 9 Aktorzy

Niektórzy aktorzy mogą stanowić formy specjalizacji innych aktorów, pewna funkcja jednego aktora może wchodzić w zakres obowiązków innego aktora. Tworzenie hierarchii aktorów pozwala na uproszczenie diagramów przypadków użycia.

Przypadek użycia może być wyspecyfikowany przez opisanie ciągu zdarzeń w formie tekstu zrozumiałego nawet dla osoby nie znającej dogłębnie tematu. Każdy przypadek ma główny ciąg zdarzeń i ciągi poboczne (alternatywne). Każdy taki ciąg nosi nazwę scenariusza. Scenariusze przypadku użycia jest tekstowym opisem kroków składających się na dany przypadek. Liczba scenariuszy jest zawsze znacznie większa niż liczba przypadków użycia. Scenariusz zazwyczaj prezentowany jest w postaci numerowanej listy, z podziałem na aktora i system, oznaczający w tym wypadku oprogramowanie implementujące analizowany przypadek. Scenariusze są pisane z myślą o klientach i powinny być dostosowane do ich poziomu zrozumienia tematu. Głównym celem jest zachowanie przejrzystości. Należy unikać zagłębiania się w opisy interfejsów użytkownika lub konkretnych rozwiązań sprzętowych. Scenariusz powinien być abstrakcyjny. Decyzje dotyczące interfejsów zostają przesunięte na etap projektowania, a scenariusz prezentuje jedynie ogólne zachowanie systemu.

Przypadki użycia podobnie jak klasy można grupować w pakiety. Można je także uporządkować przez zdefiniowanie uogólnień między nimi oraz związków zawierania i rozszerzania. Polega to na wydzieleniu wspólnych fragmentów zachowania przez usunięcie ich z obejmujących je przypadków użycia lub wspólnych wariantów przez wklejanie ich do rozszerzających je przypadków. Uogólnienie między przypadkami użycia oznacza, że przypadek użycia - potomek dziedziczy całe zachowanie i znaczenie po przypadku użycia przodka. Związek zawierania między przypadkami polega na tym, że bazowy przypadek użycia jawnie włącza zachowanie innego przypadku użycia w miejscu przez siebie określonym. Włączany przypadek nigdy nie występuje samodzielnie – jego egzemplarze mogą być tylko częścią większego zawierającego go przypadku użycia. Związek zawierania używa się w celu uniknięcia wielokrotnego opisywania tego samego ciągu zdarzeń. Wspólne zachowanie jest definiowane w odrębnym przypadku użycia, który jest następnie włączany przez bazowe przypadki

użycia. Związek zawierania obrazuje się w postaci zależności stereotypowanej jako `<<include>>`. Związek rozszerzania między przypadkami użycia polega na tym, że bazowy przypadek użycia w sposób domniemany włącza zachowanie innego przypadku w miejscu określonym pośrednio przez rozszerzający przypadek użycia. Bazowy przypadek użycia może wystąpić samodzielnie, ale pod pewnymi warunkami jego zachowanie może być rozszerzone przez zachowanie innego przypadku użycia. Związek rozszerzania służy do modelowania fragmentów przypadku użycia postrzeganych przez użytkownika jako opcjonalne zachowanie systemu. Obrazuje się go w postaci zależności stereotypowanej jako `<<extend>>`. Uogólnianie, zawieranie i rozszerzanie zostało przedstawione na rysunku (Rys. 10).

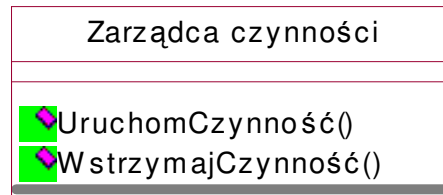


Rys. 10 Uogólnienia i związki między przypadkami

Kolejne trzy rodzaje elementów strukturalnych, to znaczy klasy aktywne, komponenty i węzły, są podobne do klas. Określają one zbiory obiektów o zbliżonych atrybutach, operacjach, związkach i znaczeniu. Są jednak na tyle inne i na tyle potrzebne do modelowania pewnych aspektów systemu obiektowego, że należy się im specjalne potraktowanie.

4.1.5 Klasa aktywna

Klasa aktywna jest podobna do zwykłej klasy, jednak zawiera obiekty, w skład których wchodzi co najmniej jeden proces lub wątek. Takie obiekty mogą samodzielnie rozpocząć przepływ sterowania. Obiekty klasy aktywnej reprezentują elementy działające równolegle z innymi. Na diagramie jest przedstawiana jak zwykła klasa - różnicą jest pogrubione obramowanie prostokąta (Rys. 11)



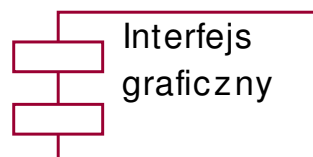
Rys. 11 Klasa aktywna

Obiekty klas aktywnych służą do modelowania niezależnych przepływów sterowania. Obiekt aktywny (egzemplarz klasy aktywnej) reprezentuje proces lub wątek. Z chwilą utworzenia obiektu aktywnego zostaje uruchomiony związany z nim przepływ sterowania. W momencie zniszczenia takiego obiektu przepływ ten zostaje przerwany. Klasy aktywne mają te same właściwości co zwykłe klasy. Mają egzemplarze, atrybuty i operacje. Mogą być elementami zależności, uogólnień i powiązań. Można wobec nich stosować wszystkie mechanizmy rozszerzania UML, to znaczy stereotypy, metki i ograniczenia. Mogą być realizacjami interfejsów i mogą być realizowane przez kooperacje. Ich zachowanie może być zdefiniowane za pomocą maszyny stanowej.

Pozostałe dwa rodzaje elementów strukturalnych, komponenty i węzły, w odróżnieniu od dotychczas omówionych, reprezentują byty fizyczne, a nie pojęciowe i logiczne.

4.1.6 Komponent

Komponent stanowi fizyczną część systemu. Jest elementem wymiennym, wykorzystującym i realizującym pewien zbiór interfejsów. Komponent jest fizycznym opakowaniem klas, interfejsów i kooperacji. Każdy system posiada wiele komponentów będących elementami procesu wytwarzania (np. pliki z kodem źródłowym) oraz komponentów już wdrożonych. Na diagramach symbolem graficznym komponentu jest prostokąt z bolcami z nazwą w środku (Rys. 12).



Rys. 12 Komponent

Komponenty mają wiele cech wspólnych z klasami: mają nazwy, realizują pewien zbiór interfejsów, mogą brać udział w zależnościach, uogólnieniach i powiązaniach, mogą być zagnieżdżone, mieć egzemplarze i uczestniczyć w interakcjach.

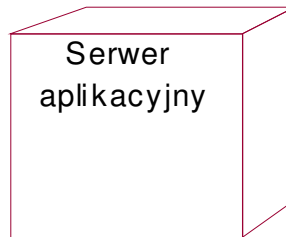
W UML jest zdefiniowanych pięć standardowych stereotypów komponentów:

1. `executable` - określa komponent, który można wykonać na węźle.
2. `library` - określa dynamiczną lub statyczną bibliotekę obiektów.
3. `table` - określa komponent reprezentujący tabelę bazy danych.

4. `file` - określa komponent reprezentujący dokument zawierający kod źródłowy lub dane.
5. `document` – określa komponent reprezentujący dokument.

4.1.7 Węzeł

Węzeł jest fizycznym składnikiem działającego systemu, który reprezentuje pewne zasoby obliczeniowe. Cechuje go posiadanie pewnej ilości pamięci i zdolności przetwarzania. Najczęściej w węzłach znajdują się komponenty, niekiedy komponenty przemieszczają się między węzłami. Symbolem graficznym węzła jest jako sześcian z nazwą w środku (Rys. 13).



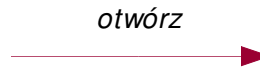
Rys. 13 Węzeł

Węzły mogą brać udział w zależnościach, uogólnieniach i powiązaniach. Mogą być zagnieżdżone i mogą uczestniczyć w interakcjach. Najczęściej występującym związkiem między węzłami jest powiązanie. W przypadku węzłów oznacza ono połączenie fizyczne, takie jak sieć Ethernet, łącze szeregowe lub wspólna szyna. Powiązań można także użyć do modelowania połączeń pośrednich, takich jak komunikacja satelitarna między odległymi procesorami. Węzeł jest podobny do klasy, można więc wykorzystywać wszystkie właściwości powiązań, czyli rolę, liczebność i ograniczenia.

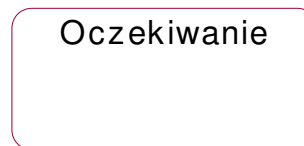
4.2 Elementy czynnościowe

Elementy czynnościowe dotyczą dynamicznej części modelu w UML. Wyrażone są czasownikami opisującymi zachowanie w czasie i w przestrzeni. Wyróżniamy dwa rodzaje takich elementów.

Interakcja jest zachowaniem polegającym na wymianie komunikatów między obiektami. Interakcje wykorzystywane są do modelowania przepływu sterowania w ramach operacji, klasy, komponentu czy przypadku użycia. Za pomocą interakcji można zdefiniować zarówno zachowanie zespołu obiektów, jak i pojedynczą operację. Interakcja składa się z komunikatów, ciągów akcji będących odpowiedzią na ten komunikat i połączeń między obiektami. Komunikat jest przedstawiany na diagramie jako strzałka z nazwą operacji (Rys. 14).

**Rys. 14 Komunikat**

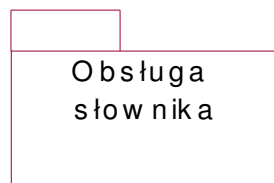
Drugim elementem czynnościowym jest maszyna stanowa. Określa ona ciąg stanów, jakie obiekt lub interakcja przyjmuje w odpowiedzi na zdarzenia zachodzące w czasie ich życia oraz ich odpowiedzi na te zdarzenia. Za jej pomocą może być zdefiniowane zachowanie pojedynczej klasy lub kooperacji. Maszyna stanowa składa się z innych elementów, takich jak stany, przejścia między stanami, zdarzenia, które powodują przejścia i czynności, będące odpowiedziami na zdarzenia. Stan jest przedstawiany na diagramie jako prostokąt z zaokrąglonymi rogami z nazwą w środku (Rys. 15).

**Rys. 15 Stan**

4.3 Elementy grupujące

Elementy grupujące odgrywają w UML rolę organizacyjną. Są to bloki, na które dany model może być rozłożony. Podstawowym rodzajem tego typu elementu jest pakiet.

Pakiet w rozumieniu UML-owym to zgrupowanie elementów modelu. Pozwala na organizowanie klas w zbiory niezależne od struktur zapisanych na diagramach klas. Oznacza to, że można umieścić wszystkie klasy na pojedynczym diagramie, można także rozbić go na szereg prostszych diagramów. Pakiety to w rzeczywistości jedynie przestrzenie nazewnicze, grupujące elementy, które powinny zawierać unikatowe nazwy w obrębie danej grupy. Z pakietów można korzystać przy porządkowaniu elementów składowych podsystemów, bez konieczności tworzenia dodatkowych przypadków użycia. Na diagramie pakiet przedstawiany jest jako prostokąt z fiszką, zazwyczaj tylko z nazwą w środku (Rys. 16).

**Rys. 16 Pakiet**

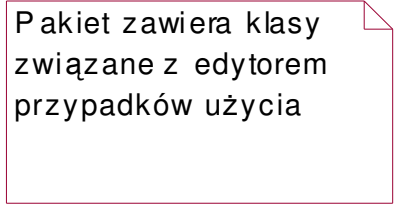
Składnikami pakietu mogą być różne byty, takie jak klasy, interfejsy, komponenty, węzły, operacje, przypadki użycia, diagramy, a nawet inne pakiety. Model w

rozumieniu UML-owym stanowi pakiet zawierający pełną reprezentację systemu w konkretnym aspekcie. Potraktowanie architektury systemu jako zbioru powiązanych i zagnieżdżonych pakietów pomaga w optymalizacji tworzonych struktur. Celem pakietyzacji elementów modelu jest rozdzielenie podstawowych części składowych systemu i uniezależnienie ich od siebie. To z kolei pozwala na enkapsulację dużych fragmentów systemu w pakietach i daje możliwość ich powtórnego wykorzystania. Pakietom towarzyszą zazwyczaj interfejsy lub zestawy interfejsów reprezentujące udostępniane przez nie usługi. W UML zakłada się, że w modelu istnieje nienazwany pakiet nadrzędny. Konsekwencją tego założenia jest konieczność unikatowego nazywania bytów każdego rodzaju, zdefiniowanych u góry modelu. Wszystkie mechanizmy rozszerzania UML dotyczą także pakietów. Najczęściej są to metki definiujące nowe właściwości pakietów.

Pakiety są podstawowymi elementami grupującymi, za pomocą których można usystematyzować model zapisany w UML. Istnieją też inne elementy tego typu, takie jak zręby, modele i podsystemy (rodzaje pakietów).

4.4 Elementy komentujące

Elementy komentujące odgrywają w UML rolę objaśniającą. Są to adnotacje, których można użyć w celu opisanego, uwypuklenia lub zaznaczenia dowolnych składników modelu. Podstawowym rodzajem tego typu elementu jest notatka. Jest to symbol umożliwiający skojarzenie dodatkowych ograniczeń i objaśnień z pojedynczym bytem lub grupą bytów. Na diagramie jest przedstawiana jako prostokąt z zagiętym rogiem, z komentarzem tekstowym lub graficznym w środku (Rys. 17).



Pakiet zawiera klasy
związane z edytorem
przypadków użycia

Rys. 17 Notatka

Notatka to podstawowy element komentujący, który może się pojawić w modelu zapisanym w UML. Zwykle używa się jej w celu wzbogacenia diagramu o ograniczenia i objaśnienia, które najłatwiej wyrazić za pomocą formalnego lub nieformalnego tekstu. Są też inne rodzaje elementów tego typu, takie jak wymagania, które definiują zachowanie oczekiwane przez otoczenie modelu.

5 Związki w UML

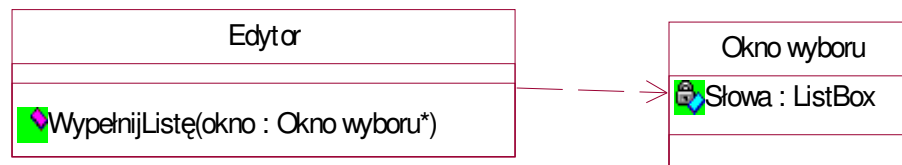
W UML są uwzględnione cztery rodzaje związków:

- 1) zależność,
- 2) uogólnienie,
- 3) powiązanie,
- 4) realizacja.

Związki te są podstawowymi blokami konstrukcyjnymi UML, służącymi do łączenia elementów.

5.1 Zależność

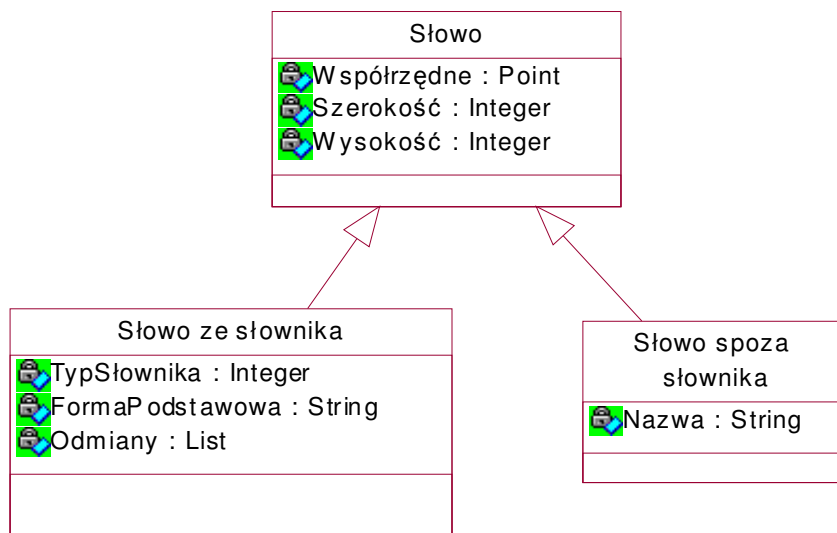
Zależność pozwala na pokazanie, że jedna klasa używa drugiej jako argumentu w sygnaturze operacji (Rys. 18). Jest to najczęściej spotykany sposób użycia zależności. Ponadto UML dopuszcza stosowanie zależności także między pakietami i notatkami, jednak są one rzadziej używane. Zmiany dokonane w specyfikacji jednego elementu mogą mieć wpływ na inny element, który go używa. Na diagramie zależność przedstawiana jest jako linia przerywana z grotem skierowanym na element, od którego coś zależy.



Rys. 18 Zależność

5.2 Uogólnienie

Uogólnienie jest związkiem między elementem ogólnym (zwanym nadklasą lub przodkiem) a pewnym specyficznym jego rodzajem (zwanym podklasą lub potomkiem). Uogólnienie polega na tym, że potomek może wystąpić wszędzie tam, gdzie jest spodziewany przodek, ale nie na odwrót. Potomek dziedziczy wszystkie właściwości przodka, w szczególności atrybuty i operacje, i zawsze może go zastąpić. Najczęściej potomek oprócz cech odziedziczonych po przodku ma także własne cechy. Operacja potomka mająca tę samą sygnaturę co operacja przodka jest ważniejsza (polimorfizm). Uogólnienie jest przedstawiane na diagramie jako linia ciągła zakończona zamkniętym, niewypełnionym grotem wskazującym przodka (Rys. 19).



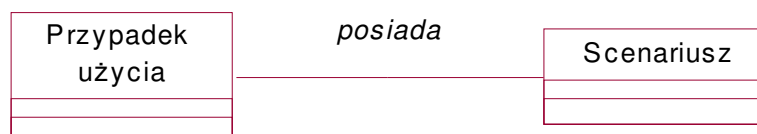
Rys. 19 Uogólnienie

Najczęściej uogólnień używa się względem klas i interfejsów w celu przedstawienia dziedziczenia. UML umożliwia tworzenie uogólnień także między innymi elementami, w szczególności między pakietami.

5.3 Powiązanie

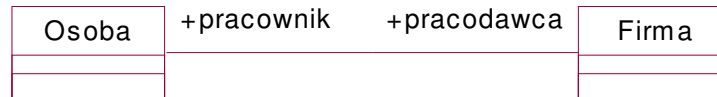
Powiązanie jest związkiem służącym do pokazania, że obiekty jednego elementu są połączone z obiektami innego. Powiązanie między dwiema klasami oznacza, że można przejść z obiektu jednej z tych klas do obiektu drugiej i odwrotnie. Możliwe jest także takie powiązanie, którego oba końce wskazują tę samą klasę. Oznacza to, że każdy obiekt tej klasy może być połączony z innymi obiektami tej klasy. Na diagramie powiązanie jest przedstawiane jako linia ciągła, łącząca klasę z sobą samą lub z inną klasą.

Powiązanie może mieć przypisaną nazwę, która określa istotę danego związku (Rys. 20). Aby uniknąć niejednoznaczności, można podać kierunek odczytu w postaci trójkątnego znacznika umieszczonego za nazwą powiązania.



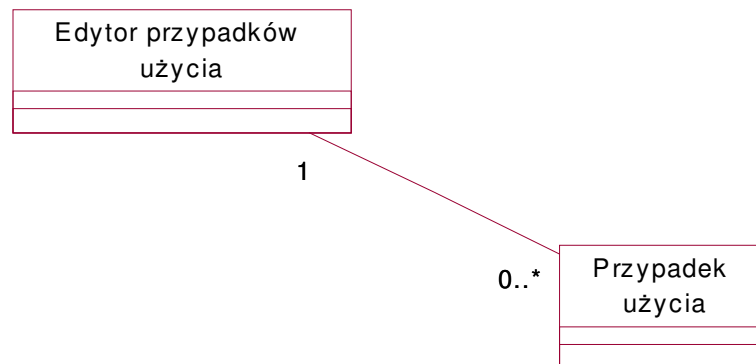
Rys. 20 Nazwa powiązania

Każda klasa biorąca udział w powiązaniu odgrywa w nim określoną rolę. Rola klasy może być jawnie nazwana w powiązaniu. Na rysunku (Rys. 21) klasa *Osoba* odgrywająca rolę *pracownika* jest powiązana z klasą *Firma* w roli *pracodawcy*.



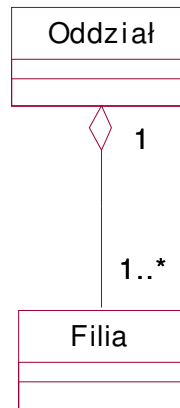
Rys. 21 Role

Często w powiązaniu zachodzi potrzeba podania liczebności, czyli liczby obiektów jaka może być połączona przez jeden egzemplarz powiązania. Ilość obiektów nazywana jest także krotnością. Liczebność zapisywana jest w postaci wyrażenia, którego wartością jest przedział liczbowy lub pojedyncza liczba (Rys. 22). Podając liczebność przy jednym końcu powiązania wskazujemy ile obiektów jednej klasy musi być połączonych z każdym obiektem klasy znajdującej się na końcu przeciwnym. Liczebność można ustalić na dokładnie jeden (1), zero lub jeden (0..1), dowolnie wiele (0..*) albo co najmniej jeden (1..*). Może to być także pewna ustalona liczba (np. 3).



Rys. 22 Liczebność

Najczęściej powiązanie dwóch klas jest związkiem strukturalnym elementów równorzędnych, czyli powiązane klasy znajdują się na tym samym poziomie pojęciowym. Czasami występuje potrzeba zapisania agregacji, czyli związku rodzaju „całość-część”. W takim związku jedna klasa reprezentuje większy element stanowiący całość, a druga reprezentuje elementy mniejsze, czyli części, z których składa się całość. Agregacja jest szczególnym rodzajem powiązania. Na diagramie wyróżniana jest przez dodanie do zwykłego symbolu powiązania pustego rombu po stronie całości (Rys. 23).



Rys. 23 Agregacja

5.4 Realizacja

Realizacja jest związkiem znaczeniowym między klasyfikatorami, z których jeden określa kontrakt, a drugi zapewnia wywiązanie się z niego. Takie związki występują najczęściej między interfejsami a klasami i komponentami oraz między przypadkami użycia a kooperacjami. Na diagramach realizacja przedstawiana jest jako połączenie symboli uogólniania i zależności, czyli jako linia przerywana zakończona zamkniętym niewypełnionym grotem (Rys. 24).



Rys. 24 Realizacja

Omówione cztery rodzaje związków to podstawowe bloki konstrukcyjne umożliwiające łączenie elementów modelu w UML.

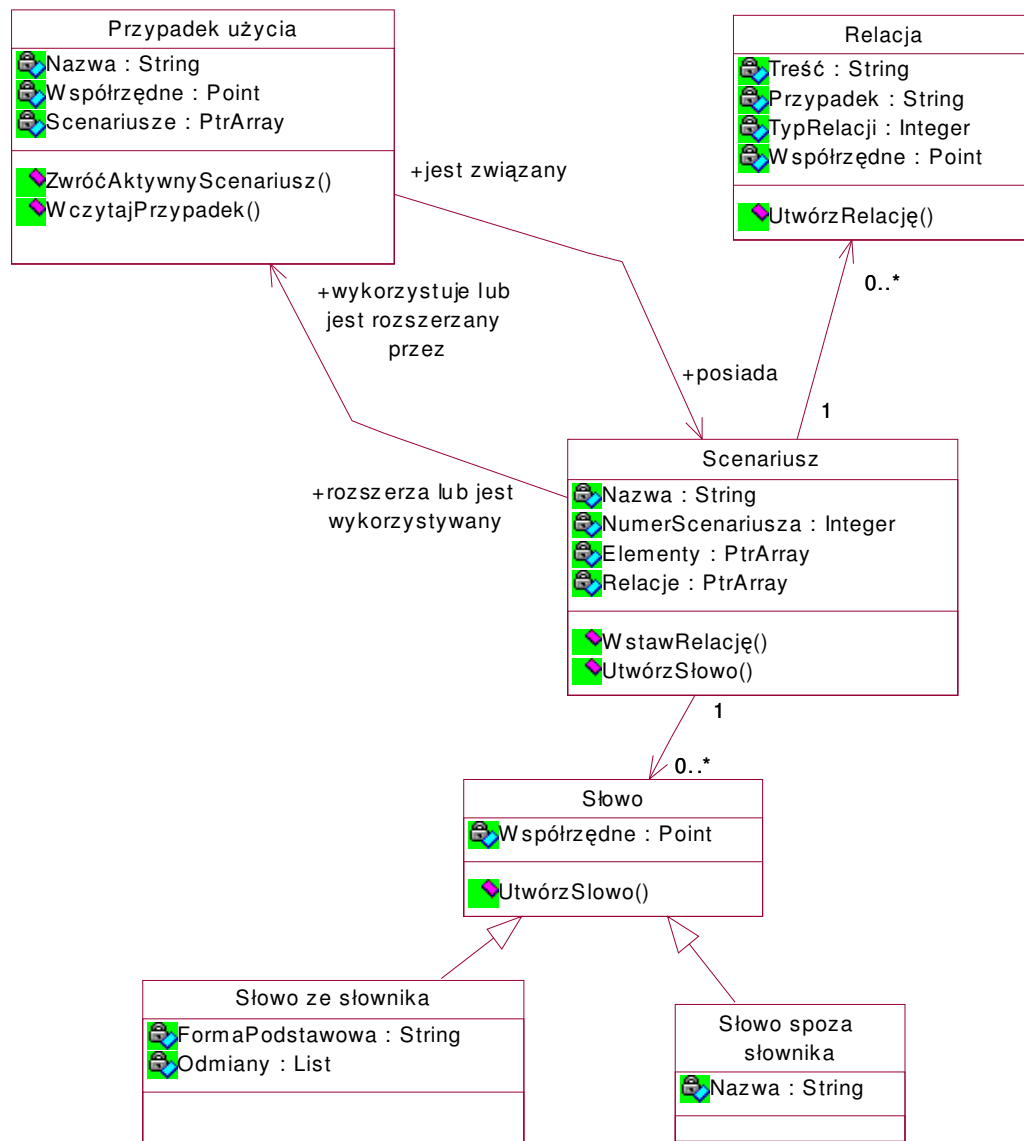
6 Diagramy w UML

Diagram jest schematem przedstawiającym zbiór bytów. Najczęściej ma postać grafu, w którym wierzchołkami są elementy, a krawędziami związki. Diagram to swego rodzaju rzut systemu. Tylko w przypadku najprostszych systemów diagram przedstawia pełny obraz bytów wchodzących w skład systemu. Ten sam byt może się pojawić na wszystkich diagramach, jednak najczęściej występuje tylko na niektórych. W wyjątkowych przypadkach dany byt może nie wystąpić na żadnym diagramie. Teoretycznie diagram może zawierać dowolną kombinację elementów i związków. W praktyce jednak tylko niektóre z kombinacji odpowiadają pięciu najbardziej użytecznym perspektywom architektonicznym systemu oprogramowania. Z tego powodu w UML wyróżnia się następujące rodzaje diagramów:

- 1) diagram klas,
- 2) diagram obiektów,
- 3) diagram przypadków użycia,
- 4) diagram przebiegu (sekwencji),
- 5) diagram kooperacji,
- 6) diagram stanów,
- 7) diagram czynności,
- 8) diagram komponentów,
- 9) diagram wdrożenia.

6.1 Diagram klas

Diagramy klas to najczęściej występujące diagramy w modelach obiektowych. Każdy z nich przedstawia określony fragment struktury systemu klas. Diagramów klas używa się do modelowania statycznych aspektów perspektywy projektowej. Wiąże się z tym w głównej mierze modelowanie słownictwa systemu, kooperacji lub schematów. Diagramy klas stanowią bazę wyjściową do dwóch innych diagramów: diagramu komponentów i diagramu wdrożenia. Diagram klas uwzględniający klasy aktywne dotyczy statycznych aspektów perspektywy procesowej. Diagramy klas nie ograniczają się tylko do samych klas, można za ich pomocą modelować interfejsy, relacje a nawet pojedyncze instancje klas. Diagramy klas pozwalają na sformalizowanie specyfikacji danych i metod. Specyfikacja ta jest związana z oprogramowaniem, ale dotyczy jego zewnętrznego opisu bez wchodzenia w szczegóły implementacyjne. Diagramy klas mogą także pełnić rolę graficznego środka pokazującego szczegóły implementacji klas np. w C++. Przykład diagramu klas przedstawiony jest na rysunku (Rys. 25).



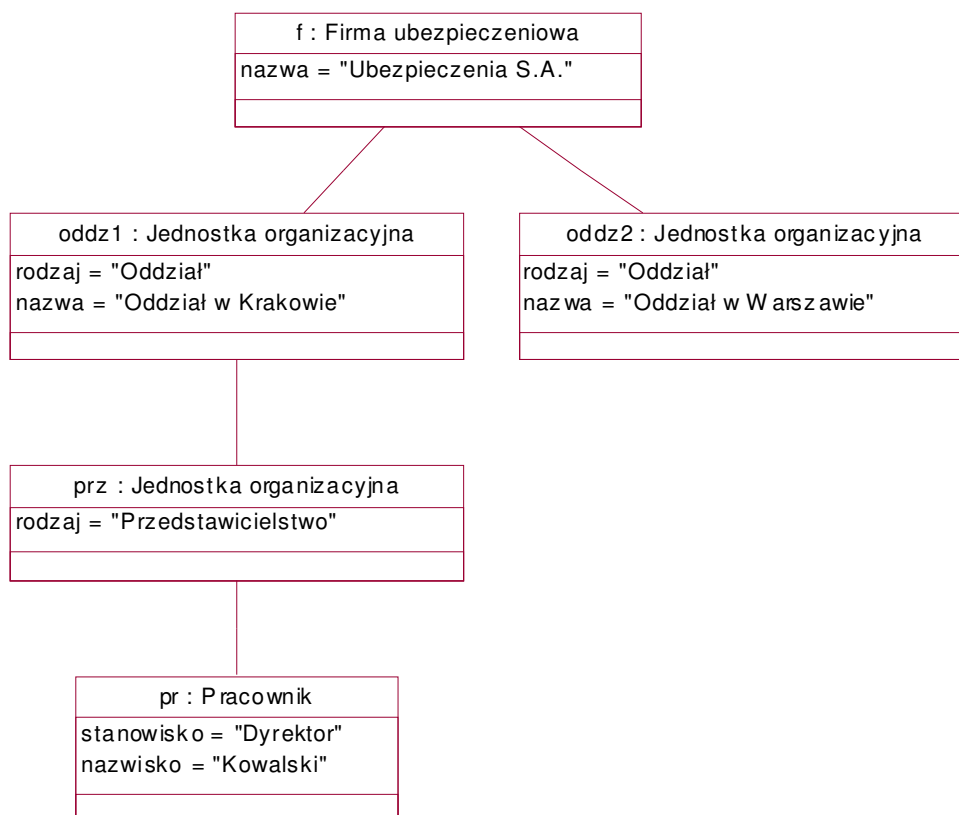
Rys. 25 Diagram klas

Na diagramach klas mogą się znaleźć również pakiety i podsystemy, używane do grupowania bytów modelu w większe porcje.

6.2 Diagram obiektów

Na diagramie obiektów przedstawia się obiekty, czyli konkretne instancje klas i związki między nimi. Diagram ten wyobraża statyczny rzut pewnych egzemplarzy elementów występujących na diagramie klas. Podobnie jak diagram klas, odnosi się do statycznych aspektów perspektywy projektowej lub procesowej. Korzystając z niego bierze się

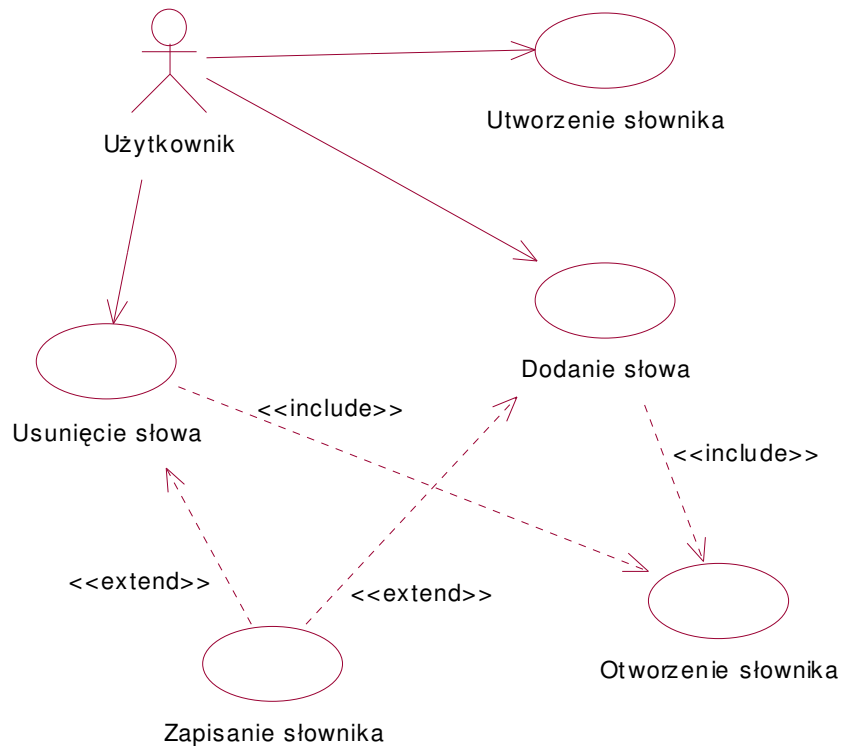
jednak pod uwagę przypadki rzeczywiste lub prototypowe. Przykład diagramu obiektów przedstawiony jest na rysunku (Rys. 26).



Rys. 26 Diagram obiektów

6.3 Diagram przypadków użycia

Diagram przypadków użycia ukazuje związki pomiędzy aktorami i przypadkami. Umieszcza się także na nim wzajemne relacje między poszczególnymi przypadkami użycia. Diagram ten odnosi się do statycznych aspektów perspektywy przypadków użycia. Wykorzystywany jest głównie do wyznaczania i modelowania zachowania systemu w taki sposób, żeby użytkownicy mogli zrozumieć jak z niego korzystać, a programiści mogli go zaimplementować. Dzięki diagramom przypadków użycia systemy, podsystemy i klasy stają się bardziej przystępne i zrozumiałe. Model przypadków użycia przekształca wymagania wstępne w przejrzystą reprezentację systemu, który należy skonstruować. Można go doprecyzowywać i uszczegóławiać poprzez dodawanie nowych aktorów, nowych przypadków użycia i powiązań pomiędzy nimi. Diagram przypadków użycia dostarcza bardzo abstrakcyjnego poglądu na system z pozycji aktorów, którzy go używają. Nie włącza szczegółów, co pozwala wnioskować o systemie na odpowiednio ogólnym, abstrakcyjnym poziomie. Przykładowy diagram przypadków przedstawiony jest na rysunku (Rys. 27).



Rys. 27 Diagram przypadków użycia

Można wyróżnić dwa zasadnicze cele istnienia diagramów przypadków użycia: modelowanie otoczenia systemu i modelowanie wymagań stawianych systemowi. Modelowanie otoczenia polega między innymi na wyznaczeniu granicy wokół całego systemu i na wskazaniu leżących poza nią aktorów, którzy wchodzi w interakcję z systemem. Diagramy przypadków użycia w tym wypadku służą do zdefiniowania aktorów i znaczenia ich ról. Modelowanie wymagań polega na określeniu, co system ma robić z punktu widzenia jego otoczenia, bez zagłębiania się w to, w jaki sposób ma to robić. W tym przypadku diagram służy do zdefiniowania oczekiwanego działania systemu. Na diagramach przypadków użycia zaznaczane są uogólnienia oraz związki zawierania i rozszerzania. Zostało to omówione w punkcie dotyczącym przypadków użycia.

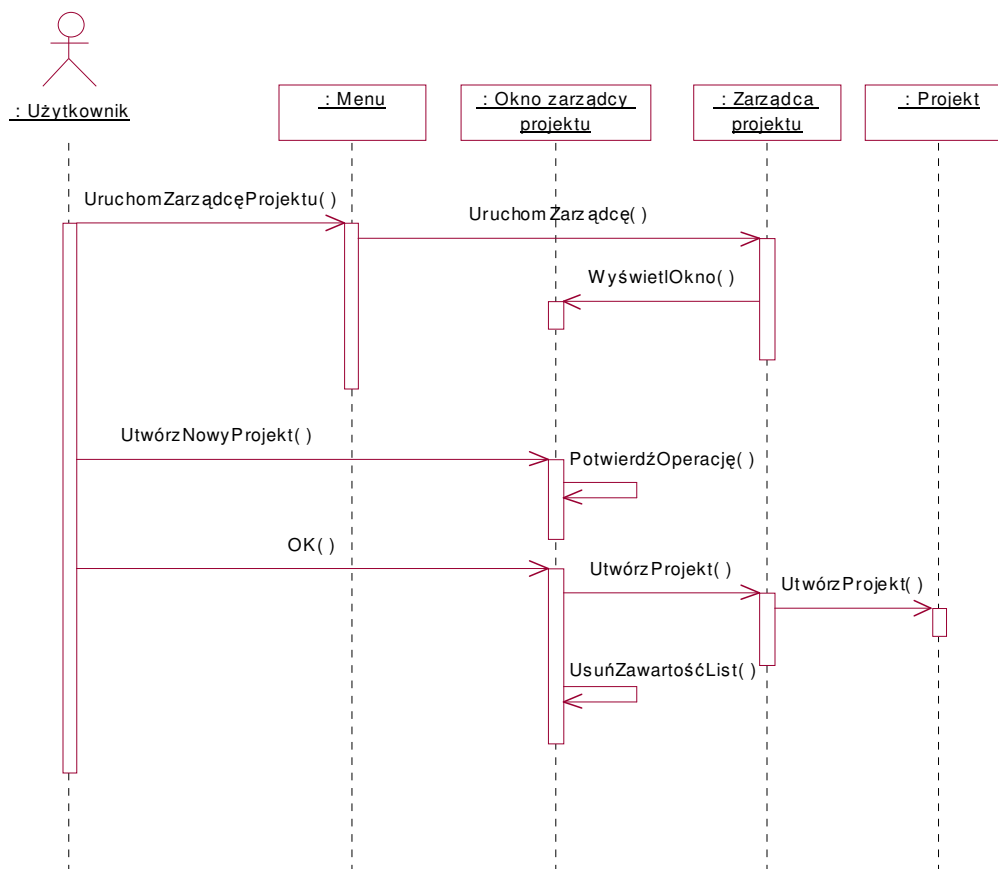
6.4 Diagram interakcji

Diagram przebiegu (sekwencji) i diagram kooperacji to rodzaje diagramu interakcji, na którym przedstawia się interakcję jako zbiór obiektów i związków między nimi, w tym też komunikaty, jakie obiekty przekazują między sobą. Diagram interakcji odnosi się do modelowania dynamicznych aspektów systemu. Diagram przebiegu obrazuje kolejność przesyłania komunikatów w czasie. Na diagramie kooperacji kładzie się nacisk na organizację strukturalną obiektów wymieniających komunikaty. Oba te diagramy można przekształcać jeden w drugi. Na diagramach interakcji uwzględnia się konkretne

i prototypowe egzemplarze klas, interfejsów, komponentów i węzłów, a także komunikaty przekazywane między nimi. Elementy te są rozpatrywane w kontekście pewnego scenariusza ilustrującego zachowanie systemu.

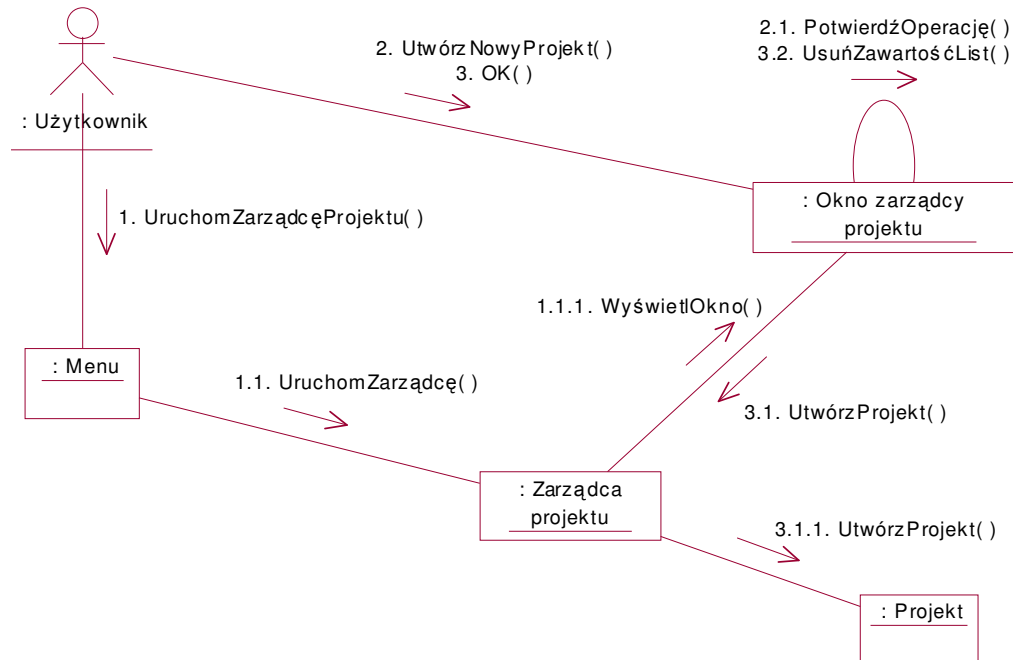
Diagram sekwencji jest połączeniem między światem funkcjonalnym i obiektywnym. Odpowiada konkretnemu scenariuszowi danego przypadku użycia. Główny nacisk położony jest na uwypuklenie kolejności komunikatów w czasie. W górnej części diagramu sekwencji, wzdłuż osi X, umieszczone są obiekty uczestniczące w interakcji. Obiekt inicjujący interakcję znajduje się zazwyczaj po lewej stronie diagramu, a coraz bardziej podrzędne kolejno po prawej. Komunikaty są uporządkowane w czasie wzdłuż osi Y, im późniejsza chwila wysłania, tym komunikat umieszczony niżej. Taki sposób obrazowania ułatwia zrozumienie przepływu sterowania w czasie.

Diagramy sekwencji mają dwie cechy, które odróżniają je od diagramów kooperacji. Po pierwsze występują na nich linie życia obiektów – pionowe przerywane kreski reprezentujące czas istnienia obiektów. Większość obiektów z diagramu interakcji żyje przez cały czas trwania interakcji. Znajdują się one w górnej części diagramu, a ich linie życia biegną od góry do dołu. Podczas interakcji mogą powstawać nowe obiekty. Ich linie życia rozpoczynają się w chwili odebrania przez nie komunikatu o stereotypie `create`. Pewne obiekty są niszczone. Ich linie życia kończą się w chwili odebrania przez nie komunikatu o stereotypie `destroy`. Drugą cechą odróżniającą diagram przebiegu od diagramu kooperacji jest uwzględnienie ośrodka sterowania. Jest to podłużny, cienki prostokąt reprezentujący okres wykonywania przez obiekt jakiejś akcji. Górna krawędź tego prostokąta znajduje się na tej samej wysokości co początek akcji, a dolna na wysokości zakończenia akcji. Zakończenie akcji może być dodatkowo oznaczone komunikatem przekazania. Można wyróżnić scentralizowany i zdecentralizowany sposób współpracy obiektów. Przy scentralizowanym sposobie wymiany komunikatów jeden z obiektów kontroluje cały przebieg przypadku, steruje operacjami i pośredniczy w wymianie danych. W przypadku zdecentralizowanego sposobu wymiany komunikatów nie ma obiektu kontrolującego i obiekty komunikują się ze sobą bezpośrednio. Przykładowy diagram przebiegu pokazany jest na rysunku (Rys. 28).



Rys. 28 Diagram przebiegu

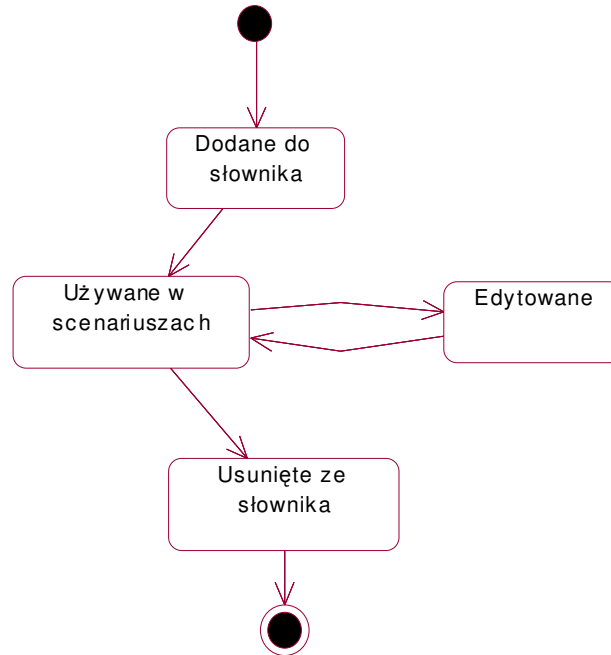
Istotą diagramu kooperacji jest przedstawienie przepływu komunikatów pomiędzy obiektami. Współpraca między obiektami łączy dwa aspekty: strukturę uczestniczących obiektów oraz sekwencję komunikatów wymienianych pomiędzy obiektami. Czas nie jest wprost odwzorowany, natomiast odwzorowane są powiązania pomiędzy obiektami. Na diagramie kooperacji uwypukla się organizację obiektów uczestniczących w interakcji. Obiekty są rozmieszczone jako wierzchołki grafu. Krawędziami grafu są wiązania łączące te obiekty. Od diagramu przebiegów odróżniają go dwie cechy. Po pierwsze, występują na nim ścieżki. Sposób połączenia jednego obiektu z drugim wskazuje się przez dodanie stereotypu ścieżki do drugiego końca wiązania. Po drugie, na diagramach kooperacji uwzględnia się ciąg komunikatów. Wskazanie kolejności komunikatu w czasie polega na poprzedzeniu go odpowiednim numerem w ciągu (pierwszy komunikat ma numer 1, a następne są ponumerowane kolejnymi liczbami naturalnymi). Zagnieżdżenia obrazuje się za pomocą notacji Doweya (1 oznacza pierwszy komunikat, 1.1 pierwszy komunikat zagnieżdżony w komunikacie numer 1 itd.). Zagnieżdżenie może mieć dowolną głębokość. Rysunek (Rys. 29) pokazuje przykładowy diagram kooperacji, który jest odpowiednikiem przedstawionego wcześniej diagramu przebiegu.



Rys. 29 Diagram kooperacji

6.5 Diagram stanów

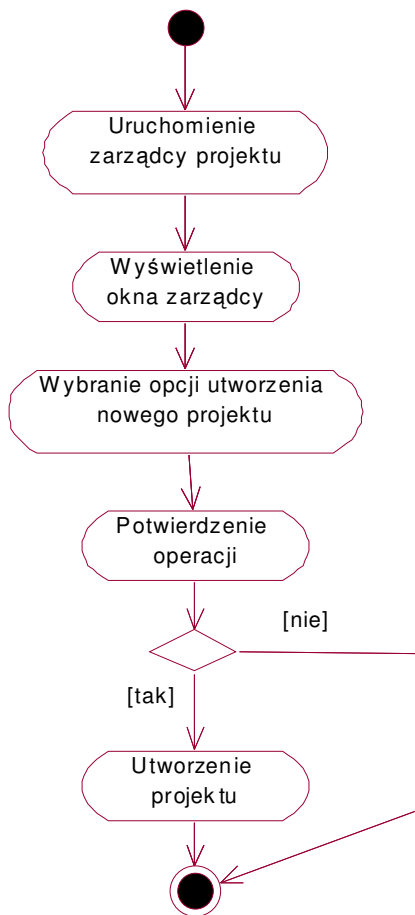
Diagram stanów nawiązuje do automatu skończonego, przedstawia maszynę stanową składającą się ze stanów, przejść, zdarzeń i czynności. Opisuje on stany pewnego procesu, które są istotne z punktu widzenia modelu pojęciowego tego procesu, oraz przejścia pomiędzy stanami wymuszane poprzez wywoływane usługi obiektu. Określa też reakcje obiektu na zdarzenia zachodzące podczas jego życia. Diagram stanów odnosi się do modelowania dynamicznych aspektów systemu. Jest szczególnie przydatny w modelowaniu zachowania interfejsów, klas i kooperacji. Przedstawia reakcje obiektów na ciągi zdarzeń i dlatego świetnie nadaje się do projektowania systemów interakcyjnych. Przykładowy diagram stanów pokazany jest na rysunku (Rys. 30).



Rys. 30 Diagram stanów

6.6 Diagram czynności

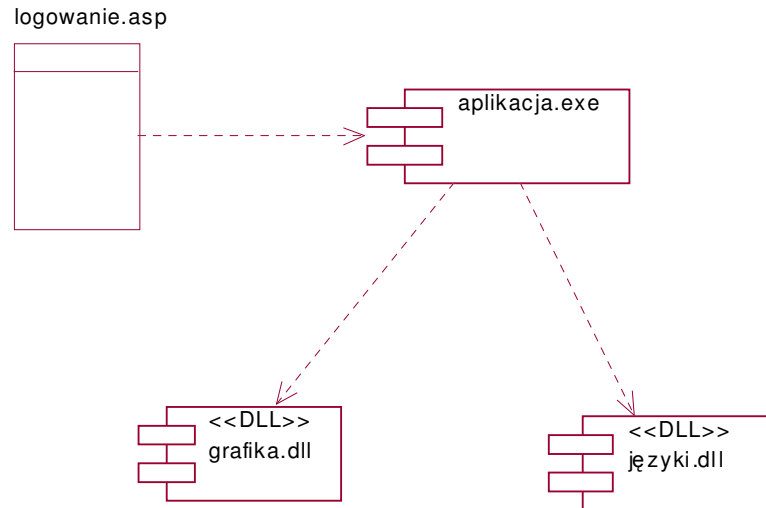
Diagram czynności to szczególny przypadek diagramu stanów, który obrazuje strumień kolejno wykonywanych czynności. Odnosi się do modelowania dynamicznych aspektów systemu. Jest bardzo przydatny w modelowaniu funkcji systemu. Kładzie się na nim nacisk na przepływ sterowania między obiektami. Większość diagramów czynności przedstawia sekwencyjne lub współbieżne kroki procesu obliczeniowego. Na diagramie czynności można także zobrazować zmiany zachodzące w obiekcie, gdy przechodzi on z jednego stanu do drugiego w różnych fazach przepływu sterowania. Rysunek (Rys. 31) pokazuje przykładowy diagram czynności.



Rys. 31 Diagram czynności

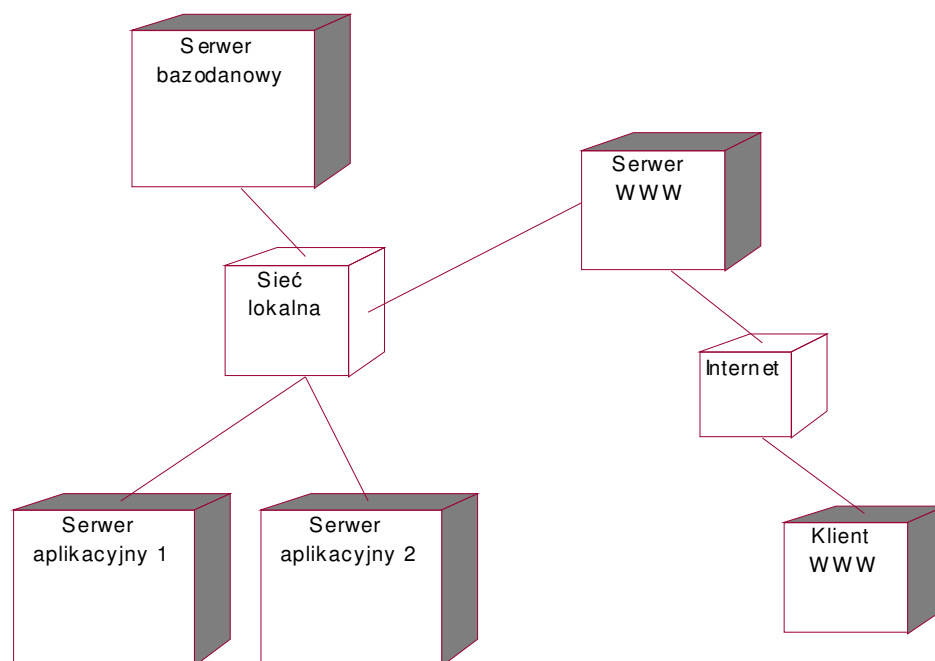
6.7 Diagram komponentów

Diagramy komponentów pokazują zależności pomiędzy komponentami oprogramowania, włączając komponenty kodu źródłowego, kodu binarnego oraz kodu wykonywalnego. Komponenty mogą istnieć w różnym czasie: niektóre z nich w czasie kompilacji, niektóre w czasie konsolidacji, inne w czasie wykonania. Diagram komponentów odnosi się do statycznych aspektów perspektywy implementacyjnej. Ściśle wiąże się z diagramem klas, ponieważ zwykle każdemu komponentowi są przyporządkowane pewne klasy, interfejsy i kooperacje. Główny nacisk położony jest na zarządzanie konfiguracją poszczególnych części systemu. Części te składają się z komponentów, które mogą być rozmaicie scalone w gotowy system. Przykładowy diagram komponentów pokazuje rysunek (Rys. 32).

**Rys. 32 Diagram komponentów**

6.8 Diagram wdrożenia

Diagram wdrożenia obrazuje konfigurację poszczególnych węzłów działających w czasie wykonania i zainstalowane na nich komponenty. Odnosi się do statycznych aspektów perspektywy wdrożeniowej. Wiąże się z diagramem komponentów, ponieważ zwykle każdy węzeł zawiera co najmniej jeden komponent. Umożliwia zbadanie układu procesorów i urządzeń, na których działa oprogramowanie. Przykład takiego diagramu pokazuje rysunek (Rys. 33).



Rys. 33 Diagram wdrożenia

Projektując nasz program korzystaliśmy z diagramu klas, diagramu przypadków użycia i diagramu przebiegu (sekwencji).

7 Podstawowe mechanizmy językowe w UML

UML jest prostszy dzięki czterem podstawowym mechanizmom, które są stosowane konsekwentnie w całym języku. Są to:

- 1) specyfikacje,
- 2) dodatki,
- 3) zasadnicze rozgraniczenia,
- 4) mechanizmy rozszerzania (wprowadzania nowych konstrukcji).

7.1 Specyfikacje

UML jest czymś więcej niż tylko graficznym językiem modelowania. Za każdym fragmentem graficznego zapisu kryje się specyfikacja definiująca składnię i znaczenie bloku konstrukcyjnego. Ikona klasy reprezentuje specyfikację określającą zestaw wszystkich atrybutów, operacji (łącznie z ich pełnymi sygnaturami) i funkcji, które ta klasa może spełniać. Symbol klasy nie musi wyobrażać całej specyfikacji, a tylko pewną niewielką jej część. Co więcej, symbol tej samej klasy może przedstawiać zupełnie inny zestaw jej składowych i będzie to całkowicie poprawne. Notacja graficzna UML służy do zobrazowania systemu, a specyfikacje do definiowania jego szczegółów. Dzięki takiemu podziałowi można przystąpić do przyrostowego konstruowania modelu – najpierw narysować diagramy, a następnie dodać do nich specyfikacje. Można także budować model od podstaw – zacząć od utworzenia specyfikacji np. za pomocą inżynierii wstecz, a potem opracować diagramy będące rzutami na ten zbiór specyfikacji.

7.2 Dodatki

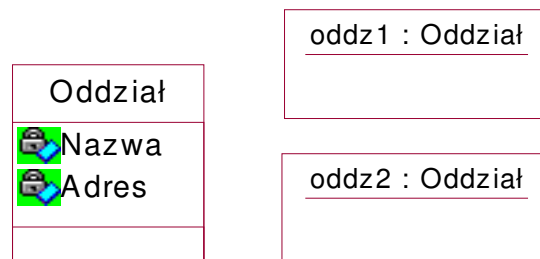
Większość bytów UML ma jedyną i niezależną postać graficzną, która oddaje najważniejsze ich aspekty. Symbol klasy jest na przykład tak zaprojektowany, aby można go było łatwo narysować, ponieważ jest najczęściej występującym składnikiem modeli obiektowych. Ten symbol podkreśla najważniejsze aspekty klasy, to znaczy nazwę, atrybuty i operacje. Specyfikacja klasy może zawierać także inne szczegóły, takie jak informacje o abstrakcyjności klasy lub widoczności poszczególnych atrybutów i operacji. Wiele z nich może być umieszczonych na diagramach jako graficzne lub tekstowe uzupełnienia prostokątnego symbolu klasy. Każdy element notacji UML składa się z symbolu podstawowego i rozmaitych charakterystycznych dla niego dodatków. Na rysunku (Rys. 34) widać przykład klasy z dodatkami wskazującymi, że jest to klasa abstrakcyjna z czterema operacjami: dwiema publicznymi, jedną chronioną i jedną prywatną.



Rys. 34 Dodatki

7.3 Zasadnicze rozgraniczenia

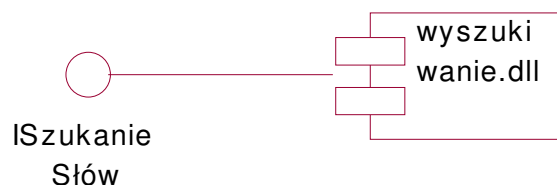
W modelowaniu systemów obiektowych świat jest podzielony na kilka sposobów. Po pierwsze, rozróżnia się klasę i obiekt. Klasa jest abstrakcją, a obiekt jest jednym konkretnym urzeczywistnieniem tej abstrakcji. W UML można modelować zarówno klasy, jak i obiekty (Rys. 35).



Rys. 35 Klasy i obiekty

Prawie z każdym blokiem konstrukcyjnym UML związana jest podobna zależność jak w przypadku klasy i obiektu. Można mieć przypadki użycia i egzemplarze przypadków użycia, komponenty i ich egzemplarze, węzły i ich egzemplarze. Graficznie symbol obiektu różni się od symbolu klasy tego obiektu jedynie tym, że nazwa obiektu jest podkreślona linią ciągłą.

Po drugie, rozróżnia się interfejs i implementację. Interfejs to deklaracja kontraktu, a implementacja to jedna z wielu konkretnych realizacji tego kontraktu. Wszystko musi przebiegać zgodnie ze znaczeniem interfejsu. W UML można modelować zarówno interfejsy, jak i ich implementacje. Na rysunku (Rys. 36) widać komponent wyszukiwanie.dll, który jest implementacją interfejsu ISzukanieSłów.



Rys. 36 Interfejsy i implementacje

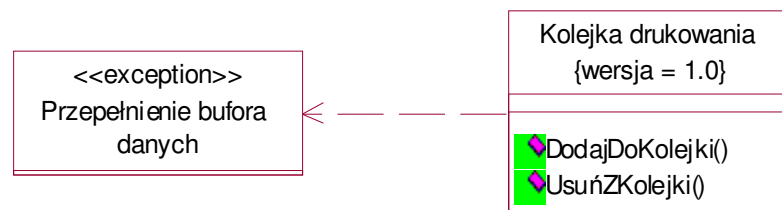
Podobnie jak w przypadku zależności klasa-obiekt, prawie z każdym blokiem konstrukcyjnym związana jest zależność interfejs-implementacja. Można mieć przypadki użycia i kooperacje, które je realizują, a także operacje i metody.

7.4 Mechanizmy rozszerzania

Język UML zapewnia standardowe środki wyrazu przydatne do zapisywania projektu systemu. Nie istnieje jednak tak uniwersalny język, w którym dałoby się wyrazić wszystkie możliwe niuanse każdego modelu dowolnego systemu w każdej dziedzinie zastosowania i w każdym czasie. Dlatego UML jest językiem otwartym. Można go rozszerzać, ale w kontrolowany sposób. Dostępne są następujące mechanizmy rozszerzania:

- stereotypy
- metki
- ograniczenia

Stereotyp umożliwia rozszerzanie słownictwa UML. Można tworzyć nowe bloki konstrukcyjne, wywodzące się z tych już istniejących, ale specyficzne dla danego zadania. Np. jeśli programujemy w języku C++ lub Java, to z pewnością chcemy uwzględnić w modelu wyjątki. W tych językach są one klasami, choć traktowanymi w szczególny sposób. Zwykle chcemy, aby wyjątki były jedynie zgłaszane i obsługiwane. Można więc sprawić, że w modelach będą traktowane jak standardowe bloki konstrukcyjne. Należy jedynie oznakować je odpowiednim stereotypem, tak jak zostało to zrobione z klasą `Przepełnienie bufora danych` na rysunku (Rys. 37).



Rys. 37 Mechanizmy rozszerzania

Metka umożliwia rozszerzanie listy właściwości bloku konstrukcyjnego UML. Można dodać nowe informacje do specyfikacji takiego bloku. Jeśli na przykład zajmujemy się tworzeniem produktów masowych, które wychodzą w wielu wersjach, to często zachodzi konieczność uwzględnienia informacji o wersjach i autorach pewnych istotnych abstrakcji. Informacje te (wersja i autor) nie są elementarnymi pojęciami UML, jednak można je dodać w postaci metki do dowolnego bloku konstrukcyjnego (np. do klasy). Na rysunku (Rys. 37) widać klasę `Kolejka drukowania` z jawnie podanym autorem i wersją.

Ograniczenie umożliwia rozszerzanie znaczenia bloku konstrukcyjnego UML. Można dodać nowe reguły lub zmodyfikować już istniejące. Można na przykład wprowadzić ograniczenie, że dodawanie zdarzeń do egzemplarzy klasy `Kolejka drukowania`

odbywa się z zachowaniem odpowiedniego porządku czy hierarchii użytkowników. Umieszcza się wtedy taką informację na diagramie w nawiasach klamrowych i łączy linią przerywaną z elementem, którego dotyczy to ograniczenie.

Te trzy mechanizmy rozszerzania umożliwiają dostosowanie UML do potrzeb konkretnego zadania i pozwalają na przystosowanie go do nowych technologii, takich jak na przykład coraz lepsze języki programowania systemów rozproszonych. Można dodawać nowe bloki konstrukcyjne, modyfikować specyfikacje już istniejących, a nawet zmieniać ich znaczenie. Należy jednak pamiętać, że UML służy przede wszystkim do przekazywania informacji, więc trzeba używać mechanizmów rozszerzania w sposób przemyślany i kontrolowany.