PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr 1094 — 2005
Informatyka Ekonomiczna 8

Monika Sitarska

WYKORZYSTANIE STANDARDU MDA (MODEL DRIVEN ARCHITECTURE) W MODELOWANIU BAZ DANYCH

1. Wstęp

Zarówno w literaturze, jak i w praktyce gospodarczej wiele miejsca poświęcono zagadnieniom dotyczącym modelowania. Zanim analitycy i projektanci bazy danych rozpoczną pracę nad jej budową, przeprowadzane są analizy mające na celu zdefiniowanie podstawowych potrzeb „systemowych" przyszłych użytkowników. Rozmowy z użytkownikami są często bardzo pracochłonne i jednocześnie mało efektywne, ponieważ użytkownicy często nie są w stanie w sposób precyzyjny określić kształtu przyszłego systemu, a tym bardziej samej bazy danych. Identyfikacja, a następnie analiza podstawowych funkcji bazy danych jest punktem wyjścia do rozpoczęcia dalszych prac. Praktyka pokazuje, że aby uniknąć nieporozumienia pomiędzy twórcami systemu a jego przyszłymi użytkownikami na pierwszym etapie prac budowane są modele baz danych.

Celem niniejszego artykułu jest charakterystyka podejścia MDA (Model Driven Architecture), zaproponowanego przez organizację Obiekt Management Group jako próba standaryzacji w zakresie modelowania kompleksowych rozwiązań informatycznych. W artykule poruszono także zagadnienia dotyczące modelowania obiektowych baz danych z wykorzystaniem języka UML.

2. Założenia standardu MDA

Innym bardzo istotnym argumentem, przemawiającym za wyborem obiektowej bazy danych, jest zaproponowany przez Object Management Group (OMG) standard budowy korporacyjnych systemów i baz danych, zorientowany na wysoki poziom abstrakcji modelu architektury rozwiązania (MDA).

Podstawową przesłanką utworzenia standardu MDA jest panujący na rynku oprogramowania chaos w zakresie standardów tworzenia oprogramowania oraz

potęgująca się konieczność budowania zintegrowanych rozwiązań informatycznych wymagających pełnej integracji z pozostałymi rozwiązaniami. Stwarza to oczywiście duże zagrożenie w postaci budowania zintegrowanych systemów związanych z pojedynczym dostawcą technologii, a w konsekwencji – ponoszenia horrendalnych kosztów budowania rozwiązania od podstaw w momencie „zestarzenia" się danego rozwiązania technologicznego.

OMG zaproponowało standard MDA, koncentrujący się na właściwościach funkcjonalnych i pracy rozproszonego rozwiązania, bez względu na uwarunkowania technologii, w której ostatecznie aplikacja lub system mają być zrealizowane. Rozdziela też szczegóły implementacyjne od funkcji biznesowych. Z drugiej strony MDA łączy ustanowione przez OMG standardy modelowania z Corbą, Javą, Net i innymi technologiami *middleware*, ułatwiając integrację aplikacji.

U podłoża architektury MDA leżą cztery standardy modelowania:

1. **Język UML** (Unified Modeling Language) – zunifikowany język zawierający pojęcia i notacje służące do obiektowej analizy, modelowania i projektowania. Przykłady notacji graficznych i ich zastosowanie zostaną omówione w dalszej części artykułu.
2. **Składnica do wymiany i przechowywania metadanych** (Meta Object Facility, MOF) – pozwala na opisanie podstawowych elementów, struktury oraz składni metamodeli wykorzystywanych do budowy zorientowanych obiektowo rozwiązań informatycznych. W skład specyfikacji MOF wchodzą:
   - abstrakcyjny model typowych obiektów oraz związków między nimi,
   - zbiór reguł dotyczących transformacji dowolnego metamodelu opartego na MOF na niezależną od języka oprogramowania definicję interfejsów,
   - reguły definiujące cykl życia oraz kompozycję elementów wykorzystywanych w metamodelach opartych na MOF.
3. **Język wymiany metadanych XMI** (XML Metadata Interchange).
4. **Metamodel hurtowni danych** (Common Warehouse Metamodel, CWM) – powszechny metamodel hurtowni danych definiuje metamodel reprezentujący dane biznesowe i dane techniczne często występujące w hurtowniach oraz systemach analitycznych. Standard ten jest wykorzystywany do wymiany instancji metadanych między heterogenicznymi rozwiązaniami informatycznymi. CWM składa się z metamodeli pozwalających na reprezentowanie źródeł danych, metod analizy danych, transformacji, wizualizacji oraz raportowania danych. Ponadto CWM definiuje metamodele reprezentujące standardowe procesy zachodzące w hurtowniach (ETL) (por. [Wilk]).

Konsekwencją omawianego ujęcia jest zaproponowanie przez OMG dwóch typów modeli.

Pierwszy z nich jest to niezależny od platformy model biznesowy (Platform Independent Model, PIM) dostarczający specyfikacji struktury i procesów biznesowych obsługiwanych przez modelowane rozwiązanie. Model ten w całości jest kompletnie niezależny od rozwiązań technologicznych. Drugi model jest to model definiujący platformę technologiczną (Platform Specific Model, PSM), który stanowi od-

wzorowanie modelu PIM na konkretną platformę technologiczną (np. Net), łącznie ze zbiorem definicji interfejsów dla tej platformy.

W niniejszej rozprawie główny nacisk został położony na zbudowanie modelu bazy danych marketingu niezależnego od platformy technologicznej. Do budowy tego modelu zostanie wykorzystane podejście zaproponowane przez OMG do tworzenia modelu PIM. Poniżej omówimy szczegółowo etapy budowy modelu niezależnego od platformy technologicznej (PIM):

1) identyfikacja procesów biznesowych wspieranych przez modelowane rozwiązanie – przypadki użycia,
2) opis statycznej struktury organizacji – klasy,
3) opis zachowań tworzonego rozwiązania (por. [Wilk]).

3. Modelowanie zakresu biznesowego aplikacji

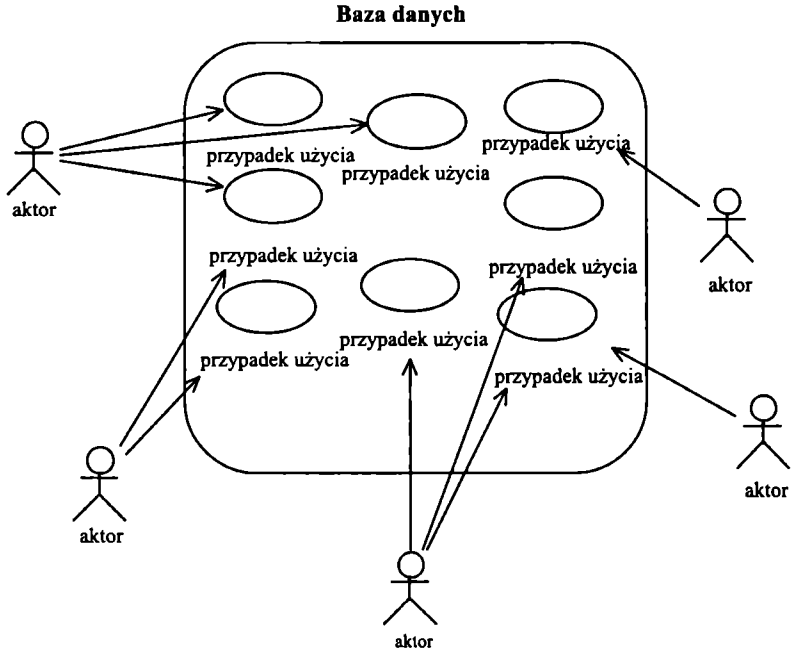
Pierwszym krokiem na długiej drodze, jaką jest budowa modelu bazy danych, jest zdefiniowane ram funkcjonalnych, a więc znalezienie odpowiedzi na pytanie, w jakim celu tworzone jest dane rozwiązanie. Należy pamiętać, że każde rozwiązanie informatyczne jest tworzone dla pewnej grupy użytkowników, aby wspomóc rozwiązanie jakiegoś konkretnego problemu biznesowego. Stąd też punktem wyjścia jest zawsze identyfikacja obszaru biznesowego rozwiązania informatycznego, a następnie zdefiniowanie funkcji i zachowań systemu. Analiza wstępna wymagań użytkowników jest przeprowadzana najczęściej w postaci wywiadów i rozmów z nimi po to, aby jak najdokładniej określić, które procesy będą obsługiwane przez system i w jakim zakresie. Dopiero na podstawie definiowanych wymagań można konstruować opis funkcji systemu, który jest odwzorowywany w postaci diagramów przypadków użycia (*use cases diagram*). Poprawnie zdefiniowane przypadki użycia opisują czynności, które powinien realizować system. Mają one szeroki wpływa na modelowanie i projektowanie bazy danych:

- Przypadek użycia reprezentuje pojedynczą, atomową transakcję realizowaną przez system. Znajomość przeprowadzanych transakcji pozwala na zaprojektowanie bazy danych – od strony zarówno logicznej jak i fizycznej.
- Przypadek użycia pokazuje, w jaki sposób system korzysta z danych elementarnych – zarówno od strony wewnętrznej (wstawianie, modyfikowanie, kasowanie), jak i na zewnątrz (poprzez zapytania). Znajomość przetwarzanych danych pozwala na zorganizowanie ich w perspektywy (widoki) i integrację w ogólnym schemacie koncepcyjnym. Przypadki użycia stanowią również podstawę do wprowadzania reguł biznesowych.
- Przypadki użycia pozwalają na mierzenie adekwatności bazy danych i stopnia, w jakim przyczynia się ona do funkcjonowania całego systemu.
- Przypadki użycia umożliwiają walidację gotowego produktu. W trakcie modelowania, projektowania i implementacji bazy możemy oceniać poprawność swojej pracy przez odniesienie konkretnych rezultatów do przypadków użycia warunkujących konkretne rozwiązania techniczne [Muller 2000, s. 89-90].

Na diagramie poza przypadkami użycia obecni są też aktorzy. Podręcznik *UML Notation Guide* definiuje to pojęcie następująco: „Aktor jest to rola pełniona przez obiekt lub obiekty zewnętrzne w stosunku do systemu, lecz wchodząca w bezpośrednią interakcję z nim jako część spójnej jednostki wykonawczej (przypadku użycia). Element typu aktor charakteryzuje funkcję pełnioną przez obiekt zewnętrzny; pojedynczy obiekt może pełnić wiele funkcji i być reprezentowany przez wielu aktorów [*Rational Software* 1997, s. 77].

Na diagramach UML aktorzy prezentowane są w postaci uproszczonej figurki człowieka. „Aktorzy stanowią kontekst przypadku użycia. Opracowanie szczegółowej listy aktorów pozwala na zdefiniowanie obiektów zewnętrznych i wewnętrznych, które będą wchodzić w interakcję z systemem. Aktor może być użytkownikiem, może jednak stanowić program albo urządzenie sprzętowe. Aktorzy pozwalają również na określenie sposobów użycia elementów bazy danych w przypadkach użycia. Opisując dany przypadek, można podać tabele, kolumny lub obiekty biorące udział w reprezentowanych przez nich transakcjach (por. [Muller, s. 90-92]).

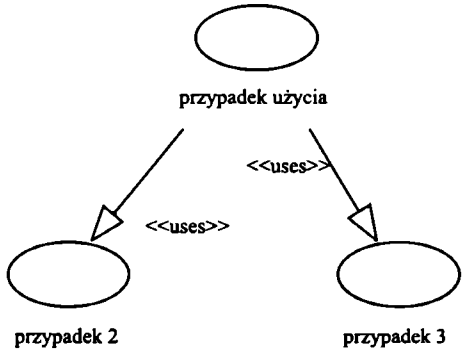
Po stworzeniu modelu aktorów w kolejnym kroku tworzone są **diagramy przypadków użycia**. Ukazują one związki między aktorami a przypadkami. Na rysunku 1 są widoczne granice bazy danych. Przypadki użycia są zaznaczone wewnątrz obwodu, natomiast aktorzy poza nim. Kolejnym krokiem jest przypisanie do każdego aktora tych wszystkich transakcji, w których bierze udział.



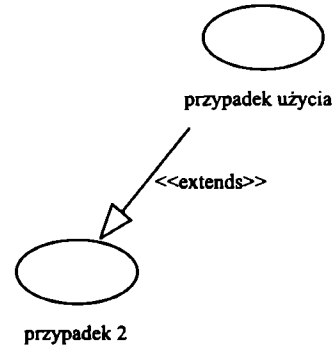
Rys. 1. Diagram przypadków użycia

Źródło: opracowanie własne.

W języku UML zdefiniowano dwa rodzaje relacji między przypadkami użycia: relację <<extends>> (rozszerza) oraz relację <<uses>> (wykorzystuje). Relacja rozszerzenia <<extends>> określa przypadek użycia, który może zostać rozszerzony przez dodatkowe operacje stanowiące część innego przypadku użycia. Relacja ta (rys. 2) składa się z warunku rozszerzenia oraz referencji do punktu rozszerzenia w rozszerzanym przypadku (miejsca, w którym mogą zostać wprowadzone dodatkowe operacje). W momencie osiągnięcia przez instancję danego przypadku miejsca stanowiącego punkt rozszerzenia sprawdzana jest wartość logiczna warunku i, jeśli jest on spełniony, przypadek ten jest rozszerzany o przypadek rozszerzający [*Rational...* 1997, s. 95].



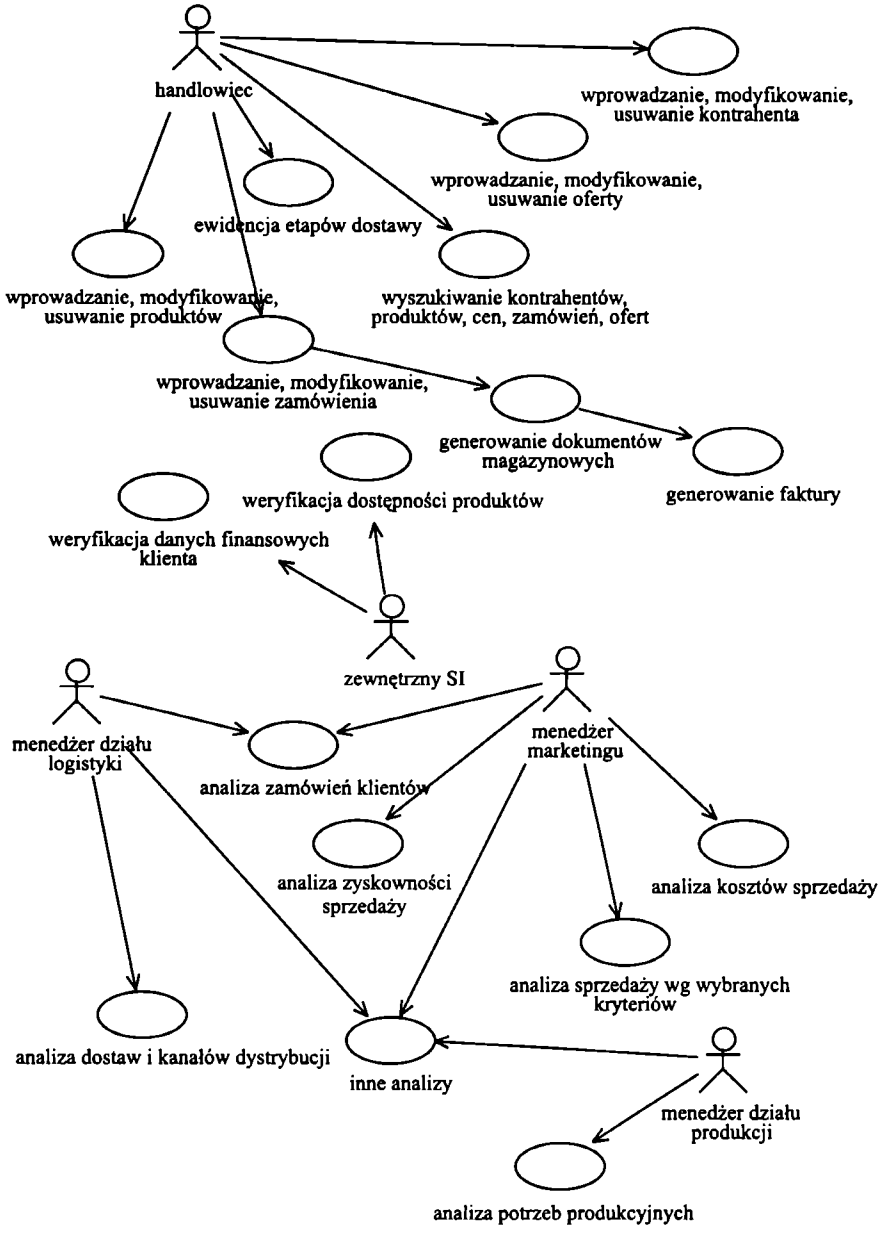
Rys. 2. Diagram przypadków użycia – relacja rozszerzenia
Źródło: opracowanie własne.



Rys. 3. Diagram przypadków użycia – relacja rozszerzenia
Źródło: opracowanie własne.

Relacja wykorzystywania <<uses>> oznacza, że sekwencja operacji zawarta w przypadku wykorzystywanym jest częścią innego przypadku (przypadku wykorzystującego). Przypadek ten może wprowadzać własne operacje w dowolnym miejscu tej sekwencji, pod warunkiem że nie zmienia kolejności operacji wykorzystywanych. Jeżeli dany przypadek wykorzystujący jest powiązany z kilkoma przypadkami wykorzystywanymi, wówczas jego sekwencja operacji będzie wynikiem scalenia sekwencji przypadków wykorzystywanych oraz własnych poleceń przypadku wykorzystującego. Przykład notacji prezentuje rys. 3.

Kompletna charakterystyka przypadków użycia obejmuje jeszcze tzw. streszczenie, czyli krótki, składający się z kilku zdań, opis mówiący o tym, czego dotyczy dany przypadek. Opis słowny dla niektórych osób może być bardziej czytelny niż najbardziej przejrzyste schematy, dlatego też kompletna charakterystyka przypadku użycia obejmuje zarówno opis, jak i diagram. Na rysunku 4 został zamieszczony przykład diagramu przypadków użycia dla obszaru funkcjonalnego dotyczącego sprzedaży. Na diagramie zaprezentowano pojedyncze przypadki użycia, które może realizować użytkownik obszaru sprzedaży.



Rys. 4. Diagram przypadków użycia w procesie sprzedaży
Źródło: opracowanie własne.

Wyróżnione przypadki użycia zostały scharakteryzowane poniżej:

- Wprowadzanie, modyfikowanie i usuwanie danych kontrahenta – jednymi z podstawowych danych przechowywanych w bazie danych (BDM) są dane dotyczące

kontrahentów (klientów, partnerów i dostawców firmy), stąd też moduł sprzedaży umożliwia wprowadzanie, modyfikowanie i usuwanie danych o kontrahentach.

- Wprowadzanie, modyfikowanie i usuwanie danych produktu i cen – proces sprzedaży obejmuje także listę produktów i usług będących w ofercie danej firmy, dlatego jednym z zadań jest możliwość wprowadzania danych o produktach i usługach.
- Wprowadzanie, modyfikowanie i usuwanie danych oferty – użytkownik ma możliwość wprowadzania, modyfikowania i usuwania danych oferty oraz przypisywania do niej odpowiednich produktów i klienta.
- Wprowadzanie, modyfikowanie i usuwanie danych zamówienia – użytkownik ma możliwość wprowadzenia do modułu nowego rekordu zamówienia, jego modyfikacji lub usunięcia. Podobnie jak do oferty, istnieje również możliwość przypisania produktów i klientów do zamówienia.
- Wyszukiwanie kontrahenta, produktu, oferty, zamówienia – przypadek ten umożliwia wyszukiwanie danych obiektów znajdujących się w module sprzedaży.
- Weryfikacja dostępności produktów – system umożliwia automatyczną kontrolę stanów magazynowych oraz przekazuje informacje o dostępności produktów.
- Weryfikacja danych finansowych kontrahenta – w trakcie generowania dokumentów handlowych (faktur) dane finansowe kontrahenta, takie jak termin płatności, należności i zobowiązania, limit kredytowy itp., są automatycznie weryfikowane.
- Generowanie dokumentów magazynowych – na podstawie zamówień system automatycznie generuje dokumenty magazynowe.
- Generowanie faktury – na podstawie dokumentów magazynowych system automatycznie generuje fakturę dla danego kontrahenta.
- Analiza zamówień klientów – prowadzi się ją na potrzeby menedżerów sprzedaży, marketingu i produkcji.
- Analiza kosztów sprzedaży – prowadzi się ją na potrzeby menedżerów sprzedaży, marketingu i produkcji.
- Analiza potrzeb produkcyjnych – na potrzeby menedżerów sprzedaży, marketingu i produkcji.
- Analiza wielkości sprzedaży według wybranych kryteriów – na potrzeby menedżerów sprzedaży, marketingu i produkcji.
- Analiza zyskowności sprzedaży – na potrzeby menedżerów sprzedaży, marketingu i produkcji.
- Wyliczanie wskaźników sprzedaży – do podstawowych wskaźników należą: wskaźnik RFM (*recency, frequency, monetary*), wskaźnik kosztu sprzedaży CPS, wartość życiowa klienta CLV, wskaźnik utraty klientów, wskaźnik dynamiki wzrostu sprzedaży, wskaźnik utrzymania klientów.
- Analiza koszykowa – modelowanie grup produktów lub usług, które są kupowane przez klientów jednocześnie lub w określonej sekwencji.
- Inne analizy – realizowane na potrzeby menedżerów sprzedaży, marketingu i produkcji.

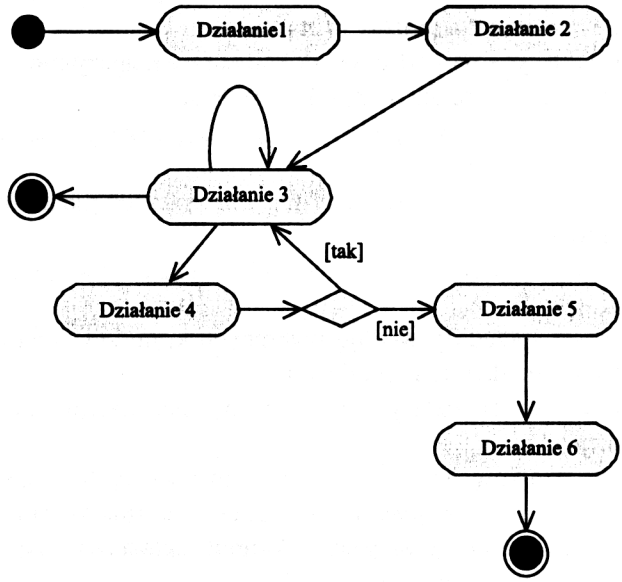
Etap modelowania obszaru biznesowego charakteryzuje wysoki poziom abstrakcji opisu, ponieważ chodzi o nakreślenie obszaru biznesowego wspieranego przez rozwiązanie informatyczne. W kolejnych fazach jest on uszczegóławiany.

Uzupełnieniem diagramów przypadków użycia, które opisują sekwencje zdarzeń wchodzących w ich skład, są **diagramy działań** *(activity diagram)*. Jest to pierwsza forma opisu zachowań systemu na bardzo wysokim poziomie abstrakcji, ukazująca kolejne kroki przebiegu procesu biznesowego. Każdy krok jest efektem wykonania jakiegoś działania (*activity state*). Na diagramie można zaprezentować poszczególne działania i przejścia (*transition*) między nimi oraz równoległe, zależne od zaistniałych warunków scenariusze.

Identyfikację działań przeprowadza się na podstawie opisów przypadków użycia. W opisie powinny być zawarte informacje wskazujące na działania, które powinny zajść w czasie realizacji przypadku użycia. Dokładna analiza danego przypadku powinna też wskazać na ewentualne alternatywne ścieżki jego przebiegu.

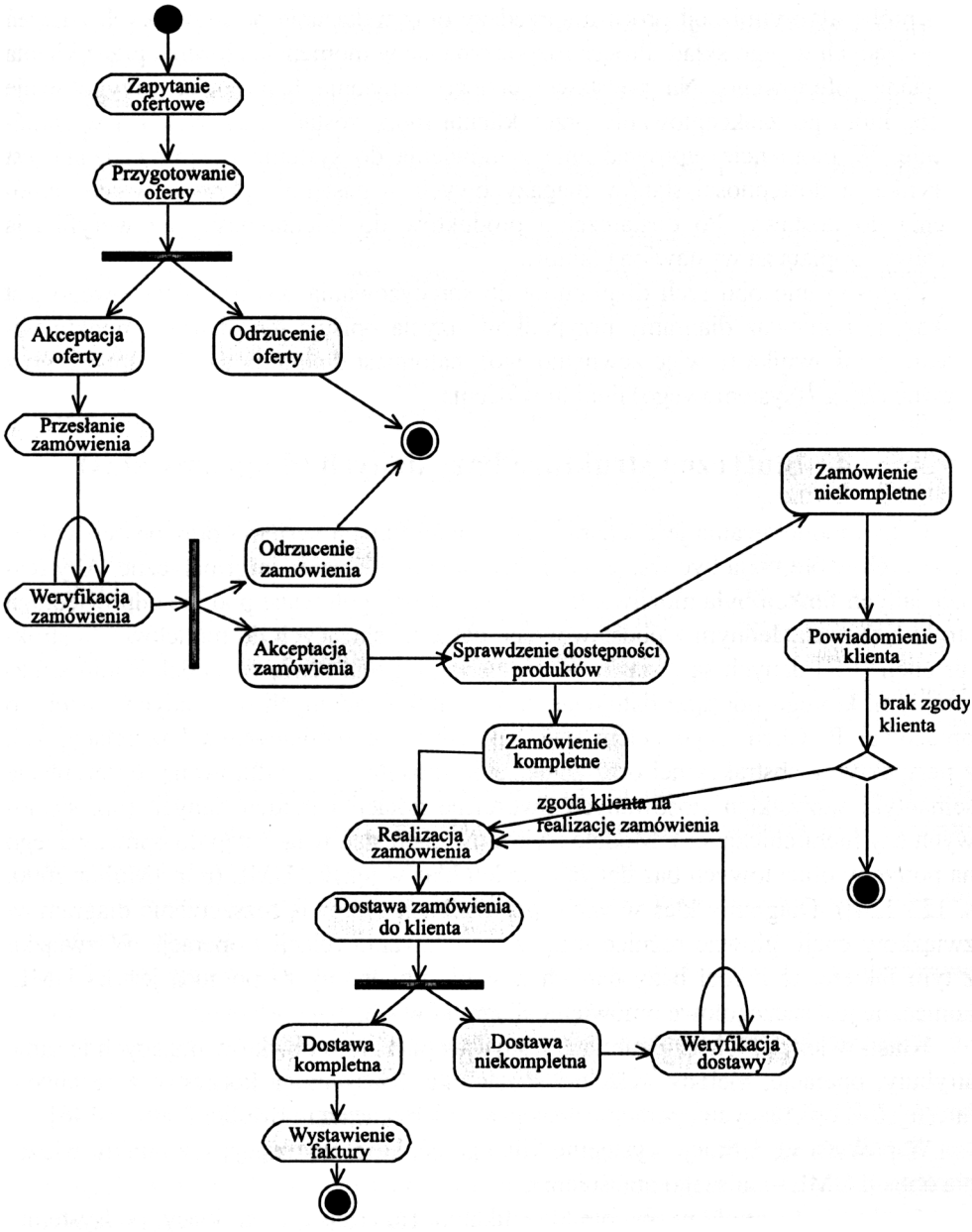
Każdy z diagramów rozpoczyna stan inicjujący (*initial state*), reprezentowany przez czarną kropkę. Stan zamykający (*final sate*) jest reprezentowany przez okrąg z czarną kropką w środku. W diagramach dopuszczalne jest występowanie kilku stanów kończących, zamykających alternatywne ścieżki przejść między działaniami.

Samo działanie reprezentowane jest przez przez „prostokąt z zaokrąglonymi rogami" (rys. 5).



Rys. 5. Diagram działań

Źródło: opracowanie własne.



Rys. 6. Diagram działań dla procesu sprzedaży
Źródło: opracowanie własne.

Pełny opis diagramu działań obejmuje także słowną charakterystykę narysowanego procesu. Rysunek 6 prezentuje przykład diagramu działań wykonany także dla obszaru sprzedaży, podobnie jak diagram przypadków użycia. Diagram ten ma na

celu próbę algorytmizacji procesu sprzedaży oraz wskazanie podstawowych zdarzeń wchodzących w jego skład. Proces rozpoczyna się w momencie złożenia przez klienta zapytania ofertowego. Na podstawie takiego zapytania handlowiec przygotowuje ofertę, która po zaakceptowaniu przez klienta może zostać przekształcona w zamówienie. W momencie wprowadzania zamówienia do systemu przeprowadzana jest weryfikacja dostępności stanów magazynowych, a następnie przygotowanie zamówienia do dostawy. Po dostarczeniu produktów do klienta następuje weryfikacja dostawy i zapłata za wystawioną fakturę.

Zastosowanie obu tych diagramów do sprecyzowania obszaru biznesowego jest wskazane ponieważ diagramy przypadków użycia opisują ten obszar z punktu widzenia użytkownika (a więc zewnętrznego), natomiast diagramy działań opisują go z wewnętrznego (systemowego) punktu widzenia.

4. Statyczna struktura bazy danych (diagramy klas)

Celem modelowania jest scharakteryzowanie funkcji systemu oraz procesów biznesowych, które mają być wspierane przez dane rozwiązanie informatyczne. Aby realizacja tych funkcji była możliwa, konieczne jest przygotowane pod nie odpowiednich struktur danych. Jednym z podstawowych narzędzi służących do modelowania struktur encji (ER) danych są związki. Korzenie tego ujęcia tkwią w modelowaniu strukturalnym, którego początki datuje się na początek relacyjnych baz danych. Ujęcie to opracował P. Chen. Jego celem było umożliwienie modelowania baz relacyjnych z perspektywy abstrakcyjnej oraz znalezienie sposobu na zunifikowaną reprezentację semantyki wszystkich stosowanych ówcześnie notacji i modeli danych (np. sieciowych czy hierarchicznych). Modele związków encji stanowią wstęp do opracowanego na potrzeby obiektowych baz danych modeli klas w języku UML (por. [Muller 2000, s. 123-124]). Diagramy klas w wielu przypadkach stanowią rozszerzenie diagramów związków encji, główne różnice leżą w modelowaniu relacji i operacji. W związku z tym faktem, iż model bazy danych zostanie utworzony za pomocą języka UML, konieczne jest szczegółowe omówienie diagramów klas.

Klasa w języku UML rozumiana jest jako opis zbioru obiektów mających te same atrybuty, operacje, metody, relacje i semantykę. Klasy mogą korzystać z zestawów interfejsów, opisujących operacje udostępniane ich otoczeniu [Muller 2000, s. 146].

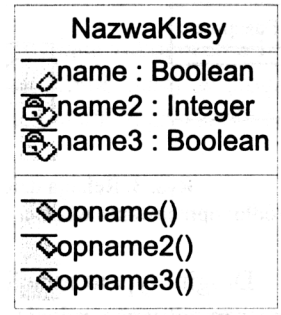
W powyższej definicji występuje kilka pojęć, które wymagają – z punktu widzenia notacji UML – dalszego objaśnienia.

- Atrybut jest to pole w obrębie klasyfikatora (interfejsu, typu, klasy, podsystemu, bazy danych lub komponentu), reprezentujące zakres wartości, które mogą być przechowywane w instancji klasyfikatora.
- Operacja jest to usługa, której można zażądać od obiektu. Każda operacja posiada sygnaturę (nazwę i parametry, łącznie z opcjonalnym parametrem zwracanym przez operację), z możliwymi ograniczeniami dopuszczalnych parametrów wejściowych.

- Metoda jest to implementacja operacji, zawierająca algorytm służący do osiągnięcia rezultatów operacji.
- Relacja jest to semantyczne powiązanie pomiędzy elementami modelu. Przykładami relacji są asocjacje i generalizacje.
- Interfejs jest to deklaracja kolekcji operacji służących do realizowania usługi udostępnianej przez instancję [Muller 2000, s. 146-147].

Symbolem klasy jest prostokąt podzielony co najwyżej na trzy części (rys. 7). W górnej części zawarta jest nazwa klasy i jej właściwości. W środkowej części umieszcza się listę atrybutów i ich własności, natomiast najniższy poziom zawiera opis operacji i ich własności.

Klasy, które mają być trwale przechowywane w bazie, należy opatrzyć stereotypem <<persistent>>. Stereotyp ten oznacza, że stan instancji nie powinien zostać zniszczony w momencie zniszczenia samej instancji. Klasy oznaczone jako <<persistent>> mają wewnętrzną implementację w bazie danych w postaci tabel relacyjnych, typów obiektowo-relacyjnych lub trwałych klas obiektowych. Istnieją również klasy nie mające instancji, które nazywane są **klasami abstrakcyjnymi.** Są one stosowane głównie do reprezentacji atrybutów i operacji współdzielonych przez szereg klas potomnych [Muller 2000, s. 156].



Rys. 7. Struktura prostej klasy
Źródło: opracowanie własne.

Atrybuty i operacje charakteryzujące klasę są opisywane przez kwalifikatory, takie jak dostępność, nazwa, typ danych, ograniczenia i właściwości. Dostępność atrybutów i operacji określana jest przez trzy symbole:

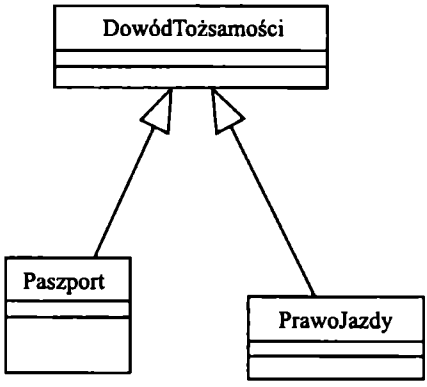
1) dostępność **publiczna** (+) oznacza, że każda inna klasa może dowolnie wpływać na wartość danego atrybutu,

2) dostępność **chroniona** (#) oznacza, że dany atrybut może być obserwowany tylko przez jego klasę macierzystą lub jej podklasy,

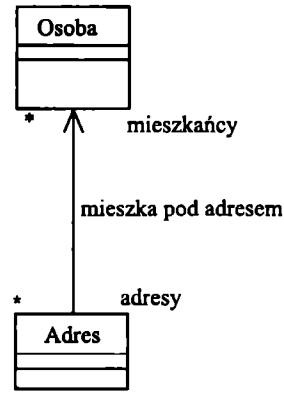
3) dostępność **prywatna** oznacza, że tylko metody klasy macierzystej mogą odczytywać i modyfikować dany atrybut [Muller 2000, s. 157].

Poszczególne klasy powiązane są między sobą relacjami. Język UML wyróżnia trzy typy relacji: generalizację, asocjację i agregację. Relacja **generalizacji** jest to związek między elementem bardziej ogólnym a elementem bardziej szczegółowym. Element szczegółowy jest w pełni spójny z elementem ogólnym, może jednak zawierać dodatkowe informacje. Relacja ta obrazuje związek między typami jako specjalizację pewnego typu wyjściowego. Relacje generalizacji między klasami wyznaczają hierarchię dziedziczenia. **Dziedziczenie** oznacza, że każdy podtyp (klasa szczegółowa) dziedziczy operacje i atrybuty swojego nadtypu (klasy ogólnej) i może wprowadzać własne operacje i atrybuty. Na rysunku 8 zawarty jest przykład relacji:

zarówno paszport jak i prawo jazdy są szczególnymi przypadkami dokumentu tożsamości.



Rys. 8. Relacja dziedziczenia
Źródło: opracowanie własne.



Rys. 9. Relacja asocjacji
Źródło: opracowanie własne.

Drugim typem relacji definiowanym na diagramach klas jest **asocjacja**. Oznacza ona semantyczną relację miedzy dwiema klasami lub większą ich liczbą, opartą na powiązaniach między ich instancjami. Asocjacje obrazują związki, które obiekty danej klasy mają z pozostałymi obiektami w systemie.

Asocjacje składają się z ról, które bardziej szczegółowo charakteryzują związek między obiektami. Asocjacje binarne mają po dwie role, a asocjacje wyższych rzędów – analogicznie więcej. Inną cechą, charakteryzującą rolę, jest jej liczebność. Język UML wprowadza pewne rozszerzenia w przypadku liczebności w stosunku do diagramów encji. Poza klasycznymi relacjami jeden-do-jednego (1..1), jeden-do-wiele (1..*) i wiele-do-wiele (*..*) wprowadzone zostały następujące typy liczebności ról:

- 0..*: zero lub wiecej obiektów
- 0..1: zero lub jeden obiekt
- 1..*: przynajmniej jeden obiekt
- *: zero lub więcej obiektów.

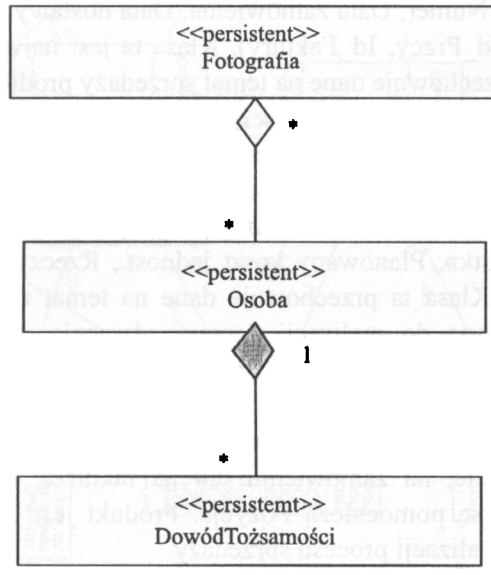
Na rysunku 9 zaprezentowano notację UML dla asocjacji.

Połączenie klas relacją asocjacji i przypisanie tym relacjom określonych ról pozwala na zdefiniowanie dopuszczalnych ścieżek przejść między klasami (nawigacji). Możliwe jest zdefiniowanie konkretnego kierunku przejścia poprzez dodanie na diagramie strzałki, która definiować będzie wzajemną widoczność klas. Domyślnie w diagramach UML stosuje się asocjacje niekierowane, oznaczające obustronną widoczność klas. Konsekwencją wprowadzania kierunków przejść jest asymetryczna widoczność klas, która oznacza, że jedne klasy mogą „widzieć" inne, same nie będąc widoczne.

Innym zagadnieniem, którego nie można w jednoznaczny sposób zaprezentować bazie w relacyjnej, jest relacja typu całość-część (**agregacja**), łącząca dwie klasy. Relacja ta obrazuje związek między klasami, z których jedna stanowi cześć drugiej. Wykorzystanie tej relacji ma szczególne znaczenie w modelowaniu baz, których obiektami są produkty, ich części zamienne oraz podzespoły. W obiektowych bazach danych zostały wyróżnione dwa typy agregacji:

1. Agregacja słaba reprezentuje sytuację, w której pewien obiekt jest właścicielem innego obiektu, ale ten drugi obiekt może jednocześnie mieć wielu innych właścicieli.
2. Agregacja silna jest stosowana w sytuacji, gdy pewien obiekt stanowi wyłączną własność innego obiektu [Muller 2000, s. 185].

W UML agregacja jest reprezentowana przez romby stanowiące zakończenia linii oznaczających relacje po stronie klas będących właścicielami. Romb pusty w środku symbolizuje słabą agregację, a czarny romb – silną. Notacja została zaprezentowana na rys. 10.



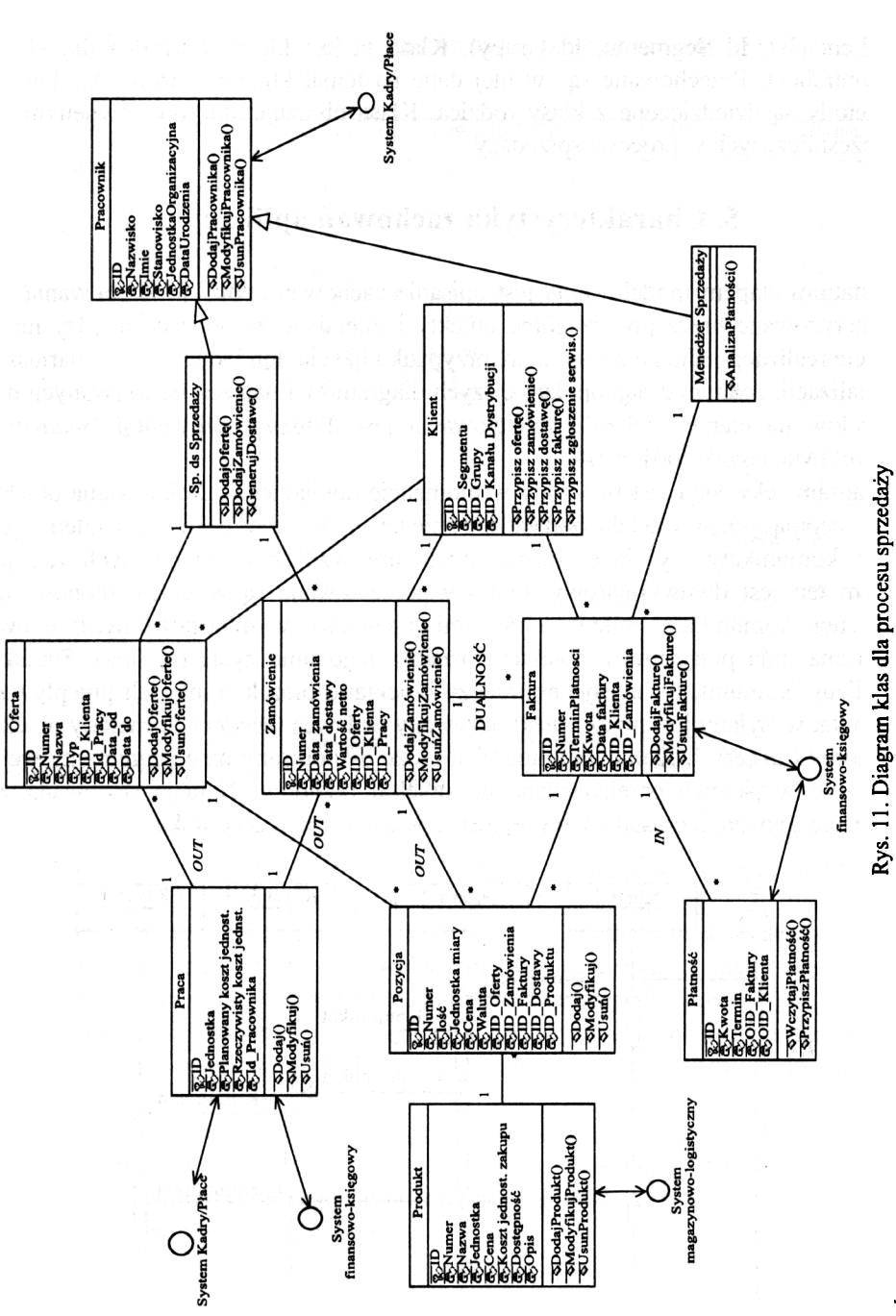
Rys. 10. Asocjacja mocna i asocjacja słaba – notacja
Źródło: opracowanie własne.

W niniejszym przykładzie agregacja słaba oznacza, że Fotografia agreguje pewien zbiór Osób, natomiast każda Osoba może pojawić się na wielu Fotografiach (relacja wiele-do-wielu). Fotografia nie jest właścicielem Osoby, co oznacza, że po usunięciu Fotografii żadna Osoba nie jest usuwana z bazy. Natomiast agregacja silna oznacza, że Osoba jest właścicielem DowoduTożsamości, co powoduje, że usunięcie Osoby z bazy skutkuje usunięciem także wszystkich powiązanych z nią DowodówTożsamości.

Szczegółowy i poprawnie zdefiniowany diagram klas umożliwia zbudowanie modelu bazy danych, który to model na etapie implementacji zostaje zaadaptowany na potrzeby odpowiedniego rozwiązania technologicznego. Jednak zbudowanie modelu bazy danych pozwala, zgodnie z założeniami koncepcji MDA, na uniezależnienie się od platformy technicznej, co skutkuje możliwością implementacji danego modelu na różnych platformach technologicznych. Jeżeli zachodzi potrzeba uaktualnienia platformy lub wymiany na inną (np. zmiany serwera aplikacyjnego), model niezależny od platformy może być nadal wykorzystywany. Na rysunku 11 został zamieszczony przykład diagramu klas dla procesu sprzedaży charakteryzowanego przez dwa poprzednie diagramy (rys. 4 i rys. 6).

Kompletny diagram klas zawiera także opis poszczególnych klas. Przykład zamieszczono poniżej:

- Oferta (ID, Numer, Nazwa, Typ, Id_Klienta, Id_ Pozycji, Id_Pracy, Data_Od, Data_Do). W klasie tej przechowywane są dane na temat ofert opracowywanych dla klientów firmy.
- Zamówienia (ID, Numer, Data zamówienia, Data dostawy, Wartość, Id_Klienta, Id_ Pozycji, Id_Pracy, Id_Faktury). Klasa ta jest najważniejszym elementem diagramu, przechowuje dane na temat sprzedaży produktów i usług.
- Faktura (ID, Numer, Termin płatności, Kwota, Id_Pozycji, Id_Klienta, Id_Zamówienia). W klasie tej przechowywane są dane na temat faktur wystawianych klientom za złożone zamówienia. Jest ona wspólna dla modułu sprzedaż i serwis oraz dla systemu finansowo-księgowego.
- Praca (ID, Jednostka, Planowany koszt jednost., Rzeczywisty koszt jednost., Id_Pracownika). Klasa ta przechowuje dane na temat ilości i jakości pracy, która została zużyta do realizacji danego zdarzenia. Klasa ta łączy się z systemem kadry/płace poprzez klasę Pracownik.
- Produkt (ID, Numer, Nazwa, Jednostka, Cena, Koszt jednst. zakupu, Dostępność, Opis). W klasie tej przechowywane są dane na temat produktów występujących na ofercie, na zamówieniu lub na fakturze. Łączy się z klasami zdarzeń przez klasę pomocniczą Pozycja. Produkt jest także zasobem zużywanym w trakie realizacji procesu sprzedaży.
- Płatność (ID, Kwota, Termin, Id_Faktury). Klasa ta przechowuje dane na temat płatności poszczególnych faktur przez klienta. Jest to zasób zwiększany w trakcie realizacji procesu sprzedaży.
- Specjalista ds. sprzedaży i Menedżer ds. sprzedaży (ID, Nazwisko, Imię, Stanowisko, Jednostka organizacyjna). Klasy te są klasami podrzędnymi dla kasy Pracownik (relacja generalizacji), przechowują dane na temat pracowników operacyjnych działu sprzedaży. Te klasy obrazują aktorów wewnętrznych.



Rys. 11. Diagram klas dla procesu sprzedaży

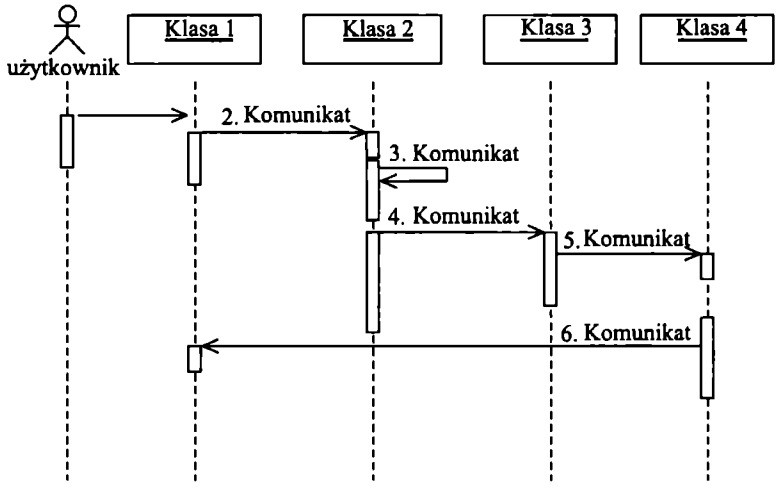
Źródło: opracowanie własne.

- Klient (ID, Id_Segmentu, Id_Grupy). Klasa ta jest klasą podrzędną dla klasy Kontrahent. Przechowane są w niej dane na temat klientów firmy. Atrybuty i metody są dziedziczone z klasy rodzica. Klasa obrazuje aktorów zewnętrznych uczestniczących w procesie sprzedaży.

5. Charakterystyka zachowań aplikacji

Ostatnim etapem modelowania jest opisanie zachowań aplikacji. Zachowania są charakteryzowane przez poszczególne obiekty i interakcje zachodzące między nimi w trakcie realizacji jednego zadania, np. przypadku użycia, lub jednego ze scenariuszy jego realizacji. Jednym z najpopularniejszych diagramów i najczęściej używanych do tych celów na etapie analizy i modelowania jest **diagram sekwencji** (*sequence diagram*) [Maciaszek 2001, s. 64].

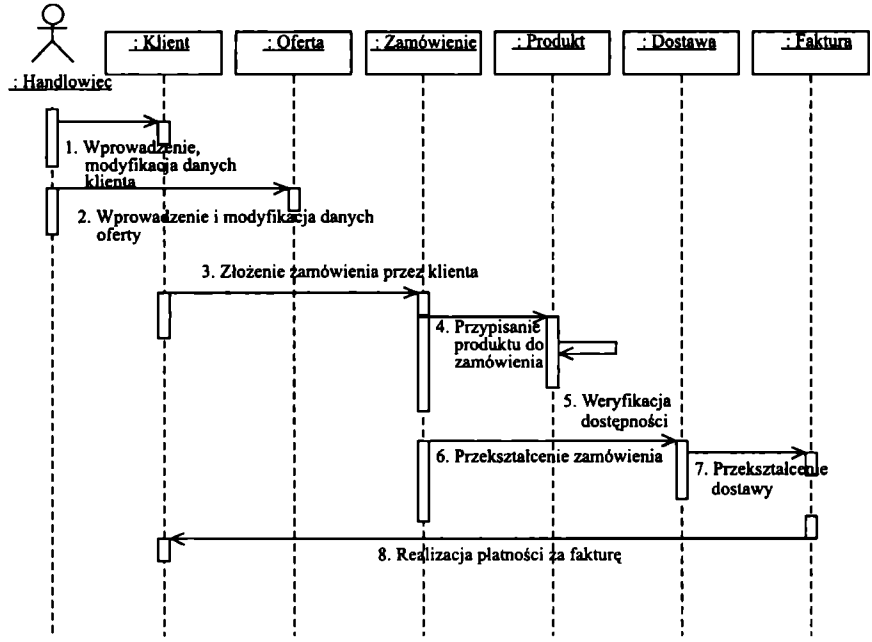
Diagram sekwencji jest tworzony w momencie dokładnego zdefiniowania obiektów występujących w modelu oraz relacji między nimi, gdyż opisuje on interakcje, a więc komunikaty wysyłane między obiektami wzdłuż zdefiniowanych relacji. Diagram ten jest dwuwymiarowy. Obiekty prezentowane są w ujęciu pionowym, a sekwencję komunikatów między obiektami przedstawia wymiar poziomy. Pionowa przerywana linia połączona z obiektem oznacza jego linie życia (*lifeline*). Strzałki symbolizują komunikaty przesyłane między obiektami oraz kierunki ich przepływu. Komunikat wysyłany jest przez obiekt nazywany nadawcą (*sender*) do metody obiektu odbiorcy (*target*). Dodatkowo komunikat może być opisany przez argumenty wejściowe lub wyjściowe przekazywane do obiektu odbiorcy. Notacja stosowana w diagramach sekwencji opisana powyżej jest zaprezentowana na rysunku 12.



Rys. 12. Diagram sekwencji

Źródło: opracowanie własne.

Przykład diagramu sekwencji dla procesu sprzedaży prezentuje rys. 13.



Rys. 13. Diagram sekwencji dla procesu sprzedaży
Źródło: opracowanie własne.

Przejście przez opisane wyżej trzy etapy prac umożliwia zbudowanie kompletnego niezależnego od platformy technologicznej modelu (PIM) rozwiązania informatycznego, które ma wspierać konkretny obszar biznesowy. Ujęcie to ma zapewnić użytkownikom biznesowym możliwość jednokrotnego zdefiniowania potrzebnych im własności funkcjonalnych i zachowania aplikacji w formie standardowego modelu PIM, a następnie implementacji tego modelu na płaszczyźnie konkretnych rozwiązań technologicznych. Implementacja taka może następować wielokrotnie bez konieczności zmian modelu PIM (por. http://www.premiumtechnology.pl/www/wwwn.nsf/-ByDocID/rs_prasa_06).

W czasach permanentnego rozwoju technologii informatycznych stworzenie biznesowego modelu rozwiązania informatycznego niezależnego od tej właśnie technologii jest wyjątkowo korzystne. Pozwala na stosunkowo płynne przejście z jednej platformy technologicznej na drugą.

Literatura

Harrington J. (2001), *Obiektowe bazy danych*, Micom, Warszawa.
Maciaszek L., Liong B.L. (2005), *Practical Software Engineering*, Perasons Addison Wesley, Edinburg.
Muller R. (Luty 2000), *Bazy danych, język UML w modelowaniu danych*, Mikom, Warszawa.
Rational Software (1997), Dokumentacja systemowa, www.rational.com.pl.
Wilk P., *Model Driven Architecture: Standard OMG dla budowy korporacyjnych systemów rozproszonych, publikacja internetowa*, http://www.premiumtechnology.pl/www/wwwn.nsf/Bitmaps/mda/$FILE/MDA_artykul.pdf
Yourdon E., (1996), *Współczesna analiza strukturalna*, Wydawnictwa Naukowo-Techniczne, Warszawa.
Yourdon E., Argila C. (2000), *Analiza obiektowa i projektowanie*, Wydawnictwo Naukowo-Techniczne, Warszawa.

THE USE OF MDA-SOLUTIONS IN DATA BASE MODELLING

Summary

This article presents issues of a standard of complex programming solutions MDA (Model Driven Architecture), described by OMG (Object Management Group). The aim of this paper is also to characterize UML language in data base modelling.