Diagramy klas

Model dziedziny i projekt aplikacji

Trudno dziś spotkać osoby zajmujące się tworzeniem systemów informatycznych, które nie słyszały o języku UML. Większość analityków, projektantów i programistów deklaruje w swoich CV znajomość tego języka. Dlatego rozpoczynamy cykl artykułów poświęconych modelowaniu systemów informatycznych w języku UML.

Dowiesz się:
- Jak efektywnie tworzyć diagramy klas w projektach informatycznych;
- Czym się różni model dziedziny od projektowego diagramu klas..

Powinieneś wiedzieć:
- Co to jest język UML i do czego służy;
- Jak przebiega projekt informatyczny i jakie są jego etapy;
- Znać notację diagramów klas języka UML.

Poziom trudności

Na rynku dostępnych jest wiele książek opisujących język UML. Łatwo jest więc poznać samą notację graficzną, dużo trudniej zaś elegancko ją wykorzystać do stworzenia poprawnego modelu. Poprawnego, a więc jak najdokładniej odzwierciedlającego modelowaną rzeczywistość biznesową. Dlatego w naszym cyklu – zamiast skupiać się na nauce notacji UML – będziemy pisać o tym, czego w książkach nie ma: jak poprawnie modelować rzeczywistość i radzić sobie w typowych sytuacjach projektowych.

UML z lotu ptaka

Specyfikacja UML nie dzieli diagramów na ważniejsze i mniej ważne. Jednak z praktycznego punktu widzenia, spośród 13 rodzajów diagramów dostępnych w języku UML (Rysunek 1), w powszechnym użyciu jest zaledwie pięć:

- diagramy czynności (ang. *activity diagrams*), służące głównie do modelowania procesów biznesowych,
- diagramy przypadków użycia (ang. *use case diagrams*), przy pomocy których projektujemy funkcjonalność systemu,
- diagramy klas (ang. *class diagrams*), definiujące obiektową strukturę systemu,
- diagramy stanów (ang. *state machine diagrams*), ukazujące cykl życia wybranych obiektów zdefinjowanych na diagramach klas,
- diagramy sekwencji (ang. *sequence diagrams*), pozwalające na pokazanie wybramych scenariuszy wymiany komunikatów pomiędzy obiektami w systemie.

Diagramy klas

Spośród tych pięciu najczęściej używanych rodzajów diagramów, jeden ma znaczenie szczególne. Diagramy klas – bo o nich mowa – są centralnym punktem analizy systemowej i projektu technicznego systemu. W nich jest zakodowane najwięcej informacji o modelowanej rzeczywistości biznesowej oraz o budowie przyszłego systemu. Nie umniejsza to roli innych diagramów, na przykład modelu procesów biznesowych, tworzonego przez analityków biznesowych w postaci diagramów czynności, czy modelu przypadków użycia. Jednak to właśnie tworząc diagramy klas podejmujemy kluczowe decyzje o kształcie systemu i sposobie, w jaki będzie on odwzorowywał rzeczywistość biznesową i wspierał swoich użytkowników. Diagramy klas są szczególnie ważne, jeśli przyszły system będzie tworzony w obiektowym języku programowania (np. w Javie), ponieważ przekładają się bezpośrednio na strukturę i kod systemu.

Diagramy klas powinny więc być wspólną płaszczyzną porozumienia dla co najmniej trzech udziałowców projektu informatycznego:

- klienta, który jest (a przynajmniej powinien być) ekspertem w swojej dziedzinie biznesowej i definiuje (a przynajmniej powinien) wymagania na system,

Objaśnienia

Artefakt (w inżynierii oprogramowania) – każdy produkt procesu produkcji oprogramowania. Może nim być: dokument, diagram, plik konfiguracyjny czy kod programu.

Z pamiętnika analityka

W projekcie dużego systemu dla instytucji ubezpieczeniowej, w którym pełniłem rolę głównego analityka, postanowiliśmy spróbować ciekawej, choć dość prostej metody współpracy z klientem – wspólnego czytania diagramów klas. W końcowej fazie prac analitycznych, gdy model dziedziny był już prawie gotowy, lecz jeszcze przed przekazaniem analizy do akceptacji, organizowaliśmy spotkania, podczas których model dziedziny prezentowaliśmy uczestnikom na ekranie. Jego omawianie nie było jednak głównym tematem spotkania. Rozmawialiśmy o tym, w jaki sposób proces biznesowy będzie wspierany przez system i dyskutowaliśmy nad funkcjonalnością systemu. Podczas rozmowy, np. o rejestrowaniu szkody ubezpieczeniowej, pokazywałem odpowiedni fragment diagramu klas i czytałem jego zawartość, np. *tu jest narysowane, że w zdarzeniu może uczestniczyć wiele pojazdów, ale musi być co najmniej jeden; nie można zarejestrować szkody bez żadnego pojazdu*, pokazując jednocześnie odpowiednie klasy, relacje i liczności. Przedstawiciele klienta weryfikowali te stwierdzenia, potwierdzając ich poprawność lub opowiadając o sytuacjach, gdy przyjęty model jest niepoprawny. Po kilku takich próbach przedstawiciele klienta na tyle przyswoili sobie podstawy notacji *UML*, że zaczęli sami odczytywać zawartość diagramu i upewniać się, że dobrze ją rozumieją. Mimo że ani my, ani nikt inny wcześniej nie uczył ich notacji *UML*, dość szybko zaangażowali się w weryfikowanie modelu.

- analityków systemowych, którzy te diagramy tworzą,
- projektantów i programistów, którzy na podstawie tych diagramów tworzą system.

Jednak czy da się stworzyć diagram klas, który będzie na tyle szczegółowy, że na jego podstawie programiści stworzą kod systemu, a jednocześnie zrozumiały dla klienta, który nie ma przygotowania technicznego ani pojęcia o programowaniu i zapewne nie chce znać szczegółów implementacji systemu? Jednego takiego diagramu zapewne stworzyć się nie da. Można natomiast utworzyć dwa odrębne diagramy (lub zestawy diagramów): model dziedziny biznesowej, przeznaczony m. in. dla klienta, oraz diagram projektowy, będący podstawą implementacji.

Model dziedziny

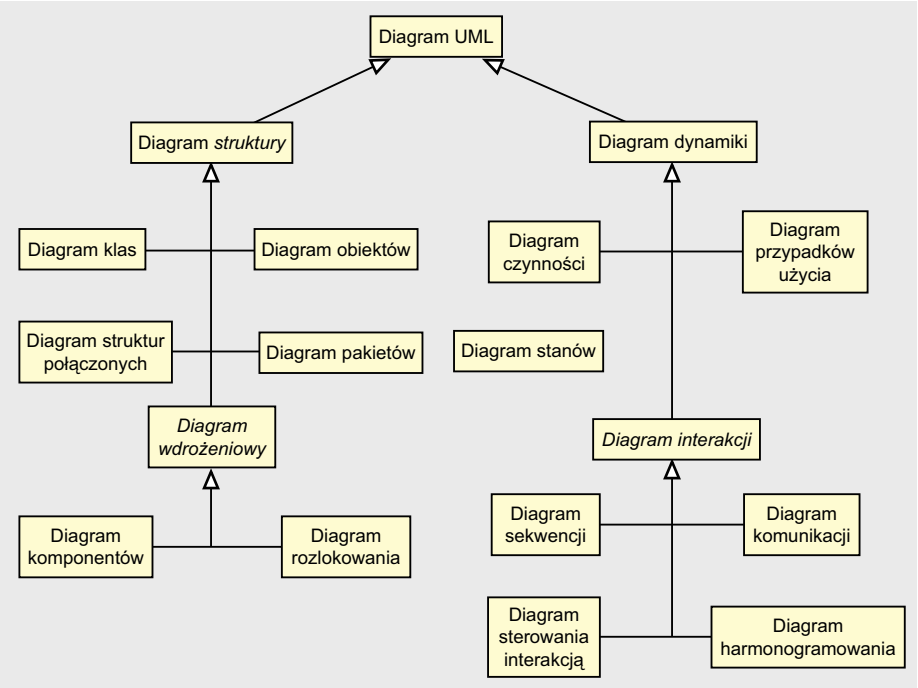
Model dziedziny to diagram klas, definiujący klasy należące do dziedziny biznesowej, którą będzie wspierał przyszły system. Są to więc obiekty należące do języka, jakim posługuje się nasz klient –przyszły użytkownik systemu. Oczywiście sam klient takiego diagramu raczej nie stworzy – jest to zadanie analityka systemowego, który powinien go narysować na podstawie artefaktów stworzonych w fazie analizy biznesowej (np. modelu procesów biznesowych, listy wymagań biznesowych i modelu przypadków użycia), współpracując przy tym blisko z klientem.

Na Rysunku 2. jest pokazany model dziedziny dla prostego programu pocztowego. W programie tym możemy tworzyć listy przeznaczone do wysłania oraz odbierać listy przesyłane do nas. Każdy taki list posiada odbiorców, z których każdy należy do grupy Do, CC lub BCC. List może też mieć załączniki. Tworząc nowy list, możemy wpisać jego odbiorców *ręcznie* lub wybrać ich z listy kontaktów. Odbiorcę wybranego z listy kontaktów można poznać po tym, że jest powiązany relacją z tym kontaktem. Każdy odbiorca ma podany dokładnie jeden adres e-mail, natomiast kontakt zapisany na liście kontaktów może mieć wiele adresów. Jeden z nich jest wyróżniony jako adres główny (zapisany w atrybucie `adresGłówny`).

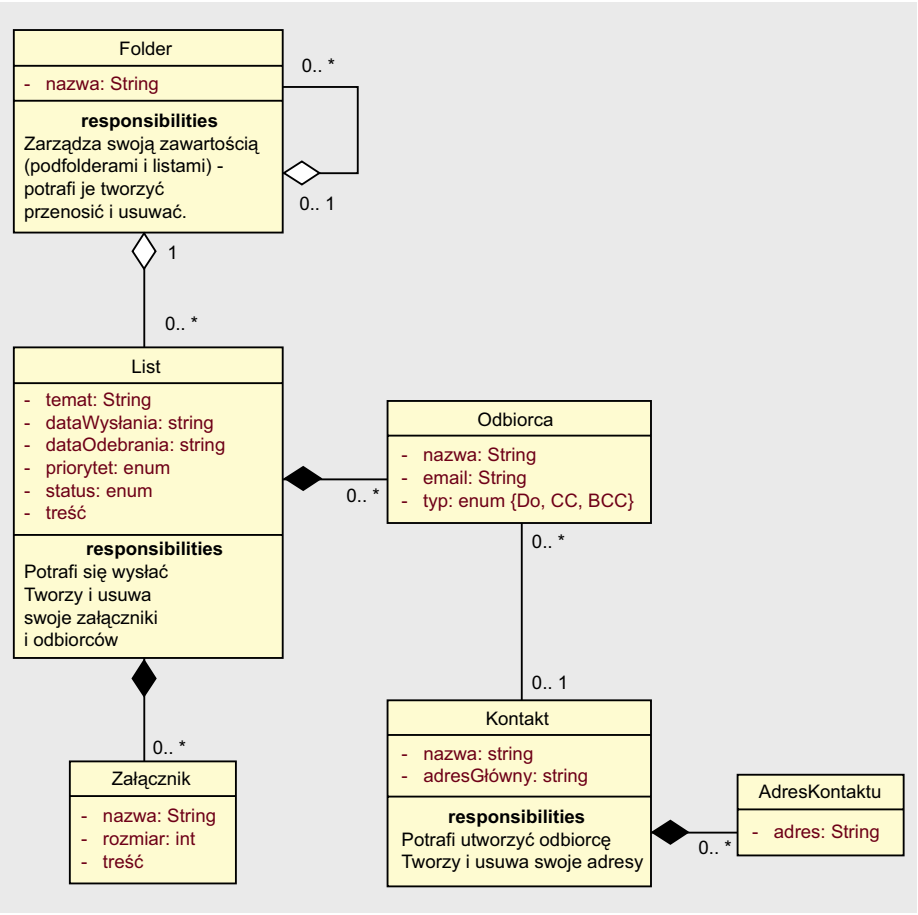
Nasz program pocztowy pozwala dodatkowo zarządzać listą kontaktów oraz folderami, które tworzą strukturę hierarchiczną (hierarchię zapewnia nam relacja klasy Folder z samą sobą).

Zwróćmy uwagę, że na modelu dziedziny nie umieszczamy klas implementujących logikę aplikacji, jej interfejs użytkownika czy warstwę dostępu do danych. Nie warto też na modelu dziedziny umieszczać metod. Dzięki temu diagram jest prosty i powinien być zrozumiały także dla klienta nie mającego przygotowania technicznego i nie zainteresowanego szczegółami implementacji systemu. Model dziedziny jest zresztą zwykle tworzony na etapie analizy, kiedy nie znamy jeszcze budowy aplikacji, więc trudno byłoby nam te metody precyzyjnie zdefiniować. Na metody przyjdzie czas w fazie projektowania. Póki co, na etapie analizy, możemy co najwyżej umieścić w klasach dodatkową sekcję Responsibilities i opisać w niej słowami tzw. odpowiedzialności klas, czyli to, co każdy z obiektów danej klasy powinien umieć zrobić. Dzięki nim w fazie projektowania łatwiej będzie stworzyć metody poszczególnych klas, nie zapominając o żadnym istotnym elemencie logiki biznesowej.

Można w tym momencie zadać pytanie, czy w ogóle klient powinien weryfikować diagramy klas? Czy nie powinien się ograniczyć do zatwierdzenia wymagań biznesowych i przypadków użycia, opisujących funkcjonalność systemu, pozostawiając diagramy klas fachowcom informatykom? Na pewno nie ma sensu, aby klient wtrącał się do



Rysunek 1. *Diagramy języka UML*



Rysunek 2. *Model dziedziny programu pocztowego*

szczegółowych diagramów projektowych, o których powiemy dalej. Jednak model dziedziny zawiera informacje o charakterze czysto biznesowym. Bez ich weryfikacji przez specjalistę z danej dziedziny ryzykujemy popełnienie fundamentalnych błędów merytorycznych. Przecież nikt inny nie powie nam, czy list musi mieć dokładnie jednego odbiorcę, czy też może ich mieć wielu. Jedynie klient jest w stanie zdecydować, czy może istnieć list, który wcale nie ma odbiorców. Decyzje te są zbyt poważne, by podejmował je *na wyczucie* nie znający dziedziny biznesowej projektant czy programista. Co jednak zrobić w sytuacji, gdy klient nie zna UML-a? Najlepiej wraz z nim czytać diagramy i objaśniać ich sens.

Projektowe diagramy klas

Po zakończeniu fazy analizy przychodzi czas na projektowanie systemu. Wtedy dopiero powinniśmy – korzystając z modelu dziedziny – utworzyć projektowe diagramy klas, będące podstawą implementacji, a więc zawierające wszystkie klasy wykorzystane w implementacji wraz z ich atrybutami i metodami. Będą wśród nich oczywiście także klasy z modelu dziedziny – w końcu one też należą do logiki tworzonego systemu. Najlepiej byłoby więc stworzyć diagramy projektowe rozbudowując model dziedziny poprzez dodanie do niego brakujących klas, atrybutów oraz metod. Model dziedziny będzie wtedy elegancko *zanurzony* w diagramach projektowych. Na Rysunku 3. jest pokazany utworzony w ten sposób diagram projektowy naszego programu pocztowego. Jest to de facto model dziedziny uzupełniony o kilka dodatkowych klas implementujących logikę biznesową. Dla uproszczenia na diagramie nie pokazujemy metod służących do operowania na atrybutach (zwanych żargonowo setterami i getterami) oraz klas implementujących interfejs użytkownika oraz warstwę dostępu do danych.

Nie zawsze jednak da się stworzyć diagram projektowy tak elegancko. Platformy do budowania aplikacji (*ang. application frameworks*) czy narzędzia MDA, pozwalające na generowanie kodu na podstawie modelu, często wymuszają stosowanie na diagramach klas pewnych konwencji i konstrukcji, które są z analitycznego punktu widzenia sztuczne. W takich sytuacjach diagram projektowy lepiej jest narysować *od zera*, korzystając z modelu dziedziny jedynie jako źródła wiedzy.

Można się zastanawiać, czy w ogóle jest sens tworzyć model dziedziny – diagram z punktu widzenia implementacji niekompletny. Czy nie jest to strata czasu? Czy nie lepiej od razu narysować szczegółowy diagram projektowy? Jeśli samodzielnie lub z kolegą tworzymy w miarę prosty program, to rzeczywiście można od razu stworzyć docelowy projektowy diagram klas. Jednak w dużych, komercyjnych projektach nie bez powodu dzielimy prace na kilka faz, takich jak: analiza biznesowa, analiza systemowa, projektowanie, implementacja, testy i wdrożenie. Model dziedziny jest elementem analizy systemowej, ukazującym biznesowe struktury danych i zależności pomiędzy nimi. W dużych projektach może on obejmować kilkaset klas. Decydując się na jego stworzenie, nie pozwalamy, aby istotne decyzje merytoryczne były podejmowane ad-hoc w trakcie implementacji.

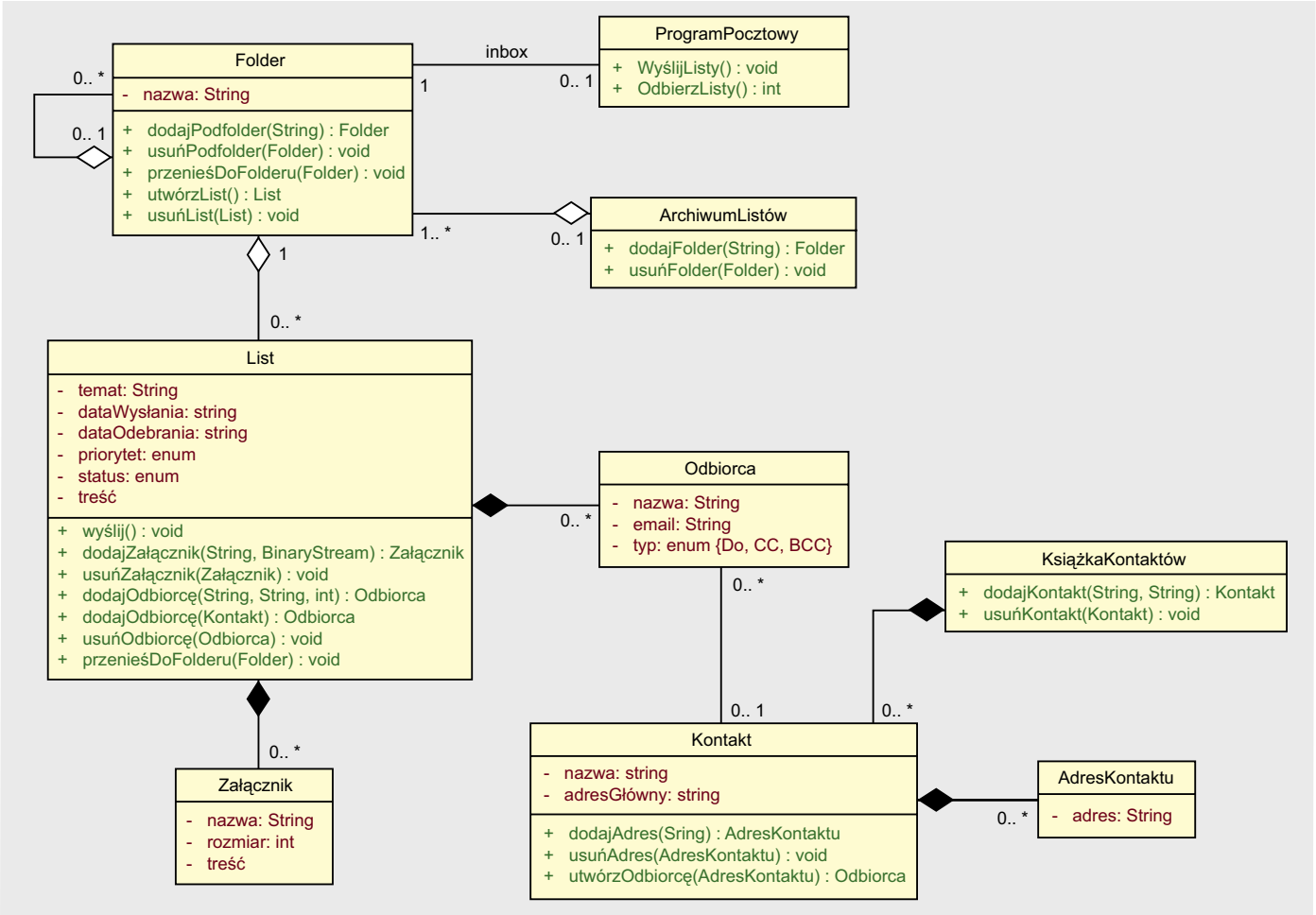
Za miesiąc

Za miesiąc zobaczymy, jak na diagramach klas poprawnie modelować role pełnione przez obiekty.

SZYMON ZIOŁO

Główny konsultant w firmie RedPill, oferującej specjalistyczne szkolenia informatyczne oraz doradztwo w zakresie zarządzania projektami, modelowania procesów biznesowych oraz analizy i projektowania systemów informatycznych. Analityk biznesowy i systemowy, specjalizujący się w analizie obiektowej, modelowaniu w języku UML oraz w XML-u i technologiach pokrewnych.

Kontakt z autorem: sziolo@redpill.com.pl



Rysunek 3. *Diagram projektowy programu pocztowego*