helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

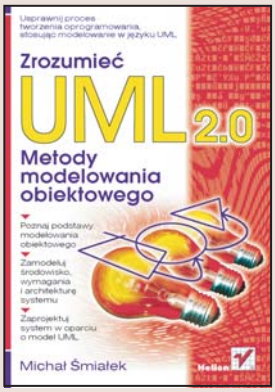
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Zrozumieć UML 2.0. Metody modelowania obiektowego



Autor: Michał śmiałek
ISBN: 83-7361-918-6
Format: B5, stron: 296

Usprawnij proces tworzenia oprogramowania, stosując modelowanie w języku UML

- Poznaj podstawy modelowania obiektowego
- Zamodeluj środowisko, wymagania i architekturę systemu
- Zaprojektuj system w oparciu o model UML

Tworzenie złożonego systemu informatycznego wymaga przygotowania projektu. Należy określić w nim środowisko działania systemu, wymagania użytkowników, procesy realizowane przez system i jego elementy składowe. Opis słowny, przydatny w trakcie zbierania założeń funkcjonalnych, może okazać się zbyt skomplikowany i niejednoznaczny na pozostałych etapach opisywania powstającego systemu. Niezbędny jest taki sposób opisu, który byłby jednakowo interpretowany i zrozumiały dla wszystkich członków zespołu projektowego. W tym celu opracowano język UML – notację umożliwiającą zamodelowanie systemu w sposób graficzny, w postaci diagramów. Modele zapisane w języku UML są jednakowo interpretowane przez wszystkie osoby zaangażowane w dany projekt. Są też niezwykle uniwersalne. Można je stosować we wszystkich fazach projektowania i budowy oprogramowania.

Książka „Zrozumieć UML 2.0. Metody modelowania obiektowego" to podręcznik modelowania systemów informatycznych z wykorzystaniem notacji UML 2.0. Przedstawia podstawy modelowania obiektowego i najważniejsze pojęcia związane z obiektowością. Opisuje sposoby modelowania otoczenia systemu, jego zakresu funkcjonalnego oraz struktury. W książce opisano również proces przejścia z modelu do kodu źródłowego systemu oraz metodyki projektowe oparte na języku UML. Każdy, kto bierze udział w procesie wytwarzania oprogramowania, znajdzie w tej książce przydatne dla siebie informacje.

- Zasady modelowania obiektowego
- Formułowanie i realizacja wymagań
- Modelowanie otoczenia systemu oraz jego funkcjonalności
- Projektowanie architektury systemu
- Realizacja systemu w oparciu o projekt
- Metodyki wytwarzania oprogramowania

Przekonaj się, jak bardzo UML może ułatwić proces tworzenia oprogramowania

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 1. Wstęp 5
1.1. Czym jest ta książka? 5
1.2. Dla kogo jest ta książka? 6
1.3. Dlaczego modelowanie? 7
1.4. Dlaczego język UML? 8
1.5. Co dalej w książce? 9
1.6. Oznaczenia typograficzne 12
1.7. Podziękowania 13

Rozdział 2. Wprowadzenie do modelowania 15
2.1. Trudna sztuka modelowania 15
2.2. Złożoność oprogramowania. Jak ją pokonać? 17
2.3. Metodyka wytwarzania oprogramowania 20

Rozdział 3. Podstawy modelowania obiektowego 25
3.1. Jak modelować świat obiektowo? 25
3.2. Obiekty 27
3.3. Klasy obiektów 32
3.4. Modelowanie struktury 37
3.5. Modelowanie dynamiki 43
3.6. Które modele muszę dobrze poznać? 50

Rozdział 4. Od opisu środowiska do kodu 53
4.1. Jak zapewnić zgodność kodu ze środowiskiem? 53
4.2. Ścieżka od opisu środowiska do kodu 57
4.3. Tworzenie systemów sterowane modelami 60
4.4. Formułowanie wymagań 63
4.5. Realizacja wymagań 65
4.6. Modele, narzędzia, jakość 67

Rozdział 5. Modelowanie środowiska 71
5.1. Jak opisać środowisko? 71
5.2. Struktura — aktorzy, jednostki organizacyjne i pojęcia 77
5.3. Dynamika — procesy, czynności i stany 87
5.4. Spójność modeli środowiska 100
5.5. Przykładowe problemy 102

Rozdział 6. Modelowanie wymagań ... 105
6.1. Jak określić zakres funkcjonalny systemu oprogramowania? ... 105
6.2. Struktura — aktorzy, klasy ... 108
6.3. Dynamika — przypadki użycia, scenariusze, czynności, stany ... 124
6.4. Spójność modelu wymagań i zgodność ze środowiskiem ... 145
6.5. Przykładowe problemy ... 148

Rozdział 7. Tworzenie architektury ... 151
7.1. Co to jest i z czego się składa architektura systemu oprogramowania? ... 151
7.2. Struktura — komponenty, interfejsy, klasy, węzły ... 155
7.3. Dynamika — interakcje, czynności ... 177
7.4. Spójność architektury i zgodność z wymaganiami ... 189
7.5. Przykładowe problemy ... 193

Rozdział 8. Projektowanie i realizacja systemu ... 195
8.1. Jak stworzyć szczegółowy projekt systemu? ... 195
8.2. Struktura — interfejsy, klasy ... 197
8.3. Dynamika — interakcje, czynności ... 205
8.4. Spójność projektu i zgodność z architekturą ... 219
8.5. Przykładowe problemy ... 222

Rozdział 9. Modelowanie w procesie wytwórczym ... 225
9.1. Nowoczesne metodyki wytwarzania oprogramowania ... 225
9.2. Proces techniczny oparty na modelach w języku UML ... 232
9.3. Podsumowanie: czy modelowanie to „srebrna kula”? ... 239

Dodatek A Podsumowanie składni języka UML 2.0 ... 241

Dodatek B Organizacja modeli w narzędziu CASE ... 259

Dodatek C Krótki słownik metod obiektowych i języka UML ... 267

Literatura ... 287

Skorowidz ... 295

Rozdział 3.

Podstawy modelowania obiektowego

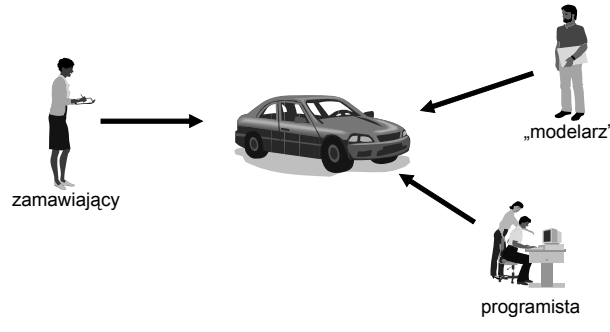
3.1. Jak modelować świat obiektowo?

Z poprzedniego rozdziału wiemy, że system oprogramowania powinien być jak najwierniejszym modelem otaczającego świata. Powstaje zatem pytanie, w jaki sposób skonstruować ten model, aby spełniał to podstawowe wymaganie wierności? Mamy bowiem do czynienia z dwoma różnymi punktami widzenia na oprogramowanie. Z jednej strony stoją zamawiający, którzy mają pewne wymagania w stosunku do systemu. Z drugiej strony mamy programistów, którzy wiedzą, jak konstruować systemy. Zamawiający znają dziedzinę problemu i potrafią o niej opowiadać za pomocą specjalistycznego słownictwa. Nie są to zazwyczaj osoby skłonne do czytania, a tym bardziej tworzenia, technicznych opisów systemu. Zamawiający chcą opisać system w sposób jak najprostszy i jak najbardziej zrozumiały w kontekście swoich codziennych obowiązków. Wynika to oczywiście z tego, że system powinien im pomagać w pracy, którą wykonują. Z drugiej strony programiści tworzą system z bardzo precyzyjnych konstrukcji odpowiedniego języka programowania. Są oni najczęściej dokładnie zorientowani w szczegółach platformy sprzętowej lub programowej, czy też w niuansach określonej biblioteki kodu. Nie są natomiast zazwyczaj ekspertami w dziedzinie, dla której tworzą system. Ich słownictwo jest zasadniczo różne od słownictwa zamawiających. Wymagają zatem bardzo dokładnego opisu sposobu konstrukcji systemu.

Żeby było jeszcze trudniej, zarówno zamawiający, jak i programiści muszą panować nad systemem, który zwykle składa się z tysięcy różnych wymagań, dziesiątków modułów czy setek tysięcy wierszy kodu. Podkreślaliśmy to już w poprzednim rozdziale. Jak zatem pogodzić sprzeczne spojrzenia zamawiających i programistów? Jak zapanować nad złożonością problemu? Czy istnieje jakaś możliwość porozumienia? Czytelnik zapewne już się domyśla, że sposobem na wzajemne zrozumienie, który będziemy chcieli zaproponować w tej książce, jest modelowanie za pomocą technik obiektowych.

Elementem, który w modelowaniu obiektowym pozwala pogodzić język użytkownika z językiem programisty oraz pokonać problem złożoności systemu, jest obiekt. Obiekty znajdujące się w środowisku ustalają wspólny język w zespole odpowiedzialnym za powstanie systemu oprogramowania (rysunek 3.1). Z jednej strony obiekty odpowiadają pojęciom z modelowanej dziedziny problemu. Z drugiej strony są podstawą implementacji realizowanego systemu. Jest to możliwe dzięki istnieniu obiektowych języków programowania. Obiekty umożliwiają realizację zasady abstrakcji, gdyż mogą reprezentować skomplikowane struktury (składające się z wielu elementów, które znowu składają się z wielu elementów itd.). Dzięki temu obiekty na danym poziomie abstrakcji ukrywają informacje nieistotne dla czytelnika modelu. Obiekty są zatem pewnego rodzaju czarnymi skrzynkami (rysunek 3.7). Dopiero po otwarciu takiej czarnej skrzynki i wejściu w głąb odkrywamy dalsze szczegóły.

Rysunek 3.1.
Obiekty — wspólny język dla różnych ról projektowych



Na bazie obiektów powstają obiektowe →**modele** (ang. *model*), czyli — tak jak już mówiliśmy w poprzednim rozdziale — kopie rzeczywistych systemów. →**Modelowanie obiektowe** (ang. *object modeling*) polega zatem na:

- znajdowaniu obiektów w otoczeniu,
- opisywaniu struktury i dynamiki działania obiektów,
- klasyfikacji obiektów,
- opisywaniu struktury powiązań klas obiektów oraz
- opisywaniu dynamiki współpracy obiektów podczas funkcjonowania systemu.

Możemy też powiedzieć, że modelowanie obiektowe polega na rysowaniu diagramów opisujących strukturę i dynamikę systemu składającego się z obiektów. Diagramy ukazują system na różnym poziomie abstrakcji.

Modele obiektowe będziemy tworzyć za pomocą jednolitej notacji. Będzie nią graficzny język UML. W rozdziale przedstawimy podstawowe pojęcia modelowania obiektowego i ich notację w języku UML: obiekty, klasy, diagramy opisu struktury i diagramy opisu dynamiki. Będzie to podstawowy język porozumiewania się. Zostanie on wykorzystany i rozbudowany w następnych rozdziałach do ustanowienia ścieżki od wymagań do kodu.

3.2. Obiekty

Jak już powiedzieliśmy, obiekty występują w środowisku, ale również mogą być elementami systemu oprogramowania. Obiekty mogą zatem być przedmiotami (urządzeniami technicznymi, budynkami, tworami przyrody), osobami, ale również różnego rodzaju jednostkami organizacyjnymi, zdarzeniami lub dokumentami. W systemie oprogramowania, oprócz obiektów odpowiadających elementom środowiska, mogą występować np. ekrany, interfejsy, bazy danych, zarządcy aplikacji czy komunikacji. Zwróćmy uwagę, że obiekty związane z opisem systemu są zależne od wykorzystywanej technologii, a nie od modelowanego środowiska. Takie obiekty techniczne pojawiają się w momencie, kiedy zaczynamy projektować system wraz z jego architekturą.

Zastanówmy się teraz nad tym, w jaki sposób należy opisywać obiekty. →**Obiekt** (ang. *object*) w rozumieniu modelowania obiektowego może być opisany za pomocą trzech elementów: tożsamości, stanu i zachowania. Każdy obiekt ma zatem indywidualną tożsamość odróżniającą go od innych obiektów. Obiekty zawierają również elementarne składniki o zmieniających się wartościach, które określają ich stan. Potrafią też zachowywać się w odpowiedni sposób w różnych sytuacjach — w odpowiedzi na komunikaty wykonują określone usługi na rzecz innych obiektów.

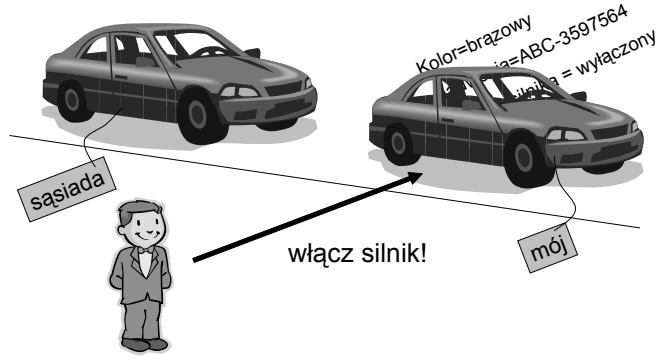
UML 2.0

W języku UML →obiekt jest reprezentowany za pomocą prostokąta, który zawiera podkreśloną nazwę obiektu (rysunek 3.2).

Rysunek 3.2.
UML: Podstawowa notacja obiektu

Zanim przedstawimy bliżej wymienione powyżej elementy opisu obiektów, rozważmy krótki przykład. Ponieważ najłatwiej jest chyba operować na przykładzie znanych nam urządzeń technicznych, weźmiemy pod uwagę nasz samochód[1] (rysunek 3.3). Ma on niewątpliwie indywidualną tożsamość, różną od tożsamości samochodu naszego sąsiada. Nie ma tutaj znaczenia fakt, że obydwa samochody wyglądają na pierwszy rzut oka identycznie. Łatwo jest rozróżnić tożsamość samochodów, jeżeli widzimy je obok siebie. Ten samochód jest nasz, a tamten — naszego sąsiada. Są to przecież różne przedmioty, nawet jeśli wyglądają tak samo. Nasz samochód jest oczywiście w określonym stanie: jest koloru brązowego, ma silnik pojemności 1200 cm^3, silnik jest wyłączony, hamulec jest zaciągnięty, numer nadwozia to ABC-3597564 itd. Samochód może się też w określony sposób zachowywać — może uruchomić silnik, może też skręcić w lewo lub w prawo albo pojechać do tyłu. Musimy tylko go o to „poprosić” — przekręcić kluczyk w stacyjce, skręcić kierownicą czy wrzucić wsteczny bieg. Na bazie tego przykładu spróbujmy teraz formalnie zdefiniować poszczególne elementy opisu obiektu.

[1] Zwróćmy uwagę, że bierzemy pod uwagę „nasz samochód”, a nie ogólnie — „jakikolwiek samochód”.

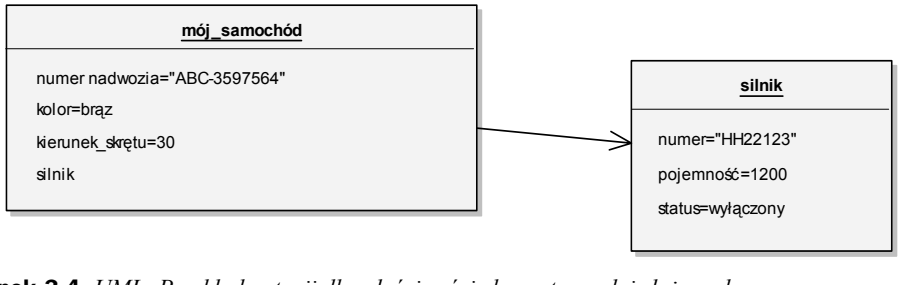


Rysunek 3.3. *Tożsamość, stan i zachowanie samochodu*

→**Stan** (ang. *state*) obiektu to zbiór wartości (cech charakterystycznych) wszystkich jego właściwości. Wartością właściwości →obiektu może być np. jakaś liczba, napis lub element innego typu. Każdy obiekt ma przypisany zbiór właściwości, które go charakteryzują. Są one nierozerwalnie związane z danym obiektem — są jego składowymi. Zwróćmy też uwagę, że właściwości są tak naprawdę obiektami, które są w całości kontrolowane przez obiekt główny. W czasie swojego życia obiekt ma zawsze ten sam zestaw właściwości. Jednakże właściwości obiektu mogą w trakcie życia obiektu przyjmować różne wartości. Te różne wartości wyznaczają stan obiektu w danym momencie.

UML 2.0

→Stan →obiektu określamy, podając wartości jego →**właściwości** (ang. *property*), które umieszczamy w →**przegródce** (ang. *compartment*) poniżej nazwy obiektu (rysunek 3.4). Po nazwie właściwości umieszczamy znak „=”, a następnie wartość właściwości. Oczywiście taka notacja jest możliwa dla właściwości, które przybierają wartości typów elementarnych (wartości liczbowe, napisy itp.). W przypadku gdy właściwość obiektu jest obiektem złożonym, wewnątrz przegródki właściwości umieszczamy zazwyczaj jedynie nazwę obiektu składowego. Stan właściwości składowej możemy pokazać, umieszczając obok obiekt odpowiadający tej właściwości i określając wartości właściwości tego obiektu składowego. Dodatkowo łączymy obiekt główny i obiekt składowy →**łącznikiem** (ang. *link*) wskazującym na obiekt podrzędny. Na rysunku 3.4 obiektem głównym jest `mój_samochód`. Jak widzimy, jedną z właściwości tego obiektu jest `silnik`. Aktualny stan silnika określa obiekt o tej samej nazwie jak odpowiednia właściwość `mojego_samochodu`.

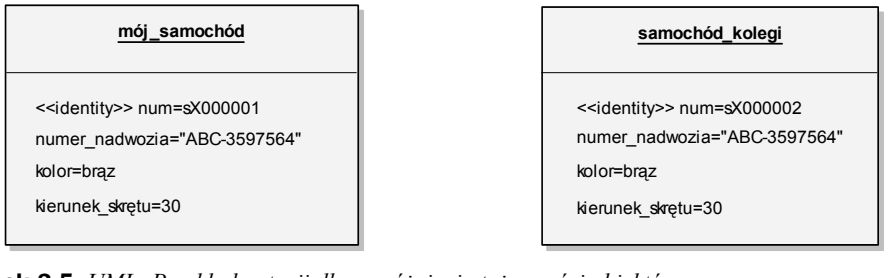


Rysunek 3.4. *UML: Przykład notacji dla właściwości elementarnych i złożonych*

→**Tożsamość** (ang. *identity*) obiektu wyróżnia →obiekt wśród innych obiektów jako osobną jednostkę. Można ją określić jako wyróżnioną cechę obiektu, która pozostaje niezmienna przez cały czas życia tego obiektu. Co więcej, wartość tej cechy powinna być unikalna wśród wszystkich obiektów, które otaczają nasz obiekt. W najprostszym przypadku cechą obiektu, która stanowi o jego tożsamości, jest umiejscowienie obiektu (np. w pamięci systemu). Może się jednak okazać, że obiekt zmienia swoje położenie albo znajduje się w zbiorze nieuporządkowanych obiektów o swobodnym dostępie (np. w bazie danych). Wtedy tożsamość obiektu powinniśmy określić poprzez dodanie wyróżnionej →właściwości (np. nadając jej wyróżnik «identity»). Należy tu podkreślić, że taka dodatkowa właściwość nie może stanowić elementu stanu obiektu. Jako taka powinna zatem przybierać wartości całkowicie abstrakcyjne — niezależne od →**dziedziny problemu** (*ang. problem domain*), czyli od otaczającej rzeczywistości. Jeżeli tożsamość nie byłaby cechą abstrakcyjną obiektu, to mogłaby ulec zmianie w trakcie życia obiektu — w miarę zmian →dziedziny problemu. Warto też podkreślić, że oczywiście możliwe jest rozróżnienie tożsamości obiektów za pomocą ich →stanu (np. stwierdzając, jaki kolor ma samochód). Może się jednak zdarzyć, że w systemie wystąpią dwa różne obiekty o identycznym stanie. Wtedy jedyną możliwością rozróżnienia obiektów będzie ich abstrakcyjna tożsamość.

UML 2.0

Najbardziej oczywiste rozróżnienie →tożsamości →obiektów to po prostu umieszczenie na diagramie różnych obiektów, które w szczególności posiadają różne nazwy („mój samochód" i „samochód kolegi" na rysunku 3.5). Wtedy rozróżnienie i dostęp do →stanu obiektów następuje poprzez odwołanie się do ich unikalnego położenia. Problem pojawia się wtedy, gdy obiekty chcemy zapisać np. w bazie danych. W takiej sytuacji musimy do każdego obiektu „doczepić" dodatkową „metkę" odróżniającą go od innych. Na rysunku 3.5 taką metką jest dodatkowa właściwość o nazwie „num". Aby zaznaczyć, że ta właściwość nie jest elementem stanu obiektu, nadaliśmy jej wyróżnik «identity»[2]. Zwróćmy też uwagę, że obiekty na rysunku 3.5 są w identycznym stanie. Wszystkie wartości ich właściwości są takie same. Dotyczy to nawet numerów nadwozi, które są (zapewne przez pomyłkę) identycznymi napisami. Gdybyśmy zapisali te dwa obiekty w bazie danych bez wyróżnika, to niemożliwe byłoby ich rozróżnienie.



Rysunek 3.5. *UML: Przykład notacji dla rozróżnienia tożsamości obiektów*

[2] Takie wyróżniki umieszczone w podwójnych nawiasach ukośnych nazywamy w języku UML stereotypami. Zostaną one omówione dokładniej w dalszych rozdziałach książki.

Na koniec zauważmy, że wyróżnik «identity» jest wartością, która powinna być generowana automatycznie, tak aby zapewnić jego unikalność (zwróćmy uwagę na wyróżniki obiektów na rysunku 3.5 będące kolejnymi unikalnymi wartościami w ciągu).

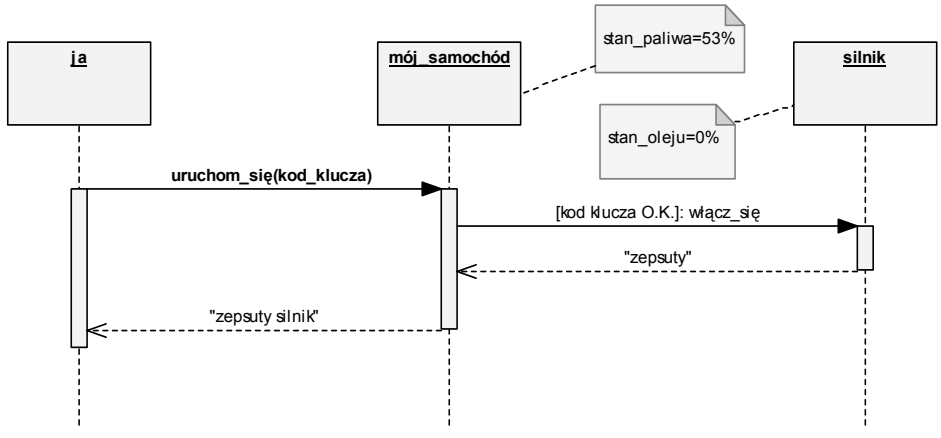
→**Zachowanie** (ang. *behavior*) obiektu to zbiór usług, które →obiekt potrafi wykonywać na rzecz innych obiektów. Zachowanie ustanawia bardzo istotny element dynamiki modelu (tzn. sposobu działania systemu). W ramach tej dynamiki obiekty mogą prosić inne obiekty o wykonanie odpowiednich usług. Obiekt reaguje na taką prośbę, jeżeli usługa jest w zbiorze obsługiwanych przez niego usług. Prośby obiektów o wykonanie usług będziemy nazywali →**komunikatami** (ang. *message*). W ramach wykonania usługi obiekt przeprowadza odpowiednie dla usługi przetwarzanie danych. Jego efektem może być zmiana →stanu obiektu albo dostarczenie drugiemu obiektowi odpowiedniego rezultatu przetwarzania. Efekt wykonania usługi zależy od trzech rzeczy: a) stanu obiektu, b) parametrów komunikatu, c) stanu innych obiektów, z których usług korzysta obiekt podczas przetwarzania. Zależność od stanu obiektu jest oczywista. Inaczej się będzie zachowywał np. samochód ze stanem paliwa równym 100%, a inaczej przy stanie paliwa równym zero. →**Parametry** (ang. *parameter*) komunikatu to lista wartości obiektów, które pozwalają na sterowanie zachowaniem usługi. Usługa może się zatem zachowywać różnie w zależności od wartości parametrów. Na przykład parametrem usługi „skręć" powinien być kąt skrętu. W trakcie wykonywania usługi obiekt może również poprosić inne obiekty o pomoc. Wtedy wysyła do nich odpowiednie komunikaty. Dlatego też przetwarzanie usługi może być zależne od stanu innych obiektów.

Po zakończeniu wykonania usługi często następuje komunikat zwrotny — od obiektu, który wykonał usługę („wykonawcy"), do obiektu proszącego o wykonanie usługi („klienta"). Przekazywane są w nim wartości obiektów, które są rezultatem przetwarzania i na które oczekuje klient usługi.

UML 2.0

Na rysunku 3.6 widzimy opis fragmentu zachowania się →obiektu „mój samochód". Jest to →**diagram sekwencji** (ang. *sequence diagram*) pokazujący wymianę →komunikatów podczas jednego z wykonań usługi „uruchom się". Komunikaty wymieniane są między kolumnami nazywanymi →**liniami życia** (ang. *lifeline*). Każda kolumna odpowiada jednemu obiektowi. Kolejność wymienianych komunikatów jest liczona od góry do dołu diagramu. Komunikat wywołujący usługę jest zaznaczony poziomą strzałką o pełnym grocie, skierowaną od obiektu klienta do obiektu wykonawcy. Na rysunku rozpoczęcie działania usługi „uruchom się" następuje poprzez skierowanie komunikatu „uruchom się" do obiektu „mój samochód". Zauważmy, że komunikat ten ma →parametr — „kod klucza". Nazwa parametru umieszczona jest w nawiasach po nazwie komunikatu. Gdyby parametrów było więcej, moglibyśmy je umieścić — rozdzielone przecinkami — wewnątrz nawiasu.

Po uruchomieniu usługi (przesłaniu komunikatu) następuje →**wykonanie usługi** (ang. *execution occurence*). Zaznaczane jest jako wąski prostokąt umieszczony na →linii życia. Z diagramu możemy wywnioskować, że po wywołaniu usługi samochód sprawdza, czy kod klucza jest odpowiedni. Prawdopodobnie sprawdzany jest również stan paliwa w zbiorniku. W tym konkretnym przypadku

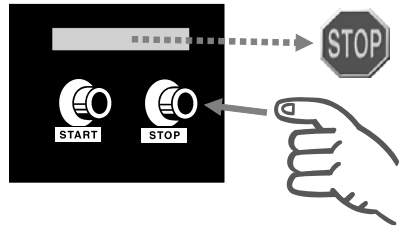


Rysunek 3.6. *UML: Przykład sekwencji komunikatów podczas realizacji usługi*

zarówno stan paliwa, jak i kod klucza były prawidłowe. W związku z tym samochód przesyła do silnika →komunikat „włącz się". Zwróćmy uwagę, że przesłanie komunikatu następuje pod pewnym →**warunkiem** (ang. *condition*). Jest on umieszczony w nawiasach prostokątnych. Możemy się domyślać, że gdyby ten warunek nie był spełniony (kod klucza nie byłby „O.K."), nastąpiłaby jakaś inna akcja (np. wysłanie innego komunikatu).

Po otrzymaniu komunikatu „włącz się" silnik zapewne próbuje się uruchomić. Z diagramu wynika, że mu się to nie udało. Wysyła zatem →komunikat zwrotny, sygnalizujący, że silnik jest zepsuty. Komunikat zwrotny zaznaczany jest przerywaną linią. Przyczynę „zepsucia" silnika wyjaśnia →**notatka** (ang. *note*) przyczepiona do obiektu — brakuje oleju. Notatki umieszczone na rysunku 3.6 zawierają w sobie opis aktualnych wartości niektórych właściwości poszczególnych obiektów (czyli fragmenty stanów tych obiektów). Po otrzymaniu komunikatu od silnika samochód wysyła komunikat zwrotny sygnalizujący zepsucie silnika. Zwróćmy uwagę, że komunikat zwrotny kończy wykonanie usługi. Wykonanie usługi zawarte jest zatem między komunikatem wywołującym usługę a komunikatem zwrotnym.

Podsumowując rozważania na temat cech →obiektów, należy stwierdzić, że bardzo istotną własnością modelowania obiektowego jest to, że obiekty stanowią pewnego rodzaju „czarne skrzynki". Obiekt pokazuje na zewnątrz tylko te swoje właściwości lub usługi, które są potrzebne innym obiektom. Inne szczegóły są ukryte. Umożliwia to realizację zasady abstrakcji, o której była mowa w rozdziale 2. Możemy sobie zatem wyobrazić obiekt jako czarną skrzynkę z przyciskami (rysunek 3.7). Naciśnięcie przycisku oznacza uruchomienie odpowiedniej usługi obiektu. Wynikiem wykonania usługi jest rezultat, który otrzymuje ten, kto nacisnął przycisk. Dla naciskającego nie jest natomiast ważne, jak usługa jest w środku wykonywana i kto jeszcze w jej wykonaniu uczestniczy. Ważne jest jednak, aby przyciski były dobrze opisane, tak aby naciskający wiedział, co dostanie po naciśnięciu przycisku.



Rysunek 3.7.
Obiekt jako czarna skrzynka z przyciskami