Część I

Podstawy języka UML 2.0

Rozdział 1.

Język UML — rozwój, struktura, pojęcia

Znaczenie obiektowości w modelowaniu systemów informatycznych

Modelowanie systemów informatycznych to nieustające i trudne wyzwanie dla twórców systemów, począwszy od pierwszych biznesowych zastosowań systemów z lat 50. Pierwsze, stosunkowo proste podejścia do **modelowania systemów** zmieniały się i ewoluowały w zawansowane teoretycznie paradygmaty tworzenia systemów informatycznych (TSI). Szczególnie dwa podejścia zdominowały analizę oraz projektowanie systemów — podejście strukturalne oraz obiektowe. Zaskakuje mnogość i różnorodność związanych z nimi metodyk i notacji. Różnorodność ta doprowadziła w latach 80. do swoistych „wojen metodologicznych", w ramach których różne grupy twórców współzawodniczyły z sobą, akcentując odrębność własnych koncepcji. Ósma dekada XX wieku to okres szczególnej dominacji podejścia strukturalnego, które zaowocowało wieloma metodykami wypracowanymi przez środowiska naukowe i biznesowe. Mimo istotnych różnic pomiędzy nimi, istniały punkty, kategorie i mechanizmy wspólne, będące podstawą dalszego rozwoju. Zaliczyć do nich należy pojęcie i składniki metodyki tworzenia systemów informatycznych, cykl życia systemu, prototypowanie, diagramy związków encji, modele relacyjnych baz danych czy też narzędzia CASE (*Computer-Aided Software Engineering*).

Naturalnie różnorodność ta wprowadzała spory wśród specjalistów z dziedziny inżynierii oprogramowania co do wartości i użyteczności oferowanych, odmiennych podejść. Również pewne obietnice podejścia strukturalnego — chociażby automatyzacja procesu tworzenia systemów informatycznych od analizy do generowania kodu — nie zostały spełnione. W celu uporządkowania wiedzy i procedur modelowania systemów informatycznych podjęto dwie znaczące inicjatywy **unifikacyjne**:

- podjętą przez IFIP (*International Federation of Information Processing*) w postaci grupy roboczej CRIS (*Comparative Review of Information Systems Methodologies*),
- podjętą przez Unię Europejską w ramach zespołu reprezentującego autorów metodyk używanych w różnych krajach członkowskich pod nazwą *EuroMethod*.

Wymienione inicjatywy unifikacji procesu projektowania i wdrażania systemów informatycznych nie zyskały aprobaty środowiska informatyków. Niemniej ogólna, zbiorowa wiedza na temat różnorodnych metodyk strukturalnych i związane z nimi narzędzia CASE, stały się w latach 80. i później bardzo przydatnym, rutynowo wykorzystywanym, klasycznym narzędziem analizy oraz projektowania systemów informatycznych.

Problemy związane z podejściem strukturalnym sprzyjały zainteresowaniu oraz rozwojowi prac naukowo-badawczych i wdrożeniowych, opierających się na innych podstawach teoretycznych. Wymienić tu należy przede wszystkim społeczne i obiektowe modelowanie systemów. Od początku lat 90. szczególnie modele obiektowe znalazły się w centrum zainteresowania twórców i użytkowników systemów. Wzrost znaczenia **obiektowości** stymulowany był przez:

- wyzwania postępu technologicznego w informatyce, polegające na coraz bardziej powszechnym użytkowaniu systemów czasu rzeczywistego i systemów wbudowanych;
- różnorodność i powszechność aplikacji internetowych w gospodarce opartej na wiedzy, czyli w takich dziedzinach jak e-business, e-health, e-government, e-learning;
- multimedialny charakter aplikacji, wykorzystujących w coraz większym stopniu dźwięk, głos, grafikę, film;
- globalizację gospodarki, a zatem integrację systemów informatycznych — w telekomunikacji, bankowości, edukacji, turystyce, transporcie;
- powszechność i dostępność aplikacji, zwłaszcza internetowych, dla potrzeb społeczeństwa informacyjnego.

Wyzwaniom tym w najwyższym stopniu sprostało podejście obiektowe. Rozwijane od lat 50. XX wieku, w latach 90. znalazło się w centrum zainteresowania badaczy i praktyków. Pojęcie obiektowości początkowo wykorzystywane było w obiektowych językach programowania, takich jak *SIMULA* czy *SmallTalk*. Podejście obiektowe ma obecnie największe znaczenie w następujących **zastosowaniach**:

- metodykach tworzenia oprogramowania (przede wszystkim *Rational Unified Process*);
- graficznych językach ogólnego przeznaczenia (*UML*);
- obiektowych językach programowania (*JAVA*, platforma *.NET*);
- bazach danych (np. *ObjectStore*).

Każdy model obiektowy opiera się na pewnych elementarnych pojęciach i kategoriach, które pod względem metodologicznym stanowią tzw. podstawowy model obiektowy. Model ten jest szczegółowo przedstawiony w wielu opracowaniach naukowych na temat podejścia naukowego. **Główne pojęcia** podstawowego modelu obiektowego — obiekt, klasę, komunikat, hermetyzację, polimorfizm i dziedziczenie — charakteryzuje tabela 1.1.

Tabela 1.1. *Podstawowe pojęcia obiektowości*

Pojęcie	Interpretacja
obiekt (ang. *object*)	Każdy byt — pojęcie lub rzecz — mający znaczenie w kontekście rozwiązywania problemu w danej dziedzinie przedmiotowej.
klasa (ang. *class*)	Uogólnienie zbioru obiektów, które mają takie same atrybuty, operacje, związki i znaczenie.
komunikat (ang. *message*)	Specyfikacja wymiany informacji między obiektami, zawierająca zlecenia wykonania określonej operacji.
hermetyzacja (ang. *encapsulation*)	Różnicowanie dostępu do obiektu poprzez ujawnienie otoczeniu tylko tych informacji o jego atrybutach lub operacjach, które są niezbędne do efektywnego odwoływania się do obiektu w systemie za pośrednictwem komunikatów.
polimorfizm (ang. *polymorphism*)	Możliwość nadawania tej samej nazwy różnym atrybutom i operacjom oraz wykonywania różnych procedur i akcji poprzez operacje o tych samych nazwach; pozwala na redukcję liczby nazw atrybutów i operacji.
dziedziczenie (ang. *inheritance*)	Przyporządkowanie atrybutów i operacji klasom obiektów na podstawie hierarchicznej zależności między nimi. Możliwe jest wielokrotne dziedziczenie, co oznacza, że dana klasa dziedziczy atrybuty i operacje z dowolnej liczby klas nadrzędnych.

Obiektowość stanowi podstawę teoretyczną języka UML (*Unified Modeling Language*). W dalszych punktach niniejszego rozdziału przedstawiono ogólne zagadnienia związane z tym językiem, a więc:

- jego genezę i ewolucję,
- klasyfikację oraz krótką charakterystykę diagramów UML,
- perspektywy architektury systemu,
- mechanizmy rozszerzalności.

Geneza i ewolucja języka UML

W przypadku modelowania obiektowego również nie udało się uniknąć metodycznego rozproszenia. Szczególną dynamiką charakteryzowały się lata 1989 – 1994, kiedy to liczba identyfikowalnych rozwiązań wzrosła z kilku do ponad 50. Jednak w odróżnieniu od metodyk strukturalnych, spory nad kształtem metodyk obiektowych zakończyły się powszechnie uznawanym konsensusem. Autorzy dominujących w omawianym okresie rozwiązań, które ewoluowały wówczas niezależnie od siebie, zaproponowali **ujednolicony język modelowania systemów informatycznych** — UML. Nie należy

przy tym utożsamiać języka UML z pełną metodyką tworzenia systemów informatycznych, która jest znacznie szerszą kategorią (por. rozdział 14.). Podstawowe trzy autorskie źródła języka UML przedstawia tabela 1.2.

Tabela 1.2. *Wkład dominujących podejść w standard UML*

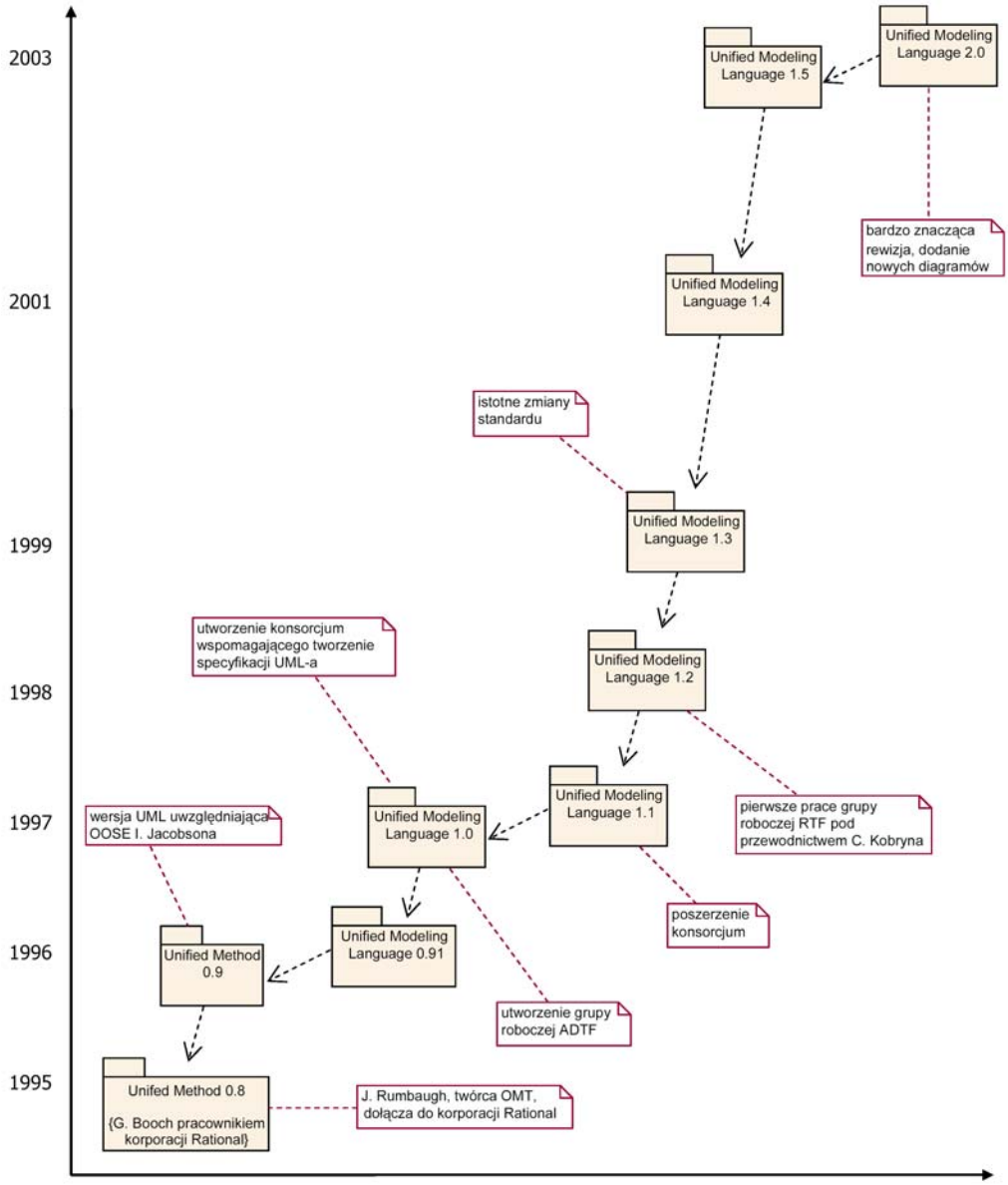
Autor	Nazwa metodyki	Skrót nazwy	Podstawowy wkład	Publikacje
Rumbaugh	*Object Modeling Technique*	OMT	Notacja diagramów UML, analiza i projektowanie	[Rumbaugh-1991]
Booch	*Object Oriented Analysis and Design*	OOAD	Analiza i projektowanie	[Booch-1991]
Jacobson	*Object Oriented Software Engineering*	OOSE	Modelowanie biznesowe, przypadki użycia, diagramy modelowania analitycznego	[Jacobson-1992]

Modelowanie systemów informatycznych obejmuje aspekty funkcjonalne oraz niefunkcjonalne. Obie kategorie dynamicznie rozwijały się w związku z wymaganiami obiektowego wytwarzania oprogramowania. W praktyce analitycy, projektanci czy integratorzy wymagali adekwatnych metod i technik umożliwiających modelowanie złożonych współbieżności, interakcji, maszyn stanowych itd. Zatem możliwość modelowania funkcjonalnych i niefunkcjonalnych aspektów systemów informatycznych była i nadal jest motywem ewolucji języka UML, którego kolejne wersje są wzbogacane o nowe kategorie, będące odpowiedzią na wymagania twórców systemów informatycznych stawiane metodom i technikom modelowania.

W stworzeniu ujednoliconego podejścia upatrywano szansy nie tylko na osiągnięcie większej przejrzystości oferty rynkowej, lecz również na uzyskanie efektu skali poprzez wymianę doświadczeń. Założenia autorów języka UML wykraczały poza dokonanie prostej kompilacji. Przede wszystkim zdawano sobie sprawę z tego, że język modelowania pośredniczy pomiędzy ludzkim rozumieniem funkcjonowania programów komputerowych a ich fizyczną realizacją w postaci kodu źródłowego. Stąd język taki powinien jednocześnie:

- w sposób ścisły definiować podstawowe i zaawansowane kategorie oraz zasady modelowania obiektowego;
- umożliwiać dostosowywanie wykorzystywanej semantyki i notacji do rozwiązywania szerokiego spektrum problemów;
- wykazywać elastyczność wystarczającą do modelowania, obok systemów oprogramowania, także systemów biznesowych;
- wykazywać daleko posuniętą niezależność od konkretnych języków programowania oraz metodyk tworzenia systemów informatycznych;
- uwzględniać skalę realizowanych współcześnie projektów, związanych z bardzo rozbudowanymi systemami o kluczowym znaczeniu dla funkcjonowania przedsiębiorstw.

Ujednolicone podejście, powstałe z uwzględnieniem metodyk źródłowych zaprezentowanych w tabeli 1.1 oraz wymienionych założeń, dynamicznie rozwijało się na przestrzeni ostatnich lat. Historię oraz ścieżkę ewolucyjną języka UML przedstawia rysunek 1.1.



Rysunek 1.1. *Ewolucja języka UML*

Prace nad językiem UML zapoczątkowano w październiku 1994 roku, kiedy J. Rumbaugh dołączył do firmy Rational Software Corporation, gdzie wraz z G. Boochem rozpoczęli pracę nad ujednoliceniem metod OMT i OOAD. Efektem współpracy stał

się zaprezentowany przez firmę Rational na konferencji OOPSLA '95 pierwowzór języka UML. Wersję tę zaprezentowano w październiku 1995 roku. Ten dokument roboczy znany był jako UM (*Unified Method*) 0.8. W tym samym roku do zespołu dołączył I. Jacobson — autor metodyki OOSE. W czerwcu 1996 roku opublikowano wersję 0.9, uwzględniającą już wkład I. Jacobsona, natomiast pod koniec tego samego roku wersję 0.91. Od tego momentu język określa się jako UML.

UML (*Unified Modeling Language*) jest graficznym językiem wizualizacji, specyfikowania, tworzenia i dokumentowania systemów informatycznych.

Partnerami projektu rozwoju UML w korporacji Rational stały się czołowe korporacje świata. Należały do nich: IBM, Digital Equipment Corporation, Hewlett-Packard, Intellicorp, I-Logix, Icon Computing, Oracle, Unisys czy też Microsoft. Ich wsparcie w istotny sposób przyczyniło się do powstania wersji 1.0 w styczniu 1997 roku. Dzięki tej współpracy wersja 1.0 była już silnym i precyzyjnie zdefiniowanym językiem modelowania. Wersja ta została przekazana jako propozycja **standardu** organizacji OMG (*Object Management Group*) — międzynarodowej organizacji, której misją jest promowanie praktyki i teorii technologii obiektowych w tworzeniu oprogramowania. W OMG powołano wówczas grupę ADTF (*Analysis and Design Task Force*), która skoncentrowała się na rozwijaniu standardu.

W lipcu 1997 roku korporacja Rational przedstawia kolejną propozycję UML-a w wersji 1.1, również przekazaną do OMG. Dwa miesiące po jej przekazaniu, w listopadzie 1997 roku, OGM ADTF ostatecznie przyjmuje kolejny standard w wersji 1.1. W tym momencie powoływany jest nowy zespół do spraw zmian standardu UML — OMG RTF (*Revision Task Force*). Na jego czele staje Cris Kobryn, pod przewodnictwem którego w czerwcu 1998 roku powstaje wersja UML 1.2. Zmiany wprowadzone w wersji 1.2 były nieznaczne, koncentrowały się na poprawkach o charakterze redakcyjnym. Wersja 1.2 oraz następująca po niej wersja 1.3 to publikacje wewnętrzne OMG — oficjalnym standardem wciąż pozostawała wersja 1.1. W wersji 1.3 wprowadzono znaczące zmiany w zakresie diagramów przypadków użycia i diagramów czynności. W kwietniu 1999 roku zespół OMG RTF publikuje kolejną wersję — UML 1.4. Jej nieoficjalna dokumentacja była dostępna w połowie 2000 roku, natomiast oficjalną specyfikację wydano we wrześniu 2001 roku. Podobnie jak w przypadku przejścia z wersji 1.1 do 1.2, nie były to daleko idące zmiany — skoncentrowano się na poprawie spójności dokumentacji.

W tym okresie OMG wyznacza wersję 2.0 jako oficjalny kierunek zmian, wyodrębniając z wówczas obowiązującej dokumentacji cztery podlegające różnym zespołom specyfikacje:

- **infrastrukturę UML**, będącą podstawą architektoniczną składni języka UML 2.0, jego metamodelem, pozwalającą na definiowanie i rozwijanie superstruktury UML [OMG-2003b];
- **superstrukturę UML**, specyfikującą ukierunkowane na użytkownika kategorie modelowania — elementarne składniki wykorzystywane w tworzeniu poszczególnych diagramów [OMG-2003a];

- **OCL** — język opisu ograniczeń [Warmer-2003];
- **wymienność diagramów UML**, czyli mechanizm przenoszenia dokumentów zgodnych ze standardem UML pomiędzy różnymi narzędziami [OMG-2003c].

We wrześniu 2002 roku pojawia się jednak nieoficjalnie wersja 1.5, która wprowadza bardzo szczegółowy model akcji. W marcu 2003 roku staje się ona wersją oficjalną.

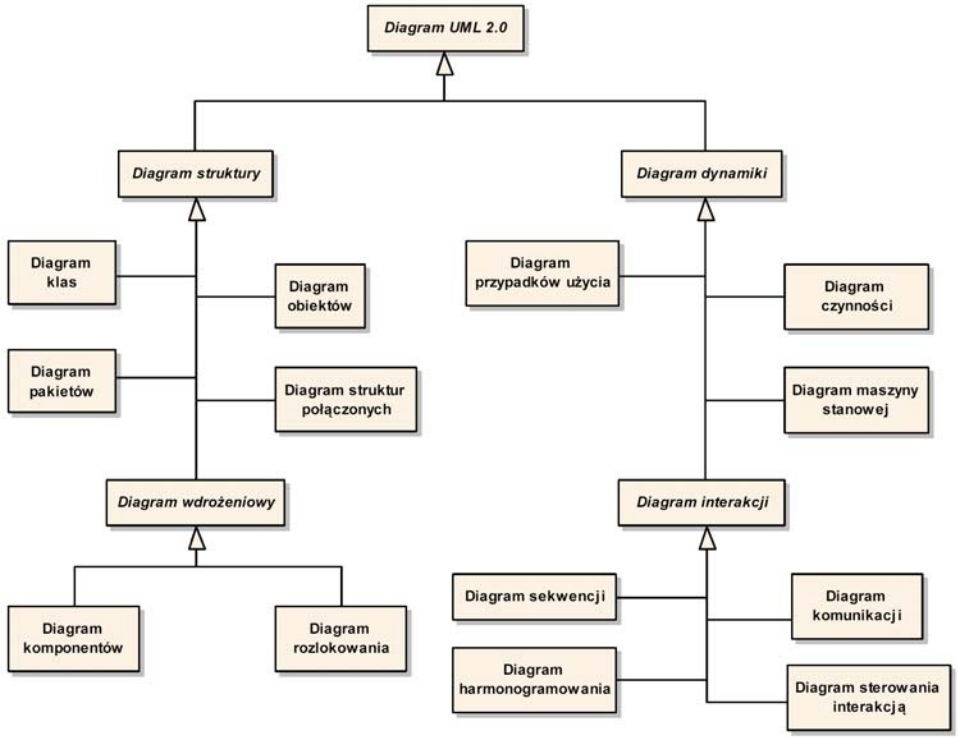
Wersja 2.0 zaprezentowana zostaje w sierpniu 2003 roku. Wersja ta znacząco różni się od poprzednich. Rozszerza ona zakres dostępnych diagramów do trzynastu oraz wprowadza szereg nowych kategorii modelowania w istniejących diagramach. Składnię i klasyfikację wielu istniejących kategorii istotnie zmodyfikowano, opierając je na opracowanej również przez OMG metaskładni **MOF** (*Meta Object Facility*). Na podstawie metaskładni MOF, OMG zdefiniowała mechanizm wymiany dokumentów zgodnych ze standardem UML, wprowadzając standardy XMI (*XML Metadata Interchange*) oraz CWM (*Common Warehouse Metamodel*). **XMI** umożliwia przejście ze specyfikacji UML do dokumentów opisanych w języku XML. Z kolei **CWM** opisuje wymianę metadanych pomiędzy hurtowniami danych, systemami klasy Business Intelligence, systemami zarządzania wiedzą oraz technologiami internetowymi.

Standard UML na bieżąco podlega aktualizacji, modyfikacjom i wzbogacaniu. W 2004 roku dokumentacja UML 2.0 ulega przeredagowaniu, co skutkuje opublikowaniem dokumentu *UML 2.0 Superstructure FTF Convenience Document* [OMG-2004]. Obecnie w pracach OMG współuczestniczy ponad 800 informatycznych organizacji biznesowych i badawczych, wprowadzających kolejne innowacje i rozszerzających zakres dokumentacji. Tym samym, ze względu na charakter i przeznaczenie niniejszej książki, mniejszą wagę przypisano kategoriom modelowania ściśle związanym z programowaniem.

Diagramy UML 2.0

Język UML przyjmuje w praktyce postać graficznej reprezentacji tworzonego systemu, składającej się z logicznie powiązanych z sobą **diagramów**. Pozwalają one na opisanie systemu od modeli ogólnych do bardzo szczegółowych. W standardzie UML w wersji 2.0 występuje trzynaście rodzajów diagramów, które charakteryzują statykę i dynamikę tworzonego systemu. Klasyfikację diagramów UML przedstawia rysunek 1.2.

Na rysunku 1.2 występują kategorie diagramów abstrakcyjnych i konkretnych. Diagramy **abstrakcyjne** są jedynie nazwami uogólniającymi grupy diagramów konkretnych. A zatem do kategorii diagramów abstrakcyjnych zalicza się diagramy struktury, dynamiki, interakcji, wdrożeniowe oraz samą kategorię diagramów UML. Abstrakcyjne kategorie pojęciowe zwyczajowo oznacza się kursywą. Pełny zestaw diagramów **konkretnych**, stanowiących standard języka UML 2.0, syntetycznie scharakteryzowano w tabeli 1.3, a szczegółowo zaprezentowano w kolejnych rozdziałach pierwszej części niniejszego podręcznika. Tabela 1.3 zawiera również wyróżniki rodzaju diagramu, które w unikalny sposób wskazują na jego specyficzną zawartość. Poza wymienionymi na rysunku 1.2 diagramami, środowiska akademickie i profesjonalne użytkujące język UML oraz metodykę RUP zaproponowały inne techniki diagramowe ściśle powiązane ze standardem UML. Są to m.in. diagramy modelowania analitycznego oraz diagramy modelowania biznesowego. Techniki te opisano w części drugiej.



Rysunek 1.2. *Diagramy w języku UML 2.0*

Tabela 1.3. *Proponowane wyróżniki i charakterystyka rodzajów diagramów*

LP	Rodzaj diagramu	Charakterystyka	Wyróżnik rodzaju diagramu		Rozdział
			polski	angielski	
1	Diagram klas (ang. *Class Diagram*)	Diagram klas to graficzne przedstawienie statycznych, deklaratywnych elementów dziedziny przedmiotowej oraz związków między nimi	kls	cld	3
2	Diagram obiektów (ang. *Object Diagram*)	Diagram obiektów to wystąpienie diagramu klas, odwzorowujące strukturę systemu w wybranym momencie jego działania	obk	od	3
3	Diagram pakietów (ang. *Package Diagram*)	Diagram pakietów to graficzne przedstawienie logicznej struktury systemu w postaci zestawu pakietów połączonych zależnościami i zagnieżdżeniami	pkt	pd	13
4	Diagram struktur połączonych (ang. *Composite Structure Diagram*)	Diagram struktur połączonych to graficzne przedstawienie wzajemnie współdziałających części dla osiągnięcia pożądanej funkcjonalności współdziałania	spl	csd	12

Tabela 1.3. *Proponowane wyróżniki i charakterystyka rodzajów diagramów (ciąg dalszy)*

LP	Rodzaj diagramu	Charakterystyka	Wyróżnik rodzaju diagramu		Rozdział
5	Diagram komponentów (ang. *Component Diagram*)	Diagram komponentów to rodzaj diagramu wdrożeniowego, który wskazuje organizację i zależności między komponentami	kmp	cod	11
6	Diagram rozlokowania (ang. *Deployment Diagram*)	Diagram rozlokowania to rodzaj diagramu wdrożeniowego, który przedstawia sieć połączonych ścieżkami komunikowania węzłów z ulokowanymi na nich artefaktami	rzk	dd	11
7	Diagram przypadków użycia (ang. *Use Case Diagram*)	Diagram przypadków użycia to graficzne przedstawienie przypadków użycia, aktorów oraz związków między nimi, występujących w danej dziedzinie przedmiotowej	uzc	ud	2
8	Diagram czynności (ang. *Activity Diagram*)	Diagram czynności to graficzne przedstawienie sekwencyjnych i (lub) współbieżnych przepływów sterowania oraz danych pomiędzy uporządkowanymi ciągami czynności, akcji i obiektów	czn	ad	4
9	Diagram maszyny stanowej (ang. *State Machine Diagram*)	Diagram maszyny stanowej to graficzne odzwierciedlenie dyskretnego, skokowego zachowania skończonych systemów stan-przejście	stn	sm	5
10	Diagram sekwencji (ang. *Sequence Diagram*)	Diagram sekwencji jest rodzajem diagramu interakcji, opisującym interakcje pomiędzy instancjami klasyfikatorów systemu w postaci sekwencji komunikatów wymienianych między nimi	skw	sd	7
11	Diagram komunikacji (ang. *Communication Diagram*)	Diagram komunikacji jest rodzajem diagramu interakcji, specyfikującym strukturalne związki pomiędzy instancjami klasyfikatorów biorącymi udział w interakcji oraz wymianę komunikatów pomiędzy tymi instancjami	kmn	cd	8
12	Diagram harmonogramowania (ang. *Timing Diagram*)	Diagram harmonogramowania jest rodzajem diagramu interakcji, reprezentującym na osi czasu zmiany dopuszczalnych stanów, jakie może przyjmować instancja klasyfikatora uczestnicząca w interakcji	hrm	td	9
13	Diagram sterowania interakcją (ang. *Interaction Overview Diagram*)	Diagram sterowania interakcją jest rodzajem diagramu interakcji, dokumentującym przepływ sterowania pomiędzy logicznie powiązanymi diagramami i fragmentami interakcji z wykorzystaniem kategorii modelowania diagramów czynności	sin	iod	10

W publikacjach polskich dotyczących języka UML, będących najczęściej tłumaczeniami literatury anglojęzycznej, używano synonimów nazw diagramów. Najistotniejsze z nich uwzględniono w słowniku angielsko-polskim (dodatek F). Specyfikacja języka UML wprowadza pojęcie **klasyfikatora** (ang. *classifier*), które ma zastosowanie w odniesieniu do praktycznie każdego rodzaju diagramu języka UML 2.0.

Klasyfikator to abstrakcyjna kategoria modelowania systemu w języku UML, która uogólnia kolekcję instancji o tych samych cechach.

Na konkretnych diagramach języka UML zamieszcza się **instancje** poszczególnych rodzajów klasyfikatorów. W świetle powyższej definicji instancję rozumieć należy następująco:

Instancja jest wystąpieniem, egzemplarzem klasyfikatora.

Tak więc klasyfikatorem jest interfejs, a jego wystąpieniem *IRejestracjaArtykułu*. Za podstawowy rodzaj klasyfikatora uznać należy klasę. Klasa ma instancje w postaci obiektów. Inne omawiane w niniejszej książce rodzaje klasyfikatorów obejmują m.in. węzeł, komponent, aktora, przypadek użycia i pakiet. Ze względu na logikę prezentacji materiału dydaktycznego wybrane rodzaje klasyfikatorów języka UML 2 podsumowano w tabeli 7.3. Obecnie w języku UML istnieje ponad 30 klasyfikatorów. Pełną taksonomię klasyfikatorów UML 2 zawiera dokumentacja OMG [OMG-2004, załącznik F].

W języku UML 2 funkcjonuje również pojęcie klasyfikatora ustrukturyzowanego (ang. *structured classifier*). Klasyfikator ustrukturyzowany zawiera specyfikację wewnętrznej struktury. Pojęcie to odnosi się do diagramów struktur połączonych (por. rozdział 12.).

Dowolny z wymienionych w tabeli 1.3 diagramów może być prezentowany w postaci:

- obramowanej,
- nieobramowanej.

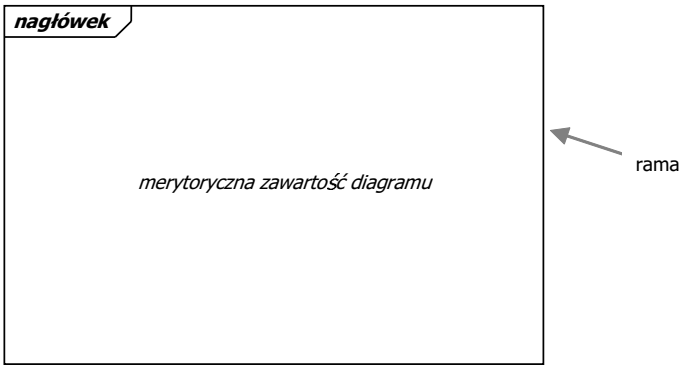
Diagram w postaci obramowanej otoczony jest prostokątną **ramą** (ang. *frame*) zawierającą nagłówek. Przyjmuje on postać pentagramu umieszczonego w lewym górnym rogu diagramu. Typowy nagłówek diagramu UML 2.0 charakteryzuje się formalną, zaprezentowaną poniżej składnią:

<*nagłówek-diagramu*> ::= [<*rodzaj*>] <*nazwa*> [<*parametry*>]

przy czym:

- *rodzaj* — pełny lub skrótowy **wyróżnik diagramu** zawartego w ramie;
- *nazwa* — syntetyczna **nazwa** odzwierciedlająca merytoryczną zawartość diagramu;
- *parametry* — szczegółowe **parametry** kluczowe dla danego diagramu, np. nazwy instancji klasyfikatorów, wartości zwrotne, operatory interakcji.

Notacja ta oznacza, że rodzaj diagramu oraz jego parametry są elementami umieszczanymi fakultatywnie, natomiast nazwa jest obligatoryjna. Elementy diagramu UML w postaci obramowanej przedstawia rysunek 1.3.



Rysunek 1.3. *Format diagramu obramowanego*

W formie obramowanej przedstawić można każdy z 13 rodzajów diagramów języka UML. Podczas tworzenia nagłówków diagramów można skorzystać z polsko- bądź anglojęzycznych wyróżników rodzajów diagramów (tabela 1.3). Kwestia wyróżników nie jest zagadnieniem ujednoliconym. I tak autorzy dokumentacji OMG oznaczają na przykład wszystkie diagramy interakcji wyróżnikiem **sd** (ang. *sequence diagram*). Propozycje i dostępność wyróżników w pakietach CASE wspierających język UML są zróżnicowane.

Konwencja tworzenia diagramów obramowanych, chociaż zalecana od momentu wprowadzenia wersji 2.0 języka UML, nie musi być bezwarunkowo realizowana. Dokumentacja systemu w języku UML, sporządzona z uwzględnieniem informacji zawartych w nagłówkach diagramów obramowanych, w przypadku rozbudowanych systemów poprawia przejrzystość dokumentacji.

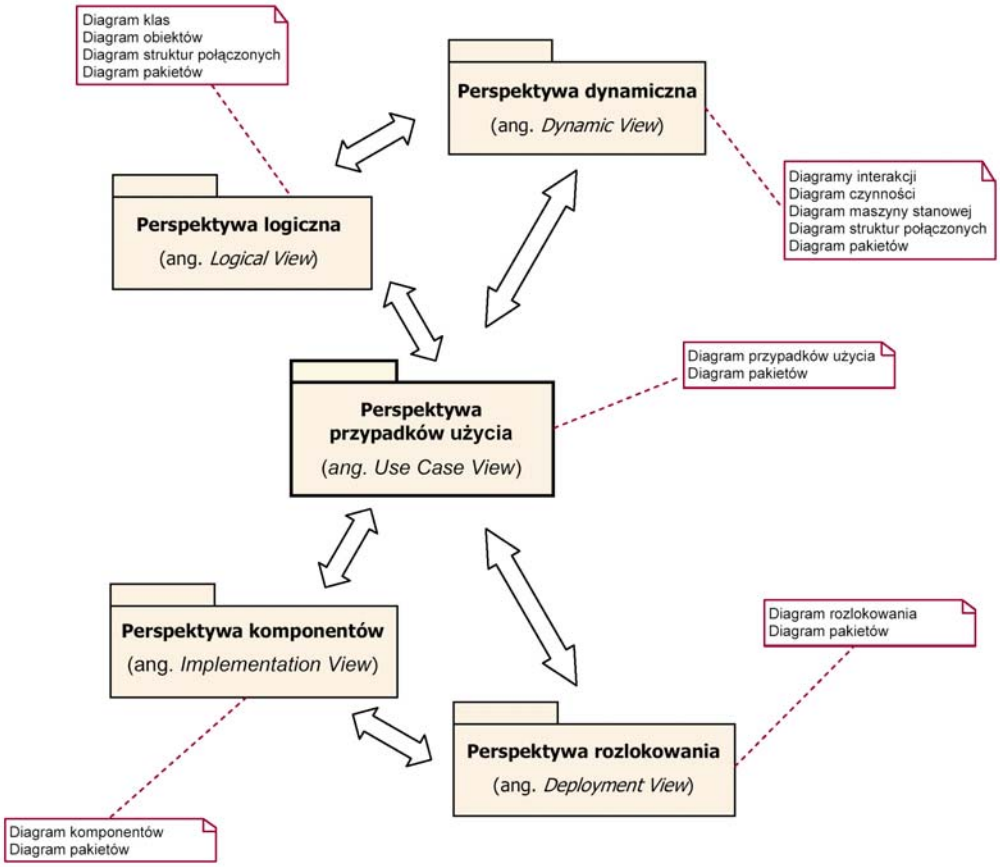
Stosowanie formy obramowanej można rozszerzyć na poszczególne instancje klasyfikatorów języka UML. I tak przedstawienie pakietu systemu w formie ramy może wiązać się z wykorzystaniem nieskrótowego wyróżnika jednoznacznie identyfikującego zawartość ramy jako pakiet — np. wyróżnika **package**. Z kolei zastosowanie ramy do opisu pojedynczego przypadku użycia opiera się na wyróżniku **useCase**.

Perspektywy w opisie architektury systemu

W skutecznym tworzeniu i użytkowaniu systemów informatycznych bierze udział wiele osób o różnych kompetencjach zawodowych i organizacyjnych. Są to właściciele, menedżerowie, analitycy, projektanci, programiści, testujący itd. Każda z tych grup zawodowych ma własny pogląd na system informatyczny, toteż poprzez swoje decyzje powinna mieć wpływ na jego architekturę. Ta różnorodność spojrzeń była zasadniczo nieuwzględniana w podejściu strukturalnym do TSI, a akcentowana w podejściu społecznym. W kontekście języka UML zróżnicowanie punktów widzenia znajduje swój wyraz w postaci powiązanej z nim metodyki RUP, proponującej pięć **perspektyw** (ang. *views*) architektury systemu informatycznego. Są to:

- perspektywa przypadków użycia — kluczowa i dominująca wobec pozostałych, definiuje zakres i oczekiwaną funkcjonalność tworzonego systemu;
- perspektywa dynamiczna — wskazuje, w jaki sposób jest realizowane zachowanie instancji klasyfikatorów w systemie;
- perspektywa logiczna — dokumentuje statykę systemu;
- perspektywa komponentów — przeznaczona głównie dla programistów, specyfikuje oprogramowanie na poziomie komponentów;
- perspektywa rozlokowania — specyfikuje sprzęt informatyczny niezbędny do funkcjonowania konkretnych komponentów systemu.

Każda z perspektyw uwypukla zatem inne aspekty architektury systemu, pomijając jednocześnie szczegóły nieistotne z punktu widzenia danej grupy zawodowej. Perspektywy wspierane są poprzez określone **zestawy diagramów** (rysunek 1.4).



Rysunek 1.4. *Modelowanie perspektyw architektury systemu*

Mechanizmy rozszerzalności

Język UML zapewnia bogaty zestaw pojęć związanych z modelowaniem systemów informatycznych oraz elementów notacji przydatnych w zapisywaniu typowych projektów systemów, niemniej odbiorcy mogą wymagać dodatkowych cech wykraczających poza te zdefiniowane w specyfikacji języka. Z tego powodu autorzy zdecydowali się na włączenie do języka UML zestawu **mechanizmów rozszerzalności** (ang. *extension mechanisms*), pozwalającego na używanie przez twórców systemu nowych kategorii modelowania specyficznych dla danej dziedziny zastosowania. Umożliwiają one wzbogacanie już istniejących kategorii pojęciowych o dodatkowe informacje, jak również modyfikację znaczeniową tych kategorii. W języku UML wyróżnia się trzy rodzaje mechanizmów rozszerzalności:

- stereotyp,
- ograniczenie,
- metka.

Stereoptyp

Stereotypów (ang. *stereotypes*) używa się do klasyfikowania bądź oznaczania istniejących elementów modelu obiektowego oraz wprowadzania nowych kategorii modelowania, wywodzących się z już istniejących. I tak:

Stereotyp to mechanizm rozszerzalności, który wzbogaca zbiorowość standardowych kategorii modelowania języka UML.

Wzbogacenie, rozszerzenie o stereotypy w praktyce oznacza wprowadzenie nowych kategorii modelowania w poszczególnych diagramach języka UML, przy czym kategorie te bazują na semantyce standardowych, istniejących kategorii modelowania.

Wyróżnia się stereotypy **tekstowe** oraz **graficzne**. Nazwa stereotypu tekstowego zwyczajowo ujmowana jest w podwójny cudzysłów ostrokątny i umieszczana na stereotypowanym elemencie. Przykładowym stereotypem tekstowym odnoszącym się do:

- pakietów jest <<*subsystem*>>,
- związku zależności — <<*include*>> czy też <<*extend*>>,
- komponentu — <<*file*>>.

Z kolei stereotypy graficzne mają domyślnie postać specyficznych symboli graficznych umieszczanych na stereotypowanych elementach. Zastosowanie stereotypów nie ogranicza w ten sposób twórcy systemu, który ma daleko posuniętą swobodę w używaniu i proponowaniu różnorodnych symboli, specyficznych dla danej aplikacji.

Twórca systemu ma do dyspozycji znaczną liczbę stereotypów standardowych, rekomendowanych przez autorów języka UML oraz OMG — i w konsekwencji powszechnie stosowanych. Zostały one wymienione i scharakteryzowane w dokumentacji standardu [OMG-2004, załącznik C].

Ograniczenie

Kolejnym mechanizmem rozszerzenia jest **ograniczenie** (ang. *constraint*).

> Ograniczenie to wyrażenie semantyczne określające warunek bądź zastrzeżenie związane z ograniczanym elementem modelowania bądź grupą elementów.

Ograniczenie jest w istocie tekstem wyrażonym w języku naturalnym, formułą matematyczną, predykatem logiki formalnej bądź instrukcją pseudokodu. W standardzie UML definiować je można w dedykowanym języku ograniczeń OCL (*Object Constraint Language*), zawierającym formalną składnię ograniczeń obiektowych. Ograniczenia umieszczane są w nawiasach klamrowych w bezpośrednim sąsiedztwie elementu lub elementów, których znaczenie jest precyzowane;

np. {*disjoint*}

{*czas < 15 min*}.

Metka

Każda kategoria modelowania języka UML charakteryzuje się specyficznym zestawem właściwości. Taki zbiór właściwości może być rozszerzany z wykorzystaniem **metek** (ang. *tagged values*).

> Metka stanowi jawne zdefiniowanie właściwości w postaci przyporządkowania nazwa-wartość.

Konwencja wprowadzania metek do diagramów jest podobna jak w przypadku ograniczeń — jest to zapis w nawiasach klamrowych, składający się z nazwy, separatora oraz wartości (z zastrzeżeniem, że jest on umieszczany bezpośrednio pod nazwą danego elementu). Dozwolone jest dołączanie metek także do stereotypów. Najpowszechniej metki stosuje się celem określenia właściwości istotnych w generowaniu kodu oraz zarządzaniu konfiguracjami;

np. {*wersja = beta*}

{*model = HP LaserJet 1500L*}.

Podstawowe pojęcia

- Common Warehouse Metamodel
- Computer-Aided Software Engineering
- Cykl życia systemu
- Diagram UML
  - Abstrakcyjny
  - Konkretny
- Diagramy dynamiki UML
  - Diagram czynności
  - Diagramy interakcji
    - Diagram harmonogramowania
    - Diagram komunikacji
    - Diagram sekwencji
    - Diagram sterowania interakcją
  - Diagram maszyny stanowej
  - Diagram przypadków użycia
- Diagramy struktury UML
  - Diagram klas
  - Diagram obiektów
  - Diagram pakietów
  - Diagram struktur połączonych
  - Diagramy wdrożeniowe
    - Diagram komponentów
    - Diagram rozlokowania
- Klasyfikator
- Ustrukturyzowany
- Mechanizmy rozszerzalności
  - Metka
  - Ograniczenie
  - Stereotyp
    - Graficzny
    - Tekstowy
- Metodyka
  - Obiektowa
    - OMT
    - OOAD
    - OOSE
  - Społeczna
  - Strukturalna
- Meta Object Facility
- Modelowanie systemów
- Nagłówek diagramu
  - Nazwa
  - Wyróżnik diagramu
  - Parametr
- Notacja
- Obiektowość
  - Dziedziczenie
  - Hermetyzacja
  - Klasa
  - Komunikat
  - Obiekt
  - Polimorfizm
- Object Constraint Language
- OMG
  - ADTF
  - RTF

Perspektywa
- Dynamiczna
- Komponentów
- Logiczna
- Przypadków użycia
- Rozlokowania

Rama
Superstruktura UML
System informatyczny
Tworzenie systemów informatycznych (TSI)
UML
Wymienność diagramów UML
XML Metadata Interchange

Pytania i zadania

1. Scharakteryzuj trendy rozwoju modelowania systemów informatycznych.
2. Jakie było znaczenie i skutki inicjatyw unifikacyjnych w dziedzinie tworzenia systemów informatycznych?
3. Jakie zjawiska sprzyjały wzrostowi znaczenia obiektowości w informatyce?
4. Podaj przykłady obszarów i narzędzi, w których model obiektowy znajduje zastosowanie.
5. Wskaż i scharakteryzuj kluczowe pojęcia obiektowości.
6. Zidentyfikuj nazwy kilku klas odnoszących się do środków transportu. Wskaż przykładowe obiekty tych klas.
7. Zaprezentuj mechanizm dziedziczenia, wykorzystując zidentyfikowane klasy środków transportu.
8. Wymień metodyki źródłowe języka UML.
9. Jakie cele sformułowane zostały w trakcie tworzenia języka UML?
10. Podaj definicję języka UML.
11. Wyjaśnij zależność pomiędzy korporacją Rational oraz organizacją OMG w kontekście rozwoju standardu UML.
12. Przedstaw fazy rozwoju języka UML.
13. Wyjaśnij relacje pomiędzy pojęciami:
    - ♦ superstruktura UML,
    - ♦ infrastruktura UML,
    - ♦ XMI,
    - ♦ wymienność diagramów UML,
    - ♦ MOF.

14. Czy modelowanie w języku UML ogranicza się do systemów informatycznych? Odpowiedź uzasadnij.

15. Przedstaw klasyfikację diagramów UML 2.0. Jakie diagramy nie występowały we wcześniejszych wersjach tego standardu?

16. Jakie postacie może przyjąć dowolny diagram? Wskaż informacje zawarte w nagłówku diagramu.

17. Wyjaśnij zależności pomiędzy następującymi pojęciami:

- klasyfikator,
- instancja,
- klasa,
- obiekt.

18. Jakie perspektywy występują w opisie architektury systemu? Omów istotę każdej z nich.

19. Zapisz w jednej kolumnie wszystkie perspektywy architektury systemu, a w drugiej poszczególne diagramy UML. Następnie powiąż odpowiednie perspektywy z odpowiadającymi im diagramami.

20. Jaka jest rola mechanizmów rozszerzalności?

21. Wymień podstawowe mechanizmy rozszerzalności języka UML.

22. Opracuj przykłady:

- stereotypów tekstowych,
- ograniczeń,
- metek.

23. Modyfikacja sieci informatycznej uniwersytetu została udokumentowana na stosownych diagramach. Zaprojektuj stereotypy graficzne odnoszące się do poszczególnych elementów sprzętu sieciowego.

24. Odszukaj w Internecie pod adresem *www.omg.org* aktualne specyfikacje języka UML.