

SYSTEMY CZASU RZECZYWISTEGO

SCR '99

Centrum Wiedzy i Informacji
Naukowo-Technicznej Politechniki Wrocławskiej



100100511495

OCW
8

ZBIGNIEW HUZAR

Wprowadzenie do języka UML

Zakopane 1999

4.7



BW-8.

**SYSTEMY CZASU RZECZYWISTEGO
SCR '99**

ZBIGNIEW HUZAR

Wprowadzenie do języka UML

Zakopane 1999

111675

B I B L I O T E K A
WYDZIAŁOWEGO ZAKŁADU INFORMATYKI
Wydział Informatyki i Zarządzania
Politechniki Wrocławskiej

6636/1

Biblioteka Główna i OIINT
Politechniki Wrocławskiej



100027298

BW-8

Warszawa 1999

© Copyright by Zbigniew Huzar, Warszawa, 1999

Redaktor: Jan Chudzikiewicz

Projekt okładki: Piotr Chęć



315707/1

Wydawca: Instytut Automatyki i Robotyki
Wydział Cybernetyki
Wojskowa Akademia Techniczna
00-908 Warszawa 49, ul. Kaliskiego 2
tel. (0-22) 685-95-52, fax (0-22) 666-90-41
<http://www.iar.wat.waw.pl>

Druk: BEL Studio Sp. z o.o., 01-355 Warszawa, ul. Powstańców Śląskich 67 B
tel./fax (0-22) 665-92-22

ISBN 83-912747-4-8

19Kc. 2004/0015/D

158

Spis treści

1. Uwagi wstępne	5
2. Podstawowe pojęcia modelowania	13
3. Elementy języka	23
3.1. Klasyfikacja	23
3.2. Elementy strukturalne	24
3.3. Elementy behawioralne	35
3.4. Elementy grupujące i objaśniające	37
3.5. Relacje	38
4. Diagramy klas i diagramy obiektów	43
5. Diagramy przypadków użycia	44
6. Diagramy interakcji	46
7. Diagramy stanów	49
8. Diagramy aktywności	58
9. Diagramy implementacyjne	61
10. Diagramy pakietów	63
11. Wybrane standardowe elementy UML	64
12. Elementy metamodelu UML	67
13. Metodyka ROPES	68
13.1. Analiza	69
13.2. Projektowanie	73
Literatura	78

1. Uwagi wstępne

Język UML (*ang. Unified Modeling Language*) jest uniwersalnym językiem przeznaczonym do specyfikacji, wizualizacji i konstruowania modeli oprogramowania oraz do tworzenia dokumentacji powstającej w procesie wytwarzania oprogramowania. Język opiera się na stosowanym w ostatnim okresie podejściu obiektowym w wytwarzaniu oprogramowania. Prekursorami UML są rozwijane od końca lat osiemdziesiątych języki specyfikacji obiektowych, wśród których największą rolę odegrały języki związane z metodą OOAD (*ang. Object-Oriented Design with Applications*) Grady Boocha, metodą OMT (*ang. Object Modelling Techniques*) Jima Rumbaugh oraz OOSE (*ang. Object-Oriented Software Engineering*) Ivara Jacobsona. Właśnie wysiłek wymienionych autorów, którzy podjęli wspólne przedsięwzięcie w ramach firmy *Rational Software Corporation*, doprowadził w 1996 roku do opracowania pierwszych wersji UML. We wrześniu 1997 roku powstała, najbardziej obecnie rozpowszechniona wersja 1.1, oraz pod koniec 1998 roku, opisywana w materiale, ostatnia wersja 1.3. UML był opracowywany z szerokim udziałem producentów oprogramowania, a także we współpracy z wpływową organizacją OMG (*ang. Object Management Group*) – międzynarodową grupą zrzeszającą producentów i użytkowników oprogramowania, zaangażowaną w opracowywanie i promocję standardów związanych z technologią obiektową.

Szerokie zaangażowanie wielu organizacji w powstanie i rozwój UML jest miarą dojrzałości technik obiektowych, a także wyrazem oczekiwania na ich wykorzystanie w wytwarzaniu oprogramowania na skalę przemysłową. Oznacza to nie tylko opracowanie języka, lecz również metod oraz narzędzi wspomagających proces wytwarzania oprogramowania wykorzystujących ten język.

UML jako język modelowania może być użyty we wszystkich zasadniczych fazach cyklu wytwarzania oprogramowania. W szczególności oznacza to jego wykorzystanie do:

- specyfikacji analizowanych systemów,
- specyfikacji projektowanego oprogramowania,
- konstruowania (projektowania) oprogramowania,
- wizualizacji i dokumentowania artefaktów związanych z wytwarzaniem oprogramowania; pod pojęciem artefaktu rozumie się tutaj informację wykorzystaną bądź wytworzoną w trakcie procesu tworzenia oprogramowania.

Dodatkowymi celami branymi pod uwagę podczas projektowania UML było dostarczenie ekspresywnego, uniwersalnego języka, który byłby językiem rozszerzalnym, modyfikowanym stosownie do konkretnych potrzeb. Uniwersalizm oznacza tu, że ma UML dostarczać pełnego repertuaru abstrakcyjnych, dobrze zdefiniowanych pojęć związanych z procesem wytwarzania oprogramowania.

UML wyrasta z dotychczasowej praktyki wytwarzania oprogramowania, ale jest niezależny zarówno od konkretnych języków programowania jak i od konkretnych metod tworzenia oprogramowania. UML nie narzuca więc konkretnej metody, ale zachęca jednak do iteracyjnego i przyrostowego procesu wytwarzania oprogramowania, opartego na analizie przypadków użycia (usług elementarnych) oraz ukierunkowanego na architekturę projektową. Opracowanie UML stało się bodźcem do opracowania metod i narzędzi wspomagających stosowanie tych metod. Między innymi firma *Rational Software Corporation* opracowała metodę (ang. *Rational Objectory Process*) i związany z nią pakiet oprogramowania wspomagający tworzenie oprogramowania. Inne metody i narzędzia przygotowały firma *Select* oferując kolejną wersję pakietu *Select Enterprise 6.0* oraz firma *i-Logix*, wytwórca znanego pakietu *Statemate*, oferując pakiet *Rhapsody*. Pakiety te nie są jeszcze całkiem dojrzałe, gdyż nie umożliwiają wszystkich prezentacji, które dopuszcza UML.

Definicja języka UML, przedstawiona w dokumentach [17], [18], [20] jest częściowo formalna. Formalnie zdefiniowano abstrakcyjną składnię bezkontekstową oraz ograniczenia kontekstowe, wyrażane jako formuły pomocniczego języka OCL (ang. *Object Constraint Language*) [19], który jest klasycznym językiem predykatów

pierwszego rzędu. Semantyka UML, tworząca tzw. metamodel, jest natomiast przedstawiona w języku naturalnym, przy czym zasadniczymi pojęciami służącymi do wyjaśniania elementów języka są tzw. metaklasy oraz metarelacje. Pojęcia te są uogólnieniem pojęcia klas i relacji, czyli tych elementów, które mają być definiowane.

Ten sposób definiowania języka może być uważany za błędny, gdyż polega na definiowaniu nieznanego przez nieznanego. To samo pojęcie, na przykład pojęcie klasy, występuje raz w roli pojęcia definiowanego, a drugi raz, nazywane metaklasą, występuje w roli pojęcia wyjaśniającego. Sytuacja ta wymaga bliższego wyjaśnienia. Z definicją i użyciem języka UML wiążą się cztery poziomy abstrakcji, w oryginalnych dokumentach standaryzacyjnych nazywane poziomami architektury metamodelu. Są to:

- meta-metamodel,
- metamodel,
- model (abstrakcyjny),
- model użytkowy.

Najwyższy poziom abstrakcji – poziom meta-metamodelu – odnosi się do języka służącego do definiowania UML. Poziom niższy – poziom metamodelu – odnosi się do definicji języka. Zatem meta-metamodel zawiera język potrzeby do definiowania UML, czyli względem UML jest jego metajęzykiem. Kolejny niższy poziom – poziom modelu – odnosi się do definicji modelu interesującego nas systemu, wyrażonej za pomocą języka UML. Wreszcie poziom najniższy – poziom modelu użytkowego – odnosi się do konkretnej komputerowej reprezentacji modelu, na przykład reprezentacji dostarczanej przez pakiet wspomagający wytwarzanie oprogramowania.

Przedstawioną architekturę metamodelu można byłoby zastosować do opisu języka programowania. Poziom meta-metamodelu odpowiadałby metajęzykowi służącemu do opisu języka, poziom metamodelu – opisowi języka, poziom modelu – tekstowi źródłowemu programowu napisanego w tym języku, a poziom modelu użytkowego – postaci wykonywalnej tego programu.

W opracowanych standardach uwagę skoncentrowano na poziomie metamodelu, domyślnie zakładając znajomość meta-metapoziomu. Akceptacja architektury metamodelu oznacza więc, że aby zrozumieć metamodel czytelnik powinien mieć ogólne pojęcie klasy, właściwie metaklasy. Natomiast studiując metamodel może upewnić się, że pojęcie klasy w UML jest uściśleniem jego rozumienia metaklasy.

Obecnie trudno byłoby UML uznać za język ostatecznie ukształtowany. Poza przedstawionymi brakami dotyczącymi nie w pełni zdefiniowanej semantyki, należy zwrócić uwagę jeszcze na dwa aspekty.

Z jednej strony UML jest bogatym, w pewnym sensie nadmiernie bogatym językiem. Oznacza to, że niektóre mechanizmy języka można uznać za nadmiarowe. Przypomina to sytuację jaka towarzyszyła wczesnej fazie rozwoju języków programowania. Przykładowo, definiowany w latach sześćdziesiątych język programowania Algol-60 posiadał wiele barokowych instrukcji złożonych, które ostatecznie zostały wyeliminowane w Pascalu powstałym na początku lat siedemdziesiątych. Zasadniczą przyczyną nadmiarowości jest włączenie do języka wielu mechanizmów, z których każdy oddzielnie może nie być nadmiarowy, ale – posiadając wspólne elementy – powodują powstanie nadmiarowości. Praktycznym skutkiem nadmiarowości mogą być niejednoznaczności w interpretacji modeli powstałych w wyniku swobodnego składania elementów języka.

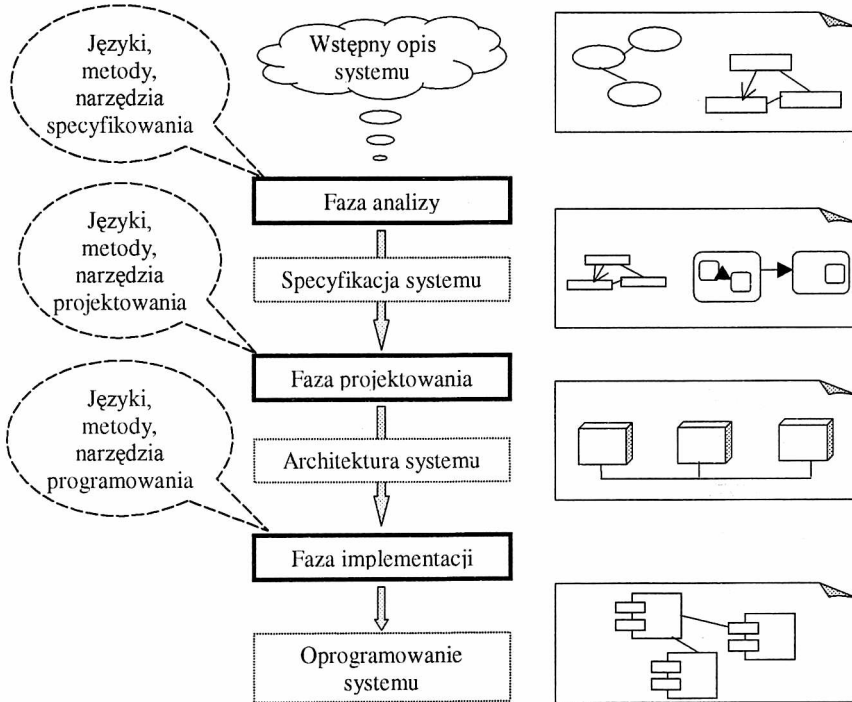
Z drugiej strony można też wskazać na pewne niedostatki UML, które nie pozwalają na wyrażanie pewnych własności, które są możliwe do wyrażenia w innych językach specyfikacji [8].

Jeszcze jedna ważna okoliczność jest związana z praktycznym użytkowaniem języka przy pomocy pakietów wspomagających. Pakiety takie nie zawsze implementują wszystkie mechanizmy języka, specyficznie interpretują pewne szczególne sytuacje i często dołączają własne pomocnicze mechanizmy.

Pamiętając przytoczone uwagi, czytelnik powinien więc wykazywać ostrożność i docieklivość przy posługiwaniu się językiem. Jednocześnie należy oczekiwać, że w

najbliższych lat będą pojawiać się nowe, udoskonalone wersje UML.

Zasadniczym celem materiału jest przybliżenie podstawowych pojęć języka i wskazanie na możliwości ich użycia w procesie wytwarzania oprogramowania. Nieformalnie, miejsce języka w procesie wytwarzania ilustruje rys. 1.



Rys. 1. Ilustracja miejsca UML w procesie wytwarzania oprogramowania

W uproszeniu zakłada się, że proces ten składa się z trzech faz: analizy wymagań, projektowania, implementacji. Punktem wyjścia do całego procesu jest wstępny opis problemu. Kolejna faza bazuje na wynikach fazy poprzedniej i dostarcza informacji dla fazy następnej. Ponadto, każda faza opiera się na pewnej pragmatyce postępowania, która decyduje o metodyce postępowania – na przykład o wyborze języków prezentacji wyników, narzędzi wspomagających. Niezbędna tu wiedza pochodzi oczywiście spoza

dziedziny, która jest źródłem rozwiązywanego problemu. W wyniku realizacji kolejnych faz powstają pewne dokumenty – odpowiednio: specyfikacja, projekt i oprogramowanie systemu. Każdy z tych dokumentów stanowi pewien fragmentaryczny opis całego systemu. Inaczej: każdy z nich jest pewnym modelem przedstawiającym cały system z pewnego punktu widzenia (z pewnej perspektywy). UML jest językiem, który pozwala na wyrażanie wszystkich takich modeli. Modele te są symbolicznie zaznaczone po prawej stronie rys. 1 w postaci zestawu – pakietu – pewnych diagramów i ewentualnych komentarzy.

UML jest językiem graficzno-tekstowym. Oznacza to, że w modelu systemu opisywanego w języku UML występują symbole graficzne i tekstowe. Zasadniczo przy tworzeniu modelu główną rolę mają odgrywać symbole graficzne, a symbole tekstowe mają stanowić ich uzupełnienie. Symbole graficzne wykorzystuje się do budowy specyficznych grafów – diagramów.

diagram

Graficzne przedstawienie zbioru elementów modelu, najczęściej prezentowane w postaci grafu, w którym wierzchołki reprezentują elementy modelu, a łuki – zachodzące pomiędzy nimi relacje. Wyróżnia się diagramy: klas, obiektów, przypadków użycia, sekwencji, współdziałania, stanów, aktywności, komponentów i rozmieszczenia.

Na diagramach a także poza diagramami mogą występować teksty. Teksty mogą mieć charakter formalny lub nieformalny. Formalne teksty mogą być wyrażane w dowolnym języku formalnym, na przykład w języku OCL [19], a nieformalne – w dowolnie ustalonym języku. Informacje tekstowe mogą dotyczyć elementów diagramu, czyli pojedynczych wierzchołków lub łuków, bądź też diagramów jako całości, a nawet kolekcji diagramów.

Uwaga

Przedstawiona wyżej definicja diagramu, wyróżniona w postaci przesuniętego akapitu, jest tłumaczeniem odpowiedniej pozycji słownika pojęć UML [20].

Konwencja ta będzie zachowana w dalszej części opracowania.

UML jest językiem rozszerzalnym. Oznacza to, że UML zawiera mechanizmy, które można wykorzystać w celu zdefiniowania nowych konstrukcji przystosowanych do konkretnych zastosowań. Obecnie zdefiniowano dwa standardowe rozszerzenia języka UML: *Objectory Process* oraz *Business Engineering* [20]. Rozszerzenie języka oznacza modyfikację semantyki elementów metamodelu. Pojęcie rozszerzenia UML należy odróżnić od pojęcia wariantu UML. Pod pojęciem wariantu rozumie się specjalizację języka, która zachowuje semantykę pojęć metamodelu języka.

Ścisłe biorąc rozszerzenie języka odnosi się tylko do składni, gdyż polega na wprowadzeniu nowego symbolu graficznego lub słowa. Natomiast z punktu widzenia semantyki wprowadzenie nowego elementu wiąże się nie z rozszerzeniem, ale z uściśleniem znaczenia istniejącego już elementu.

Efektem każdej z faz jest pewien model systemu budowany z określonego punktu widzenia – określonej perspektywy. Model jest przedstawiany za pomocą odpowiednio dobranego zestawu diagramów oferowanych przez UML wraz z uzupełniającą informacją tekstową.

Niektóre spośród diagramów są przeznaczone do konkretnych faz, inne mogą być używane w różnych fazach. Na przykład, diagramy przypadków użycia (być może lepszym określeniem byłoby – diagramy usług) są przeznaczone do fazy analizy wymagań, a diagramy komponentów i rozmieszczenia są związane z fazą implementacji, natomiast diagramy klas i stanów mogą być wykorzystywane w zasadzie we wszystkich fazach.

Diagramy współdziałania i sekwencji noszą również wspólną nazwę *diagramów interakcji*, zaś diagramy komponentów i rozmieszczenia – *diagramów implementacji*.

UML dopuszcza również użycie innych rodzajów diagramów. Przykładem są tu diagramy pakietów. Pełnią one rolę pomocniczą w każdej fazie wytwarzania oprogramowania i zasadniczo służą do zestawiania w całość różnych innych diagramów i pojęć pomocniczych, umożliwiając w ten sposób tworzenie dokumentacji prac nad

oprogramowaniem – służą do przedstawiania modeli systemów. Dodatkowo, w przypadku złożonych systemów, mogą służyć do przedstawiania modeli ich komponentów – podsystemów.

Przykładowo, rola UML w procesie wytwarzania oprogramowania – przy założeniu jego podziału na fazy analizy, projektowania oraz implementacji – może być widziana następująco.

Celem fazy analizy jest opis systemu z punktu widzenia jego przyszłych użytkowników. Widzą oni system przez zestaw usług, których system dostarcza oraz przez sposób komunikacji z systemem podczas korzystania z tych usług. Model systemu jest tu przedstawiany przez diagram przypadków użycia, który przedstawia usługi systemu oraz łączące je relacje oraz tzw. aktorów reprezentujących obiekty z otoczenia systemu. Obiektami tymi mogą być użytkownicy, urządzenia pomiarowe lub wykonawcze, lub inne systemy. Diagram przypadków użycia jest zwykle uzupełniany nieformalnym opisem znaczenia poszczególnych przypadków (usług) oraz łączących ich relacji. Interpretacja przypadków użycia wymaga zbudowania diagramów klas, które wyrażają aspekty statyczne, oraz diagramów interakcji, które wyrażają aspekty dynamiczne modelu.

W fazie projektowania tworzy się model, w którym wyraża się sposób reprezentacji i funkcjonowania elementów modelu uzyskanego w poprzedniej fazie. Przy definiowaniu tego modelu bierze się pod uwagę dostępne środki i związane z nimi ograniczenia. Do definiowania modelu projektu można używać tych samych diagramów co w fazie poprzedniej, a także diagramów komponentów i rozmieszczenia.

Faza implementacji, rozumiana tylko jako faza programowania, w zasadzie nie wykorzystuje diagramów UML. Dopuszcza się jednak użycie języków programowania w niektórych diagramach do definiowania znaczenia pewnych elementów, na przykład do definiowania metod na diagramach klas.

2. Podstawowe pojęcia modelowania

UML jest językiem, przy pomocy którego opisuje się modele interesujących nas systemów. W rozdziale przedstawia się określenia pojęć dotyczących modelowania. Pojęcia te są potrzebne do właściwego rozumienia dowolnego języka modelowania oraz do poprawnego posługiwania się takim językiem. Z tego względu pojęcia dotyczące modelowania są przedmiotem zainteresowania instytucji standaryzacyjnych. Zebrane i komentowane niżej terminy i pojęcia bezpośrednio związane z UML są kojarzone z odpowiednimi terminami i pojęciami opracowanymi w ramach prac prowadzonych przez ISO nad modelem systemów przetwarzania rozproszonego ODP (*ang. Open Distributed Processing*) [10].

Modelowanie, czyli tworzenie pewnego *modelu* odnosi się do wybranej *dziedziny*. Zarówno wybrany fragment pewnej dziedziny, jak i zbudowany model stanowią pewne *systemy*. Model jest systemem stanowiącym tylko pewne odzwierciedlenie systemu z wybranej dziedziny. Model jest więc pewną *abstrakcją* modelowanej dziedziny. Związek pomiędzy modelem a modelowaną dziedziną zależy od przyjętego punktu widzenia – od wybranej *perspektywy* modelowania.

dziedzina

Obszar wiedzy lub działalności charakteryzujący się specyficznym zbiorem pojęć i terminologią.

model

Semantycznie spójna abstrakcja systemu.

system

Zbiór wspólnie powiązanych jednostek w celu wykonywania wspólnych zadań. System może być opisywany przez jeden lub więcej modeli odzwierciedlających różne perspektywy widzenia systemu. W przypadkach złożonych system może być przedstawiony jako złożenie podsystemów.

podsystem

Zbiór elementów opisujących zachowanie innych, zagnieżdżonych w nich elementów.

To czym są jednostki składowe modelu (systemu) zależy od przyjętych zasad dekompozycji i abstrakcji. W UML mogą to być, na przykład, obiekty (faza analizy), komponenty lub węzły (faza projektowania).

Pojęcie dekompozycji bezpośrednio w UML nie występuje. Dekompozycja polega na przedstawieniu charakterystyki bytu przez odwołanie się do charakterystyk innych bytów – jego części składowych. Pozwala to na oddzielne rozpatrywanie najpierw części składowych, a następnie całości w terminach odpowiedniego połączenia tych składowych.

abstrakcja

Specyficzna charakterystyka bytu, która pozwala na jego wyróżnienie wśród innych bytów. Abstrakcja ustala rozgraniczenia pomiędzy bytami względem perspektywy przyjętej przez obserwatora.

Abstrakcję określa się również jako proces bądź jako efekt procesu polegającego na usuwaniu nieistotnych szczegółów w celu uzyskania uproszczonego opisu interesującego systemu.

Dekompozycja i abstrakcja są dwoma podstawowymi sposobami redukcji złożoności podczas modelowania.

perspektywa

Projekcja modelu, w której ujęte są elementy istotne z danego punktu widzenia (poziomu abstrakcji), a pominięte są wszelkie inne elementy.

UML jest językiem modelowania. Podczas tworzenia oprogramowania przechodzi się przez kolejne fazy, które charakteryzują się specyficznymi perspektywami.

Przykładami perspektyw w metodzie *Unified Software Development Process* [2] są:

- perspektywa usługowa (*ang. use case view*),

- perspektywa projektowania ogólnego (*ang. design view*),
- perspektywa projektowania szczegółowego (*ang. process view*),
- perspektywa implementacyjna (*ang. implementation view*),
- perspektywa instalacyjna (*ang. deployment view*).

Inne perspektywy wyróżnia model ODP [10]. Są to:

- perspektywa przedsięwzięcia (*ang. enterprise view*),
- perspektywa informacyjna (*ang. information view*),
- perspektywa obliczeniowa (*ang. computation view*),
- perspektywa inżynierska (*ang. engineering view*),
- perspektywa technologiczna (*ang. technological view*).

okres modelowania

Odnosi się do tego co występuje w fazie modelowania w procesie wytwarzania oprogramowania. Obejmuje czas analizy i projektowania. Czas modelowania należy odróżniać od czasu wykonywania obliczeń.

Model powinien odzwierciedlać dwa aspekty systemu – strukturę systemu oraz jego działanie (funkcjonowanie). Inaczej, model powinien odzwierciedlać aspekt statyczny i dynamiczny.

aspekt modelu

Skupienie się na szczególnych właściwościach metamodelu.

strukturalny aspekt modelu

Aspekt modelu koncentrujący się na strukturze obiektów w systemie, włączając w to ich typy, klasy, relacje, atrybuty i operacje.

architektura

Struktura organizacyjna systemu. Architektura może być rekursywnie dekomponowana na części, które współdziałają ze sobą przez wspólne interfejsy, musi też określać relacje zachodzące pomiędzy częściami oraz określać ograniczenia dotyczące ich łączenia ze sobą.

RM ODP [10] określa architekturę jako zestaw reguł definiujących strukturę systemu oraz związki pomiędzy jego częściami składowymi.

behavioralny aspekt modelu

Aspekt modelu, który skupia się na zachowaniu instancji (obiektów) w systemie, włączając w to ich metody, współdziałania i historie stanów.

zachowanie

Obserwowane efekty operacji lub zdarzenia, włączając w to ich wyniki.

Pojęcie zachowania, chociaż bardzo ważne, bywa rozumiane różnie. RM ODP [10] określa zachowanie (obiektu) jako zbiór akcji (czegoś co może się zdarzyć) wraz ze zbiorem ograniczeń dotyczącym ich wystąpień. Należy przy tym zaznaczyć, że termin 'akcja' w RM ODP obejmuje znaczenie 'akcji', a także 'zdarzenia' w sensie UML. Szczegółową analizę pojęć w UML i w RM ODP przedstawia praca [11].

Wydaje się, że najbardziej wyczerpujące pojęcie zachowania wprowadza LOTOS [15]. Po pierwsze wyróżnia się system oraz jego zewnętrznego obserwatora. Następnie określa się to, co obserwator może dostrzec obserwując system, czyli obserwowalne efekty. Przy tych ustaleniach zachowanie (procesu – odpowiednika obiektu) rozumie się jako zbiór wszystkich ciągów efektów, które są możliwe do zaobserwowania. Ogólnie, efekty które obserwator może dostrzegać to zmiany stanów systemu lub interakcje (komunikacje) systemu z jego otoczeniem. Założenie, że obserwuje się stany systemu oznacza, że ma się jakieś wyobrażenie systemu. Przyjęcie takie założenia jest właściwe na przykład podczas testowania istniejącego systemu. Natomiast w przypadku specyfikowania tworzonego dopiero systemu, bardziej odpowiednie jest założenie obserwacji interakcji systemu z otoczeniem i abstrahowanie od jakiegokolwiek wiedzy o budowie systemu.

Pojęcie zachowania w UML nie jest ani tak wyczerpujące, ani też tak precyzyjne jak w LOTOSie. Stąd pojawiają się w UML inne uzupełniające pojęcia, na przykład scenariusz, rozumiany jako 'ciąg akcji ilustrujących zachowanie'.

Zasadniczymi elementami modelu – modelami bytów z wybranej dziedziny są *klasy* i *obiekty*. Chociaż obiekty można w pewnym sensie uważać za pojęcia pierwotne, to język UML wprowadza pojęcie obiektów jako pojęcie pochodne od pojęcia klasy.

Należy pamiętać, że klasy i obiekty – jako elementy zasadnicze – są elementami składowymi wielu modeli wyrażalnych w UML, ale nie są elementami jedynymi. Obok nich występują także:

- interfejsy,
- przypadki użycia,
- grupy współdziałania,
- komponenty,
- węzły (jednostki rozmieszczenia).

Pojęcie te są omówione w następnych rozdziałach.

element modelu

Element stanowiący abstrakcję wyprowadzoną z modelowanego systemu.

klasa

Opis zbioru obiektów posiadających takie same atrybuty, operacje, metody, relacje i semantykę. Klasa może używać interfejsów do definiowania zestawu operacji oferowanych jej środowisku.

Podane określenie klasy wprowadza pewne niejednoznaczności interpretacyjne.

Ogólnie, pojęcie klasy bywa rozumiane na trzy sposoby:

- Klasa jako pewien *zbiór* (kolekcja) ustalonych elementów (obiektów). W tym sensie klasa ma ustalać konkretny zbiór obiektów.
- Klasa jako *typ*. W tym sensie klasa to pewien predykat, który jest albo nie jest spełniony dla danego elementu.
- Klasa jako *wzorzec*. W tym sensie klasa jest szablonem, według którego można generować elementy (obiekty).

Klasa w UML z pewnością nie ma pierwszego znaczenia – konkretnego zbioru elementów (nazywanego też ekstensją). Czasem to znaczenie może być rozumiane

trochę inaczej, jako uniwersum dla danych rozważań, czyli jako zbiór wszystkich potencjalnych bytów. Pozostałe dwa znaczenia są używane w UML, przy czym kontekst użycia wskazuje, o które znaczenie chodzi. Na przykład, podczas wprowadzania klas na etapie analizy używa się pojęcia klasy jako typu, natomiast na etapie programowania klasy są używane głównie jako wzorce.

Użycie klasy jako typu jest podobne do traktowania tego pojęcia w taki sam sposób jak, na przykład, w ujęciu słownika języka polskiego są rozumiane pojęcia ogólne ‘koń’, ‘zwierzę’ itp. W każdym z tych przypadków słowo ‘koń’ czy ‘zwierzę’ nie określa konkretnego konia czy konkretnego zwierzęcia, ale raczej klasę abstrakcji na określonym domyślnie uniwersum. W ścisłym rozumieniu typ jest pewnym predykatem. Pozwala to na uściślenie wielu pojęć. Oznacza to, że dany obiekt jest obiektem danej klasy A, gdy spełnia odpowiedni predykat P_A . Stąd wynika, na przykład, pojęcie podklasy. Jeżeli klasa A jest określona przez predykat P_A oraz klasa B jest określona przez predykat P_B , taki że $P_A \Rightarrow P_B$, to mówimy, że klasa A jest podklasą klasy B.

Użycie klasy jako wzorca jest podobne do odlewania pewnych wyrobów metalowych z tej samej, być może w pewnym zakresie zmienianej, formy. Ścisłym odpowiednikiem wzorca jest pewien tekst oraz algorytm, który na podstawie tego tekstu i ewentualnych dodatkowych parametrów generuje nowy tekst reprezentujący nowy obiekt.

Rozumienie klasy jako typu i jako wzorca prowadzi do różnych konsekwencji. Elementami klasy, jako typu, mogą być w zasadzie dowolne byty pod warunkiem, że posiadają pewne własności określone przez odpowiedni predykat. Natomiast elementami klasy jako wzorca są „podobne” byty różniące się co najwyżej pewnymi parametrami.

Niezależnie od tych niejednoznaczności, wątpliwości budzi określenie, że obiekty w klasie mają takie same relacje i semantykę. Można to rozumieć tylko na pewnym poziomie ogólności, na przykład – w przypadku relacji – można mówić co najwyżej o jednakowej ich sygnaturze.

sygnatura

Nazwa i parametry własności behawioralnej. Opcjonalnie sygnatura może zawierać parametr zwrotny (parametr odpowiedzi).

W dokumentach standaryzujących ODP [10] dokonuje się wyraźnego rozróżnienia pomiędzy wymienionymi trzema znaczeniami. Pojęciu zbioru odpowiada termin *class*, pojęciu wzorca – termin *template*, natomiast ostatniemu z analizowanych znaczeń odpowiada pojęcie typu – *type*. Typ jest tu rozumiany jako pewien predykat, który dla danego argumentu, w szczególności może nim być obiekt, jest albo nie jest spełniony, co pociąga stwierdzenie, że obiekt jest albo nie jest obiektem danego typu.

obiekt

Obiekt jest jednoznacznie identyfikowalnym bytem, ściśle rozgraniczonym od swego otoczenia, ukrywający własny stan i zachowanie. Stan obiektu jest reprezentowany przez jego atrybuty oraz relacje z innymi obiektami, a zachowanie jest reprezentowane przez operacje, metody i maszyny stanowe. Obiekt jest instancją klasy.

W ujęciu ontologicznym obiekt jest bytem o własnej tożsamości, tzn. jest bytem, który istnieje i jest odróżnialny od innych bytów. Odróżnialność bytów można osiągać na różne sposoby, najczęściej osiąga się to przez nadanie im unikatowych nazw. Na gruncie konkretnego języka pojęcie bytu jest uściślane przez przydanie mu specyficznych właściwości, np. przez atrybuty czy operacje.

Stan obiektu jest na ogół rozumiany jako wartościowanie jego atrybutów oraz relacji. Stan obiektu należy odróżniać od stanu maszyny stanowej, która może być związana z danym obiektem. Stany maszyny stanowej, w najprostszym przypadku, są określone jako partycja na zbiorze wszystkich stanów obiektu, czyli na zbiorze wszystkich wartościowań atrybutów i relacji obiektu. W przypadkach bardziej złożonych stan maszyny może dodatkowo ujmować historię obiektu. Warto zaznaczyć, że historię obiektu daje się również wyrazić przez dołączenie dodatkowych atrybutów (atrybutów historycznych), wówczas pojęcie stanu maszyny można nadal traktować

traktować jako partycję na zbiorze wartościowań rozszerzonego zbioru atrybutów i relacji obiektu.

Model systemu, złożony w pewien sposób z powiązanych ze sobą obiektów, ma opisywać funkcjonowanie poszczególnych składowych oraz ich współdziałanie ze sobą oraz z otoczeniem systemu. Funkcjonowanie składowych systemu jest opisywane w terminach wykonywanych akcji i aktywności, a współdziałanie przez opis wymiany komunikatów (wiadomości). UML nie definiuje jawnie otoczenia systemu, lecz skupia się na opisie współdziałania systemu z otoczeniem. Współdziałanie polega na wzajemnej wymianie informacji – przesyłaniu komunikatów – przez dobrze zdefiniowany interfejs. Akt komunikacji pomiędzy systemem a jego otoczeniem, czy też pomiędzy składowymi systemu, jest przykładem akcji.

akcja

Wykonywalna, niepodzielna (atomowa) czynność obliczeniowa, której wynikiem jest zmiana stanu systemu lub dostarczenie pewnej wartości.

Akcja w UML ma na ogół zerowy czas trwania. W UML za akcję uważa się także ciąg akcji. Jest to niezręczność definicji powodująca niepotrzebną komplikację pojęcia, chociaż wynika to z oczywistej obserwacji, że ciąg czynności o zerowym czasie wykonywania ma także zerowy czas realizacji.

Wyróżnia się następujące akcje:

- akcję wywołania operacji,
- akcję przekazania wyniku obliczeń operacji,
- akcję wysłania sygnału,
- akcję kreacji obiektu,
- akcję normalnego kasowania obiektu,
- akcję awaryjnego kasowania obiektu,
- akcję przypisania,
- akcję o interpretacji nieokreślonej w UML.

To co zdarza się w danym momencie jest określone w UML jako zdarzenie. Efektem akcji może być wykreowanie zdarzenia.

Pojęcie akcji w RM ODP jest rozumiane podobnie: jako coś, co się zdarza. Może to być interakcja, czyli przekazanie informacji pomiędzy dwoma komponentami lub akcja wewnętrzna, co skutkuje tylko zmianą stanu. Tak jak w UML, akcja w ODP jest niepodzielna.

aktywność

Podzielna (nieatomowa) czynność obliczeniowa zdefiniowana w obrębie maszyny stanów.

W odróżnieniu od akcji aktywność jest czynnością obliczeniową zajmującą na ogół niezerowy czas. Akcje mogą być elementami aktywności. Mogą w nich występować sekwencyjnie lub równolegle.

zdarzenie

Specyfikacja istotnego wystąpienia mającego przyporządkowanie w czasie i w przestrzeni.

Istotną właściwością zdarzenia jest związenie z danym momentem, a zatem jego natychmiastowość. Szczególnymi rodzajami zdarzeń wyróżnianych w UML są: odbiór sygnału, odbiór wywołania operacji, upływ zadanego okresu czasu oraz zmiana stanu (rozumiana na przykład jako spełnienie pewnego predykatu).

Warto zwrócić uwagę, że w UML nie wyróżnia się, na przykład, zdarzenia wysłania sygnału (asynchronicznego komunikatu). Należy to uznać za niedostatek języka, gdyż nie pozwala to, na przykład, na wyrażanie własności czasowych wiążących moment wysłania sygnału z momentem jego odbioru.

Zdarzeniu jest zawsze przyporządkowany moment czasu jego wystąpienia, a ponadto może mu być przyporządkowana nazwa oraz zestaw wartości.

komunikat (wiadomość)

Przekazanie informacji pomiędzy obiektami związane z oczekiwaniem, że spowoduje to pewną akcję po stronie odbierającej. Komunikatem jest wywołanie operacji i wysłanie sygnału. Odbiór komunikatu jest zwykle traktowany jako wystąpienie pewnego zdarzenia.

Specyfikacja ta określa nie tylko rodzaj komunikacji – wysłanie sygnału, wywołanie operacji, utworzenie lub skasowanie obiektu, ale także role nadawcy i odbiorcy w obrębie danego połączenia komunikacyjnego, oraz informacje przekazywane pomiędzy nadawcą a odbiorcą. W UML dopuszcza się zarówno synchroniczny jak i asynchroniczny sposób komunikacji pomiędzy nadawcą a odbiorcą.

interfejs

Deklaracja zbioru operacji, które łącznie definiują pewien zestaw usług oferowanych przez instancję (przypadku użycia, obiektu lub komponentu).

Interfejs jest ogólną charakterystyką pewnego spójnego zestawu usług reprezentowanych przez wybrane operacje danego przypadku użycia obiektu lub komponentu. Obiekt lub komponent mogą mieć wiele interfejsów.

System w danym momencie jest scharakteryzowany przez swój stan, który determinuje zachowanie się (funkcjonowanie) systemu w przyszłości. Stan systemu jest określony przez stany wszystkich jego obiektów składowych.

stan

Sytuacja w czasie życia obiektu, scharakteryzowana przez pewien warunek, wykonywaną aktywność lub oczekiwanie na pewne zdarzenie.

Sytuację obiektu w danym momencie opisuje stan jego maszyny stanowej.

Podsumowując, system (bądź jego model) może być opisywany wyłącznie z punktu widzenia zewnętrznego albo wewnętrznego obserwatora. Obserwator zewnętrzny opisuje system przez określenie jego usług oraz jego zachowania, natomiast obserwator wewnętrzny opisuje system przez określenie jego stanów i przejść pomiędzy stanami w wyniku reakcji na zachodzące zdarzenia.

3. Elementy języka

3.1. Klasyfikacja

Elementy języka UML można pogrupować w trzy kategorie:

- *elementy podstawowe*,
- *elementy wiążące*,
- *diagramy*.

Podstawowe elementy, z których buduje się model danego systemu można podzielić na następujące cztery podkategorie:

- *elementy strukturalne*, na które składają się:
 - (a) klasy,
 - (b) interfejsy,
 - (c) grupy współdziałania,
 - (d) przypadki użycia,
 - (e) klasy aktywne,
 - (f) komponenty,
 - (g) węzły,
- *elementy behawioralne*, na które składają się:
 - (a) interakcje,
 - (b) maszyny stanów,
- *elementy grupujące* – zawierające tylko pakiety,
- *elementy objaśniające* – zawierające tylko notki.

Elementy wiążące, które służą do wiązania elementów podstawowych przy tworzeniu modelu, obejmują następujące relacje:

- zależności,
- asocjacji,
- generalizacji,

- realizacji.

Diagramy służą do wizualizacji tworzonych modeli. Zasadniczą częścią każdego diagramu jest pewien graf etykietowany, z którego wierzchołkami – elementami podstawowymi – i krawędziami – elementami wiążącymi – są związane dodatkowe informacje wyrażane w postaci tekstowej. Zestaw diagramów służy do budowy modelu rozważanego systemu.

Zestaw *diagramów* do prezentacji modelu lub jego fragmentów obejmuje:

- diagramów klas (*class diagrams*),
- diagramów obiektów (*object diagrams*),
- diagramów przypadków użycia (*use case diagrams*),
- diagramów sekwencji (*sequence diagrams*),
- diagramów współdziałania (*collaboration diagrams*),
- diagramów stanów (*state diagrams*),
- diagramów aktywności (*activity diagrams*),
- diagramów komponentów (*component diagrams*),
- diagramów rozmieszczenia (*deployment diagrams*).

Diagramy klas, obiektów, przypadków użycia, komponentów i rozmieszczenia służą do wyrażania aspektów statycznych tworzonych modeli, natomiast diagramy sekwencji, współdziałania, stanów i aktywności służą do wyrażania ich aspektów dynamicznych.

Diagramy sekwencji i diagramy współdziałania noszą wspólną nazwę *diagramów interakcji*. Natomiast diagramy komponentów i diagramy rozmieszczenia noszą wspólną nazwę *diagramów implementacji*.

3.2. Elementy strukturalne

Klasy

Elementy strukturalne stanowią komponenty modeli budowanych w różnych fazach tworzenia oprogramowania. Zasadniczym pojęciem jest definiowana już

wcześniej klasa, jako opis zbioru obiektów posiadających takie same atrybuty, operacje, metody, relacje i semantykę. Na poziomie metamodelu klasa jest szczególnym rodzajem klasyfikatora. Dokładniej, pomiędzy pojęciem klasy a pojęciem klasyfikatora na metamodelu zachodzi relacja generalizacji – klasa jest metaklasą specyficzną, a klasyfikator – metaklasą ogólną. Elementy klasyfikatora nazywamy instancjami. Instancjami klasy są obiekty.

klasyfikator

Mechanizm opisujący własności behawioralne i strukturalne elementów modelowania. Klasyfikatorami są klasy, aktorzy, komponenty, typy danych, interfejsy, węzły, sygnały, podsystemy i przypadki użycia. Klasy są najbardziej ogólne spośród wszystkich klasyfikatorów, co intuicyjnie oznacza, że pozostałe klasyfikatory można rozumieć jako klasy z pewnymi ograniczeniami dotyczącymi ich treści lub użycia.

atrybut

Nazwana właściwość klasyfikatora, określająca zestaw wartości, które mogą przyjmować instancje klasyfikatora.

Pojęcie atrybutu odnosi się ogólnie do wszystkich klasyfikatorów. Klasyfikator jest pojęciem pomocniczym (z metamodelu), które obejmuje: interfejsy, klasy, typy danych i komponenty.

Atrybut ma swoją nazwę, a ponadto z atrybutem wiążą się następujące cechy (metaatrybuty):

- typ,
- widzialność (publiczny, chroniony, prywatny),
- zakres wartości (instancja typu),
- zakres własności (wartość atrybutu wspólna dla całej klasy, czy indywidualna dla obiektów klasy),
- krotność atrybutu określająca liczbę wartości atrybutu przyporządkowaną każdemu jego właścicielowi,

- sposób zmiany wartości atrybutu (brak ograniczeń, tylko odczyt, tylko dołączanie nowych wartości).

operacja

Usługa dostarczana przez obiekt, która manifestuje się przez odpowiednie zachowanie obiektu. Operacja ma sygnaturę, która określa jej dopuszczalne parametry.

Operacja ma swoją nazwę, listę parametrów formalnych i typ zwracanego wyniku obliczeń, a ponadto z operacją wiążą się następujące cechy:

- rodzaj dostępu (sekwencyjny, pośredni, równoległy),
- polimorfizm (polimorficzna, niepolimorficzna)
- zakres (operacja wspólna dla całej klasy, czy indywidualna obiektów klasy),
- widzialność (publiczna, chroniona, prywatna).

Pojęcie operacji należy odróżniać od pojęcia metody, która jest implementacją operacji. Z każdą operacją może być związana jej specyfikacja wyrażona formalnie lub nieformalnie. Może też być określony skutek jej wywołania (zmiana lub brak zmiany wartościowania atrybutów), czy jest polimorficzna (jedna czy wiele implementacji).

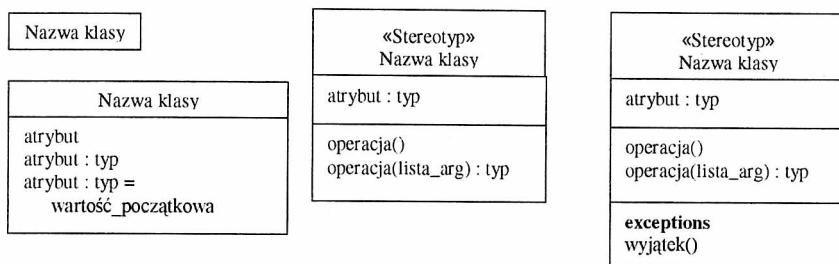
metoda

Implementacja operacji. Określa algorytm lub procedurę, która dostarcza wyniku operacji.

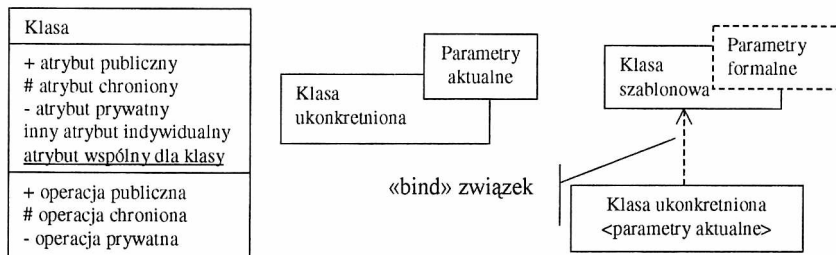
Metoda posiada cechy operacji, a ponadto może być z nią skojarzone zachowanie definiowane przez odpowiednią maszynę stanową, treść wyrażana nieformalnie lub w wybranym języku programowania lub grupa współdziałania określająca jej implementację.

Graficzną reprezentację klas przedstawia rys. 2. Kolejny rys. 3 przedstawia notację określającą dalsze szczegóły związane z klasami. Poza sekcją dla nazwy, która musi być zawsze, oraz sekcjami dla atrybutów i operacji, dopuszcza się wprowadzanie dodatkowych sekcji, np. sekcji dla definiowania wyjątków, odpowiedzialności klasy itp. Omówione klasy nazywa się klasami konkretnymi w odróżnieniu od klas szablonowych.

Klasy szablonowe bezpośrednio nie określają konkretnej klasy. Definiują one konkretne klasy dopiero po ukonkretnieniu, czyli po ustaleniu wartości odpowiednich parametrów generycznych klas szablonowych. Używa się jeszcze pojęcia klas abstrakcyjnych. Klasy abstrakcyjne nie służą do określania konkretnych obiektów, w UML pełnią one pomocniczą rolę przy definiowaniu innych klas.



Rys. 2. Podstawowa notacja klas



Rys. 3. Widzialność i zasięg atrybutów i operacji, klasy parametryzowane

relacja

Semantyczne powiązanie pomiędzy elementami modelu, na przykład: relacja asocjacji lub generalizacji.

Pojęcie powiązania odnosi się ogólnie do dowolnych klasyfikatorów.

Pojęcie semantyki, odnoszące się do klasy jest rozumiane nieformalnie. Pośrednio semantykę można określić przez ustalenie odpowiedzialności związanej z klasą. Pojęcie

odpowiedzialności – poza postacią notki – nie ma w UML bezpośredniej reprezentacji, natomiast z tego pojęcia wynikają własności dynamiczne klasy.

odpowiedzialność

Kontrakt lub zobowiązanie klasy lub innego elementu.

Odpowiedzialność jest przede wszystkim pojęciem metodycznym. Pojęcie kontraktu nie jest definiowane w UML, jest natomiast uwzględniane przez RM ODP. Kontrakt jest umową zawieraną pomiędzy serwerem a klientem (klasą a jej użytkownikami), która może zawierać:

- specyfikację ról i związanych z nimi interfejsów,
- określenie jakości usług,
- ustalenie okresów ważności umowy,
- wskazanie zachowań koniecznych, dopuszczalnych, zabronionych itp.

Ostatecznie zobowiązania i kontrakty związane z odpowiedzialnością znajdują swoje odzwierciedlenie w zachowaniu, które ma swą reprezentację w UML w postaci diagramu stanów lub diagramu aktywności związanego z daną klasą.

Klasy aktywne

Klasa aktywna to klasa, której obiekty posiadają jeden lub więcej procesów lub wątków, i przez to mogą przejawiać działania sterujące. Proces oraz wątek nie są pojęciami występującymi na poziomie metamodelu. Są użytecznymi pojęciami podczas tworzenia szczegółowego projektu oprogramowania.

klasa aktywna

Klasa której instancje są obiektami aktywnymi.

obiekt aktywny

Obiekt posiadający własny wątek sterowania i mogący inicjować sterowanie.

Klasy aktywne mają reprezentację graficzną różniącą się od reprezentacji innych klas tylko pogrubioną krawędzią bloków.

proces

Jednostka równoległości (ciąg czynności obliczeniowych) w językach programowania lub w systemach operacyjnych. Jest to również standardowy stereotyp wskazujący na sposób implementacji aktywnego obiektu jako procesu.

Ogólnie, podane określenie nie jest jedynym znaczeniem kojarzonym ze słowem proces. Proces może też oznaczać wykonywanie pewnych czynności, na przykład, wykonywanie obliczeń pewnego algorytmu, lub wytwarzanie oprogramowania.

wątek

Pojedyncza ścieżka wykonania programu lub inna reprezentacja przepływu sterowania. Także stereotyp dla implementacji aktywnego obiektu jako „lekkiego” procesu.

Wątek może być uważany za szczególny rodzaj procesu. Zasadnicza różnica pomiędzy wątkiem i procesem jest związana z ich mechanizmami synchronizacji i komunikacji. W porównaniu do procesów wątki mają te mechanizmy ograniczone, co powoduje, że wykorzystywanie wątków wymaga od ich użytkowników przestrzegania pewnych ograniczeń, zwykle chodzi o zagwarantowanie wykluczającego się dostępu do wspólnej pamięci.

Typy danych

typ danych

Opis zbioru wartości, bez wyróżnionej tożsamości. Typy danych mogą być predefiniowane (standardowe) lub definiowane przez użytkowników. Elementarnymi typami predefiniowanymi są liczby, napisy, oraz wartości czasowe. Typami definiowanymi przez użytkownika są typy wyliczeniowe.

Typ danych w UML jest rozumiany inaczej niż w językach programowania, gdzie przez typ rozumie się pewien zbiór wartości oraz pewien zbiór związanych z nim operacji. Typ danych w językach programowania ma swoją tożsamość co oznacza, że można kreować różne typy posiadające ten sam zbiór wartości i ten sam zbiór operacji. Natomiast w UML kreowanie nowego typu, którego zbiór wartości byłby identyczny ze

zbiorem wartości istniejącego już typu, na przykład typu całkowitoliczbowego, nie ma sensu, gdyż pomiędzy różnymi zbiorami całkowitoliczbowymi nie ma odróżnienia.

Pojęcie interfejsu było wyjaśnione wcześniej. Interfejs może być przyporządkowany przypadkowi użycia, klasie lub komponentowi ma swoją nazwę. Interfejsy nie muszą być rozłączne.

Graficznie interfejs przedstawia się kółkiem. Rys. 4 przedstawia klasę i pakiet powiązane relacją asocjacji z interfejsami.



Rys. 4. Przykład klasy pakietu z interfejsami

Grupy współdziałania

Pojęcie grupy współdziałania zostało wprowadzone w wersji UML 1.3.

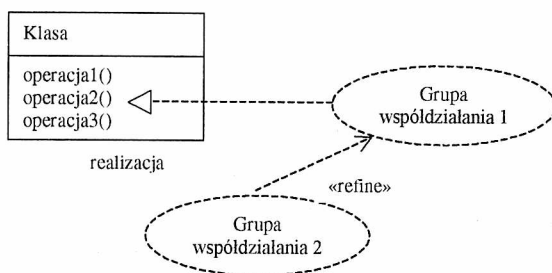
grupa współdziałania

Opis zbioru obiektów i ich powiązań, które współdziałając ze sobą w pewnym otoczeniu implementują zachowania przypadku użycia lub operacji. Grupa współdziałania ma część statyczną i dynamiczną. Część statyczna opisuje role obiektów i ich powiązań, które mogą pełnić w instancji grupy współdziałania. Część dynamiczna zawiera opis przepływu w czasie komunikatów pomiędzy obiektami w celu realizacji wspólnych obliczeń.

W poprzednich wersjach UML pojęcia tego nie było. Jest ono przydatne dla powiązania kolejnych faz wytwarzania oprogramowania. Można jawnie określać relacje pomiędzy elementem modelu fazy wcześniejszej z grupą współdziałania, która jest jej implementacją i która stanowi fragment modelu fazy późniejszej. Na przykład, można w ten sposób wskazywać na grupę współdziałania jako realizatora przypadku użycia lub operacji. Zbiór grup współdziałania implementujących operacje danej klasy można traktować jako implementację klasy.

Aspekty statyczne grupy można wyrażać wykorzystując diagramy klas lub obiektów, zaś aspekty dynamiczne – wykorzystując diagramy interakcji.

Na rys. 5 grupa współdziałania 1 jest realizacją pewnej operacji wskazanej klasy (powiązanie przez relację realizacji), zaś grupa współdziałania 2 – przez relację zależności ukonkretnioną stereotypem «refine» – jest określona jako pewne uściślenie grupy 1.



Rys. 5. Przykład wykorzystania grup współdziałania

Uogólnieniem pojęcie grupy współdziałania jest *parametryzowana grupa współdziałania*. Konkretyzacja parametryzowanej grupy wyznacza konkretną grupę współdziałania. Zwykle podstawowymi jej parametrami są klasyfikatory. Konkretyzacja polega na powiązaniu parametrów formalnych z konkretnymi klasyfikatorami.

Pojęcie to jest przydatne do określania tzw. *wzorców projektowych*. Na rys. 6 jest przedstawiony przykład parametryzowanej grupy współdziałania oraz oparty na niej wzorec.

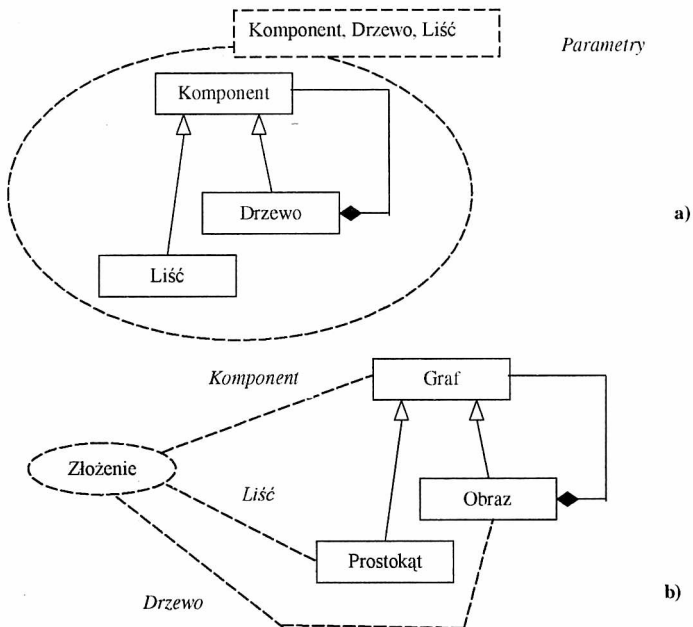
Przypadki użycia

Przeznaczeniem przypadków użycia jest opis systemu bądź jego części z punktu widzenia użytkownika. Z pragmatycznego punktu widzenia przypadki użycia mogą stanowić opis funkcjonalnych wymagań użytkownika dla mającego powstać systemu, bądź mogą stanowić opis funkcjonowania istniejącego już systemu. Użytkowników reprezentują aktorzy.

przypadek użycia (usługa)

Specyfikacja sekwencji akcji, z włączeniem sekwencji alternatywnych i sekwencji wyjątkowych, które system, podsystem lub klasa może wykonać współdziałając ze swoim otoczeniem reprezentowanym przez zewnętrznych aktorów.

Przypadek użycia jest opisem zbioru skończonych ciągów akcji – scenariuszy, które system wykonuje po ich inicjacji przez użytkownika systemu i które mają dostarczyć użytkownikowi oczekiwanego wyniku. Inaczej: przypadek opisuje pewną usługę, którą system świadczy użytkownikowi. Termin ‘przypadek użycia’ rozpowszechnił się już dosyć szeroko, chociaż właściwszym terminem byłaby właśnie ‘usługa’ lub ‘zestaw usług’, co lepiej oddaje znaczenie pojęcia. W opisie uwagę zwraca się na interakcje zachodzące pomiędzy opisywanym systemem a jego otoczeniem. Jednym z rodzajów akcji jest bowiem wywołanie operacji, które – dokonywane przez aktora zewnętrznego – jest początkiem sekwencji akcji należących do przypadku użycia.



Rys. 6. Przykład parametryzowanej grupy współdziałania a) wzorzec b) konkretyzacja

Zwykle przypadek użycia zawiera sekwencję podstawową, która opisuje ciąg akcji podczas wykonywania pewnej usługi przy założeniu pewnych typowych warunków, oraz sekwencje dodatkowe, które opisują przebieg realizacji usługi w przypadku zajścia nietypowych warunków.

W sekwencji akcji umieszcza się znaczniki rozszerzeń określające miejsce, w którym dana sekwencja może się rozgałęzić bądź rozszerzyć o inny ciąg. Ze znacznikiem może opcjonalnie być związany warunek, który określa kiedy to może nastąpić i jaki ciąg ma być wybrany do rozgałęzienia lub do wstawienia. Znaczniki można umieszczać w postaci listy poniżej nazwy przypadku użycia – rys. 6.b.

Przypadki użycia, tak jak inne klasyfikatory, mogą mieć atrybuty, operacje i metody. Mogą być opisywane przez diagramy interakcji, diagramy aktywności lub diagramy stanów. Instancją przypadku użycia jest pojedynczy ciąg akcji – pojedynczy scenariusz.

Przypadki użycia, jako klasyfikatory, mogą być w relacji z innymi klasyfikatorami – aktorami lub innymi przypadkami użycia.

aktor

Spójny zestaw ról, które pełnią użytkownicy podczas interakcji z przypadkami użycia. We współpracy z danym przypadkiem użycia aktor może pełnić tylko jedną rolę.

rola

Nazwane, specyficzne zachowanie bytu występującego w danym kontekście. Rola może być statyczna, np. rola końca asocjacji, bądź dynamiczna, np. rola współdziałania.

Aktorzy reprezentują obiekty (inne systemy lub ludzi), które są poza modelowanym systemem, czyli należą do otoczenia systemu reprezentowanego przez przypadki użycia.

Na rys. 6.a pomiędzy aktorem a przypadkami użycia (przypadki 1 oraz 3) zachodzą asocjacje. Pomiędzy przypadkiem 1 a 2 zachodzi relacja zależności «extend», zaś

między przypadkiem 1 a przypadkiem 3 – relacja zależności «include». Natomiast pomiędzy przypadkami 4 i 2 zachodzi relacja generalizacji.

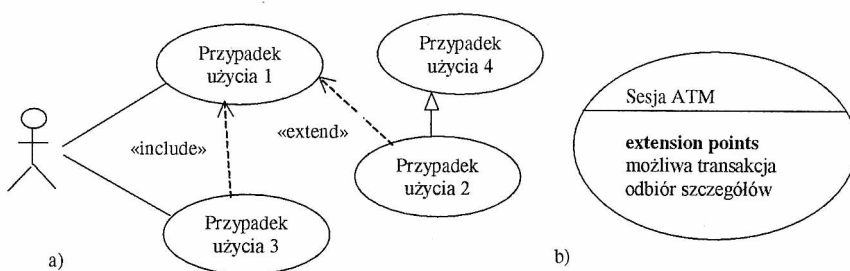
Asocjacje wiążące aktora z przypadkami użycia (1 oraz 3) określają z których usług aktor może korzystać.

Relacja zależności «extend» oznacza, że przypadek użycia 2 zawiera sekwencje akcji, które – w przypadku spełnienia odpowiednich warunków – mogą być wstawiane do sekwencji akcji należących do przypadku użycia 1 w miejscach określonych przez odpowiednie znaczniki.

Relacja zależności «include» oznacza, że sekwencje akcji należące do przypadku użycia 1 są wstawiane w miejsca wskazane przez odpowiednie znaczniki do sekwencji akcji przypadku użycia 3.

Relacje zależności «extend» oraz «include» są tzw. stereotypami. Pojęcie stereotypu jest wyjaśniane dalej, intuicyjnie rola stereotypu polega na uściśleniu znaczenia pewnego elementu języka, tu dotyczy to relacji zależności.

Relacja generalizacji oznacza, że przypadek specjalizowany (przypadek 2) może definiować dodatkowe atrybuty, operacje, sekwencje akcji, a także może wstawiać w dowolne miejsca sekwencji należących do przypadku bazowego (przypadek 4) dodatkowe ciągi akcji.



Rys. 7. Przykłady przypadków użycia

Komponenty i węzły

Komponenty oraz węzły, reprezentując oprogramowanie i sprzęt, służą do budowy

modeli implementacyjnych.

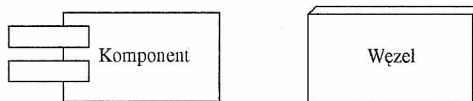
komponent

Fizyczna, przemieszczalna część systemu, stanowiąca moduł realizujący pewien zbiór interfejsów. Komponent reprezentuje fizyczną część implementacji systemu, włączając w to moduły programowe (źródłowe, binarne lub wykonywalne) lub ich odpowiedniki, na przykład skrypty lub pliki z komendami.

węzeł

Węzeł jest fizycznie funkcjonującym obiektem reprezentującym zasoby obliczeniowe, dysponujące co najmniej pamięcią a często również jednostkami przetwarzającymi. Działające obiekty i komponenty mogą być ulokowane w węzłach.

Reprezentację graficzną komponentów i węzłów przedstawia rys. 8.



Rys. 8. Notacja dla komponentów i węzłów

3.3. Elementy behawioralne

Elementy behawioralne służą do opisu zachowań. Można takiego opisu dokonywać na dwa sposoby: przez określanie pewnych ciągów wiadomości (komunikatów) wymienianych pomiędzy współdziałającymi elementami bądź przez określanie ciągów stanów, przez które przechodzą wybrane elementy. Interakcje służą do tworzenia pierwszego rodzaju, zaś maszyny stanowe do drugiego rodzaju opisów.

interakcja

Specyfikacja przesyłania wiadomości pomiędzy obiektami lub innymi instancjami w celu wykonania konkretnego zadania. Interakcja jest definiowana w kontekście grupy współdziałania.

Obiekty oraz inne instancje należące do grupy współdziałania komunikują się ze sobą, w celu realizacji wspólnego zadania. Komunikacja polega na przesyłaniu wiadomości, którymi mogą być sygnały lub wywołania operacji. Odbiór wiadomości jest pewnym zdarzeniem, zatem interakcja określa pewną kolejności zachodzenia zdarzeń. Pośrednio, do interakcji zalicza się również inne zdarzenia, którymi są zdarzenia czasowe, określające upływ pewnych chwil, np. momentów przeterminowania, oraz zdarzenia określające zajście pewnych warunków, np. spełnienie zadanego predykatu w rezultacie wewnętrznych aktywności obiektu.

sygnał

Specyfikacja asynchronicznej komunikacji pomiędzy obiektami. Sygnał może mieć parametry wyrażone jako jego atrybuty.

Sygnał jest jawnie nazwanym klasyfikatorem. Instancja komunikatu reprezentuje nazwany zestaw wartości (komunikat) przekazywany pomiędzy obiektami. Przesłanie sygnału następuje w wyniku wywołania operacji sygnału przez dowolny obiekt. Sygnał może być kierowany do dowolnego zbioru obiektów-odbiorców. W zamierzeniu sygnał jest rozgłaszany do wielu odbiorców. Ponieważ jednak jest wysłany w trybie asynchronicznym oznacza to, że może dotrzeć do odbiorców w różnych chwilach czasu. Pomiędzy sygnałami może zachodzić relacja generalizacji. Klasa lub interfejs mogą zawierać sygnał w swych deklaracjach, wskazując w ten sposób

Interakcje są skojarzone z przepływem wiadomości w obrębie konkretnej grupy współdziałania. Można je opisywać za pomocą diagramów sekwencji i diagramów współdziałania.

maszyna stanowa

Specyfikacja sekwencji stanów, przez które podczas swego istnienia przechodzi obiekt lub grupa współdziałania. Przejścia pomiędzy stanami są reakcją na zachodzące zdarzenia, przejściom tym mogą towarzyszyć akcje podejmowane przez obiekt lub grupę współdziałania. Maszyna stanowa może opisywać

zachowanie klasy, grupy współdziałania, metody lub może opisywać zachowanie konkretnej instancji.

Maszyna stanowa jest uogólnieniem automatu skończonego. Uogólnienie polega na wprowadzeniu wartościowań ustalonego zbioru atrybutów. Oznacza to, że przechodzeniu pomiędzy stanami może towarzyszyć zmiana wartościowań atrybutów.

3.4. Elementy grupujące i objaśniające

Jedyną konstrukcją służącą do grupowania różnego rodzaju elementów są pakiety. Dzięki pakietom jest możliwa dekompozycja tworzonych modeli i panowanie nad problemami związanymi ze złożonością projektu wytwarzania oprogramowania. Zasadniczym przeznaczeniem pakietu jest użycie go do definiowania systemu i podsystemów. Zarówno w obrębie systemu jak i podsystemu może być wiele modeli, stanowiących ich wyobrażenie z różnych punktów widzenia (perspektyw).

pakiet

Mechanizm ogólnego przeznaczenia służący grupowaniu elementów. Pakiety mogą być zagnieżdżane w innych pakietach. System może być reprezentowany jako pakiet najwyższego poziomu, w którym jest zawarte wszystko, co należy do systemu.

Pakiet wprowadza pewną przestrzeń nazw dla zagnieżdżonych w nim elementów, tzn. ich nazwy, w obrębie pakietu, są jednoznacznie rozróżniane.

Fakt, że pakiet zawiera pewne elementy formalnie oznacza zachodzenie relacji silnej agregacji, w której pakiet pełni rolę agregatu. Z pakietem są związane zasady uwidoczniania (*private*, *protected*, *public*) i udostępniania (*export*, *import*) jego elementów. Pakiety mogą pozostawać ze sobą w relacji generalizacji.

Wyróżnia się tylko jeden rodzaj elementów objaśniających – tzw. *notki*. Służą one różnym celom: objaśnianiu, komentowaniu, opisywaniu, uwagom itp. Mogą one być wiązane z dowolnymi elementami tworzonych modeli.



Rys. 9. Notacja dla pakietów i notek

3.5. Relacje

Elementy modelu mogą być w różny sposób powiązane ze sobą – mogą pomiędzy nimi zachodzić pewne relacje.

relacja

Semantyczne połączenia pomiędzy elementami modelu, na przykład: relacja asocjacji lub generalizacji.

Pojęcie relacji jest określone bardzo ogólnie – można rozumieć je jako dowolną relację, w sensie mnogościowym, pomiędzy pewnymi zbiorami elementów. Wyróżnia się cztery specyficzne relacje o ściślej określonym znaczeniu. Są to relacje:

- zależności,
- generalizacji,
- asocjacji,
- realizacji.

Relacja zależności jest określona bardzo ogólnie. W zasadzie relacja ta powinna być wykorzystywana z podaniem konkretnego znaczenia. Znaczenie takie można nadać relacji przez jej powiązanie z odpowiednim stereotypem. Wśród relacji zależności wyróżnia się, m.in., relację uściślenia, śladowalności.

zależność

Relacja pomiędzy dwoma elementami modelowania polegająca na tym, że zmiana w jednym – niezależnym elemencie, pociąga zmianę w drugim – zależnym elemencie.

Relacja uściślenia ma służyć do wyrażania powiązań pomiędzy elementami modelu powstającego, m.in., w fazie projektowania i implementacji oprogramowania.

uściślenie

Relacja wskazująca na pełniejszą specyfikację czegoś, co było wcześniej wyspecyfikowane na pewnym poziomie szczegółowości. Na przykład, klasa zdefiniowana w fazie projektu jest uściśleniem odpowiedniej klasy zdefiniowanej w fazie analizy.

Relacja śladowalności ma wskazywać na powiązanie elementów w procesie projektowym. Może, na przykład, dla pewnej klasy wskazywać na inne klasy które były brane pod uwagę przy jej definiowaniu.

Inny rodzaj relacji określa generalizacja:

generalizacja

Relacja pomiędzy elementem ogólnym a specyficznym. Element specyficzny jest całkowicie zgodny z elementem ogólnym i zawiera dodatkową informację. Egzemplarz elementu specyficznego może być użyty wszędzie tam, gdzie dopuszcza się egzemplarz elementu ogólnego.

Relacja generalizacji określa powiązanie pomiędzy dwoma elementami, w szczególności pomiędzy klasami – klasą ogólną i klasą specyficzną. Obiekty klasy specyficznej dziedziczą własności strukturalne i behawioralne – atrybuty i operacje – obiektów klasy ogólnej. Relacja generalizacji jest głównie wykorzystywana na etapie tworzenia modelu. Pozwala ona na usystematyzowanie tworzenia modelu metodą zstępującą, a z drugiej strony pozwala na wykorzystywanie wcześniej zbudowanych modeli. Ten ostatni wzgląd wiąże się z postulatem zastępowalności sformułowanym przez Liskov (*Liskov Substitution Principle*):

Egzemplarz nadklasy musi być całkowicie zastępowalny odpowiednim egzemplarzem podklasy.

Spełnienie tego postulatu polegające na zdefiniowaniu elementu specyficznego tylko przez dołączenie dodatkowych atrybutów lub operacji jest oczywiste, natomiast

jego spełnienie w odniesieniu do zachowań zastępowanych obiektów jest trudne. Rozszerzenie zachowania obiektu należy wiązać z pewnym rozszerzeniem maszyny stanowej, która reprezentuje jego zachowanie. Okazuje się, że nasuwające się możliwości, np. dowolne dołączenie nowego stanu lub nowego łuku do maszyny stanowej, nie gwarantują zachowania postulatu zastępowalności.

Asocjacja jest relacją, która może zachodzić pomiędzy dwoma lub więcej klasami – albo ogólniej – pomiędzy klasyfikatorami. Zwykle tę relację utożsamia się z możliwością nawigacji (przechodzenia) pomiędzy elementami asocjacji, tzn. pomiędzy obiektami powiązаныmi asocjacją. Relacja służy do wyznaczenia struktury między składowymi obiektami modelu. Może być relacją binarną, ternarną lub wyższego rzędu.

asocjacja

Semantyczna relacja pomiędzy dwoma bądź więcej klasyfikatorami, które ustanawiają połączenia pomiędzy ich instancjami.

Relację asocjacji nazywa się także związkiem, a elementy relacji asocjacji nazywa się powiązaniami. Asocjacji na diagramach klas może, poza nazwą, towarzyszyć informacja o liczności i rolach jej końców. Rys. 10. przedstawia asocjację o nazwie *zatrudnienie* wiążącą klasy o nazwach *Instytucja* i *Osoba*. Czarny trójkąt przed nazwą asocjacji wskazuje na uporządkowanie argumentów tej relacji. Zakładając, że klasy traktujemy jako pewne zbiory, relację tę można wyrazić w następującej postaci: $zatrudnienie \subseteq Osoba \times Instytucja$. Wtedy też role można traktować jako funkcje:

$$pracownik(inst) = \{prac \in Osoba \mid \langle prac, inst \rangle \in zatrudnienie\}$$

$$pracodawca(prac) = \{inst \in Instytucja \mid \langle prac, inst \rangle \in zatrudnienie\}$$

Niech $card(A)$ oznacza licznosc elementów zbioru A . Wtedy licznosc końca asocjacji związanej z rolą *pracownik* oznacza, że

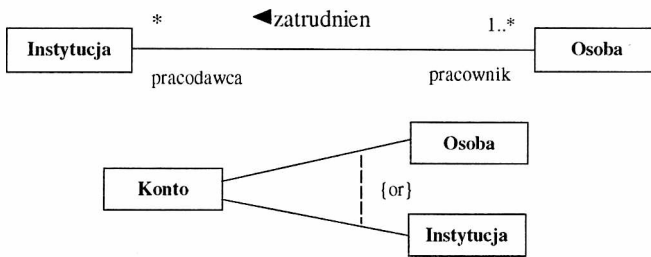
$$1 \leq card(pracownik(inst)) < \infty,$$

zaś licznosc końca związanej z rolą *pracodawca* oznacza, że

$$card(pracodawca(prac)) < \infty.$$

Nazwy asocjacji oraz nazwy ról są opcjonalne. W przypadku braku nazwy roli jest ona

przyjmowana domyślnie jako nazwa klasy na tym samym końcu powiązania. Natomiast brak podanej liczności oznacza domyślnie licznosc równą 1. Inne informacje dotyczące asocjacji można podawać w postaci ograniczeń. Przykład ograniczenia jest podany na rys. 10. Ograniczenie to wskazuje na to, że dany obiekt klasy *Konto* może być w asocjacji albo z obiektem klasy *Osoba* albo z obiektem klasy *Instytucja*.



Rys. 10. Przykłady oznaczeń związanych z asocjacją

Szczególnymi rodzajami asocjacji są agregacja oraz silna agregacja. Intencją jest tu powiązanie ze sobą we wspólną całość instancji należących do różnych klasyfikatorów.

agregacja

Rodzaj binarnej asocjacji pomiędzy klasyfikatorami, która określa relację „całość-część” pomiędzy agregatem (całością) a jego częściami.

Oznacza to, że egzemplarz należący do klasyfikatora reprezentującego “całość” zawiera – jako swoje komponenty – elementy należące do klasyfikatora reprezentującego “część”. W przypadku klas, wartościami atrybutów obiektu zagregowanego mogą być obiekty należące do klas reprezentujących „części”. Relacja agregacji jest tranzytywna i antysymetryczna. Antysymetryczne musi być też złożenie relacji agregacji.

Szczególnym rodzajem agregacji jest silna agregacja.

silna agregacja

Relacja agregacji, w której elementy składowe należą tylko do jednego elementu macierzystego, a okres ich życia zawiera się w okresie życia elementu macierzystego.

Elementy składowe mogą być kreowane po utworzeniu elementu macierzystego. Raz utworzone istnieją i są kasowane wraz ze swoim elementem macierzystym. Mogą też być kasowane przed momentem kasowania elementu macierzystego.

Agregacja może być rekursywna.

klasa złożona

Klasa będąca w relacji w relacji silnej agregacji z jedną lub więcej klasami.

obiekt złożony

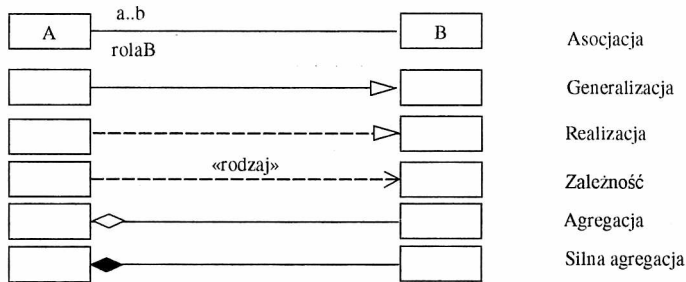
Obiekt reprezentujący “całość”, którego elementami są inne obiekty reprezentujące “części”.

realizacja

Semantyczny relacja pomiędzy dwoma klasyfikatorami, w którym jeden z klasyfikatorów specyfikuje kontrakt gwarantowany przez drugi klasyfikator.

Relacja realizacji może być, z jednej strony, uważana za pewnego rodzaju relację zależności, z innej zaś strony przypomina generalizację. Z uwagi na te skojarzenia jest ona reprezentowana przez strzałkę rysowaną przerywaną linią i z trójkątnym grotem.

Zestawienie graficznych symboli relacji przedstawia rys. 11.



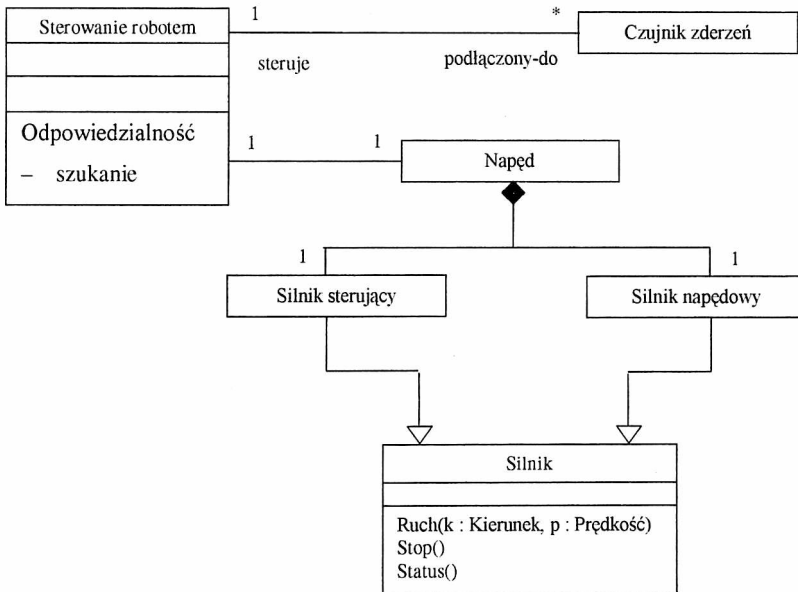
Rys. 11. Symbole graficzne relacji

4. Diagramy klas i diagramy obiektów

diagram klas

Diagram zawierający kolekcję deklaratywnych (statycznych) elementów modelu, takich jak klasy (i ich zawartość), typy, interfejsy oraz grupy współdziałania wraz z relacjami zależności, generalizacji, asocjacji.

Poza wymienionymi elementami, na diagramach klas mogą znajdować się jeszcze inne oznaczenia, np. liczności relacji. Diagramy klas służą do wyrażania statycznego aspektu budowanych modeli. Diagramy takie mogą powstawać w różnych sytuacjach projektowych, np. podczas tworzenia słownika pojęć, modelowania grup współdziałania, modelowania logicznego schematu baz danych. Behawioralne aspekty diagramów klas mogą przedstawiać związane z nimi diagramy maszyn stanów i diagramy interakcji.



Rys. 12. Przykład diagramu klas

Uwaga

Diagramy klas odgrywają w UML rolę szczególną. Z jednej strony są one elementem języka, czyli należą do metamodelu, z drugiej strony diagramy klas, a dokładniej diagramy metaklas, są elementami opisu języka. Klasy są instancjami metaklas.

diagram obiektów

Diagram zawierający obiekty i zachodzące pomiędzy nimi relacje w danej chwili czasu. Może być uważany za szczególny przypadek diagramu klas lub diagramu współdziałania. Diagram obiektów odnosi się do określonej chwili istnienia systemu. Natomiast diagram klas nie odnosi się do konkretnej chwili istnienia systemu, lecz określa dopuszczalne relacje jakie mogą zajść w dowolnej chwili istnienia systemu.

Asocjacje na diagramach klas mogą na swych końcach zawierać dwa rodzaje informacji. Pierwsza określa tzw. licznosc, czyli liczbę elementów (obiektów) klas, które mogą być powiązane ze sobą. Druga nazywa odpowiednie role asocjacji.

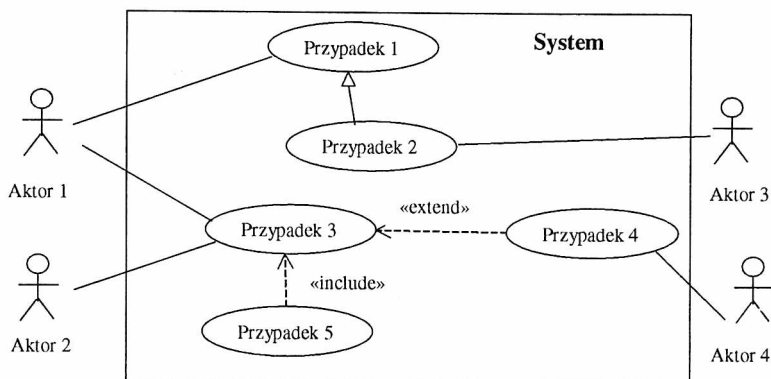
5. Diagramy przypadków użycia

diagram przypadków użycia (usług)

Diagram przedstawiający relacje pomiędzy przypadkami użycia (usługami) a aktorami w obrębie systemu.

Diagram przypadków użycia (rys. 13) przedstawia tylko strukturę przypadków użycia, przez pokazanie łączących ich relacji oraz relacji pomiędzy przypadkami użycia a aktorami, którzy reprezentują użytkowników systemu. Przypomnijmy, że przypadek użycia jest opisem zbioru skończonych ciągów interakcji – scenariuszy, które system wykonuje po ich inicjacji przez użytkownika systemu, i które mają dostarczyć użytkownikowi oczekiwanego wyniku. Zatem znaczenie poszczególnych przypadków użycia jest opisywane za pomocą diagramów interakcji (pojedyncze scenariusze) oraz

diagramów maszyn stanów (zbiór wszystkich scenariuszy). Przypadki użycia należące do jednego diagramu można zamykać w pakiety.



Rys. 13. Diagram przypadków użycia

Pomiędzy przypadkami użycia mogą zachodzić relacje generalizacji, np. przypadek 5 jest specjalizacją przypadku 3, oraz relacje zależności określone przez standardowe stereotypy «include» i «extend».

Aktor jest także standardowym stereotypem stanowiącym specjalizację klasy. Reprezentuje on spójny zestaw ról użytkownika systemu, przy czym podczas pracy z danym przypadkiem użycia, aktor może wystąpić tylko w jednej roli.

Asocjacje pomiędzy aktorami a stereotypami określają możliwości komunikacji. Relacja generalizacji pomiędzy przypadkami użycia oraz stereotypy «extend» oraz «include» były wyjaśnione wcześniej.

Stereotyp «extend» oznacza, że przypadek użycia, od którego prowadzi strzałka (przypadek 4) rozszerza zachowanie reprezentowane przez przypadek użycia, do którego strzałka wchodzi (przypadek 3). Scenariusz należący do przypadku 3 może posiadać znacznik, który – w przypadku spełnienia odpowiednich warunków – wskazuje na inny dalszy przebieg scenariusza. Alternatywne przebiegi, odpowiednio znakowane, są zgromadzone w przypadku 4. Jeżeli podczas realizacji pewnego scenariusza z

przypadku 3, w miejscu znacznika, jest spełniony związany z nim warunek, wówczas realizowany jest odpowiedni scenariusz z przypadku 4, a po jego zakończeniu następuje powrót do miejsca znakowania w przypadku 3.

Stereotyp «include» oznacza, że przypadek 3 w jawnie wskazanych miejscach swoich scenariuszy włącza odpowiednie scenariusze należące do przypadku 5.

6. Diagramy interakcji

diagram sekwencji

Diagram przedstawiający, uporządkowane w kolejności czasowej, interakcje obiektów. W szczególności przedstawia obiekty uczestniczące w tych interakcjach i kolejność wymienianych wiadomości.

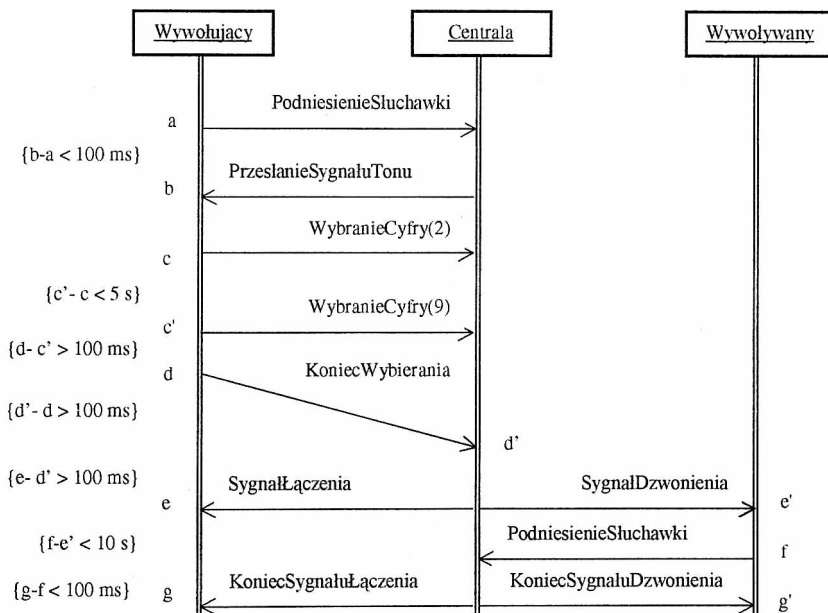
diagram współdziałania

Diagram pokazujący interakcje pomiędzy połączonymi instancjami (obiektami lub komponentami). Inaczej niż diagram sekwencji, diagram współdziałania pokazuje relacje pomiędzy instancjami. Diagram nie wyróżnia osi czasu, dlatego sekwencje wymienianych wiadomości określa się przez etykietowanie wiadomości kolejnymi liczbami. Etykietowanie takie musi przebiegać oddzielnie w ramach równoległych wątków.

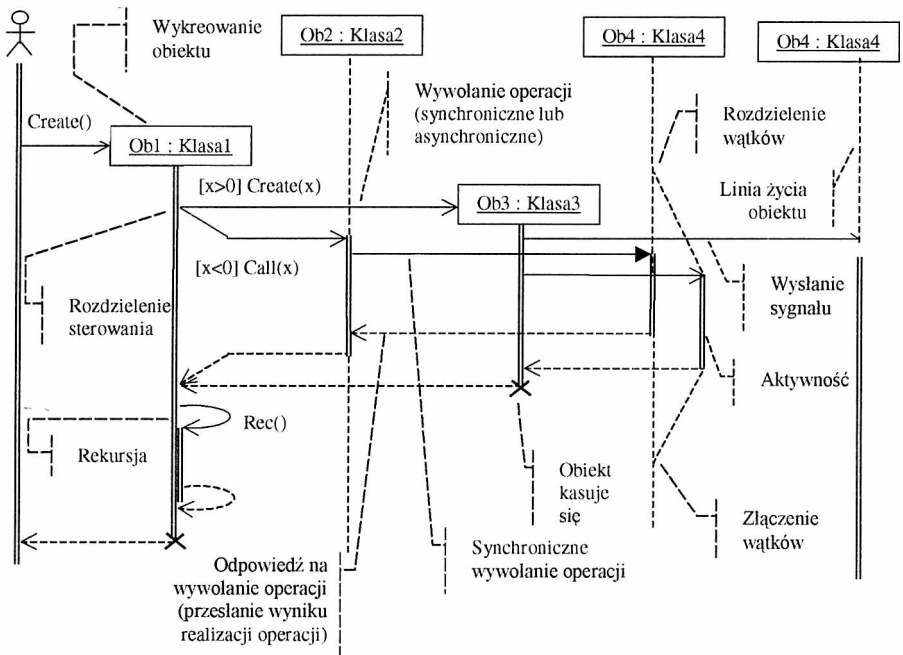
Inaczej niż diagramy współdziałania, diagram sekwencji przedstawia uporządkowanie czasowe, lecz nie przedstawia relacji pomiędzy obiektami. Diagram sekwencji może istnieć w formie ogólnej (generycznej), opisującej różne możliwe scenariusze, oraz w formie konkretnej, opisującej konkretny scenariusz. Postać generyczna różni się od postaci konkretnej tym, że posiada dodatkowe napisy, często w postaci pseudokodu, które odnoszą się do fragmentów scenariusza i określają w jakich warunkach i ile razy mogą być te fragmenty powtarzane.

Diagramy sekwencji i diagramy współdziałania przedstawiają, w różny sposób, podobną informację.

Z każdym obiektem jest związana tzw. linia życia rysowana w postaci pionowej przerywanej linii. Początek linii odpowiada momentowi wykreowania obiektu, a jej koniec, dodatkowo zaznaczony symbolem X, odpowiada momentowi skasowania obiektu. Podwójna linia oznacza okres aktywności obiektu. Przesyłane komunikaty – wywołania operacji obiektów – są przedstawiane strzałkami. Początek i koniec strzałki są oznakowane symbolami literowymi, które reprezentują zdarzenia. Ogólnie, zdarzenie wysłania komunikatu i zdarzenie odbioru nie muszą nastąpić w tej samej chwili. Jeżeli czasy wystąpienia zdarzeń mają spełniać pewne ograniczenia, to można je wyrażać w postaci formuł zapisywanych w nawiasach klamrowych, obok linii życia obiektu. Przykłady diagramów sekwencji przedstawiają rys. 14 i rys. 15.



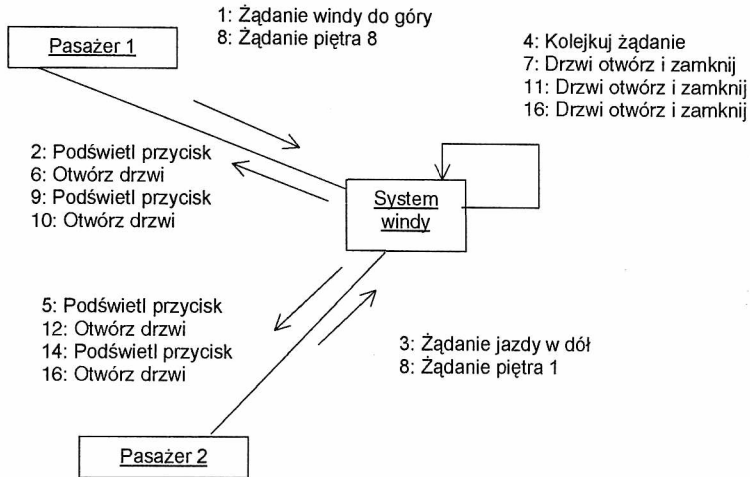
Rys. 14. Przykład diagramu sekwencji



Rys. 15. Diagram sekwencji – przykład

Są na nim pokazane kreowanie i kasowanie obiektów, rekursywne wywołanie przez obiekt swoich operacji, rozdzielanie i łączenie wątków, warunkowe przekazywanie sterowania. Strzałki ciągle reprezentują wywołania operacji, a strzałki przerywane reprezentują przesyłanie odpowiadających im wyników wyliczeń.

Na rys. 16 pokazano prosty przykład diagramu współdziałania. Jest to diagram obiektów etykietowany strzałkami przedstawiającymi przepływ komunikatów. Scenariusz reprezentowany przez diagram jest ciągiem kolejno numerowanych komunikatów.



Rys. 16. Przykład diagramu współdziałania

7. Diagramy stanów

diagram stanów

Diagram przedstawiający maszynę stanową.

maszyna stanowa

Specyfikacja sekwencji stanów, przez które podczas swego istnienia przechodzi obiekt lub grupa współdziałania. Przejścia pomiędzy stanami są reakcją na zachodzące zdarzenia, przejściom tym mogą towarzyszyć akcje podejmowane przez obiekt lub grupę współdziałania. Maszyna stanowa może opisywać zachowanie klasy, grupy współdziałania, metody lub może opisywać zachowanie konkretnej instancji.

Przypomnijmy, że zdarzeniami wyróżnianymi w UML są: odbiór wywołania operacji, odbiór sygnału, zdarzenie czasowe, zdarzenie spełnienia warunku. Dodatkowo wyróżnia się zdarzenia kreacji i kasowania obiektu. Zdarzenia te można traktować jako

specyficzny rodzaj wywołania operacji kierowanej do klasy. Należy też zwrócić uwagę na to, że nie wyróżnia się jako zdarzenia wysłania sygnału. Ponieważ sygnały są przekazywane w trybie asynchronicznym, moment wysłania i moment odbioru sygnału nie muszą być takie same. Podobnie dotyczy to wywołani operacji w trybie asynchronicznym.

Diagramy stanów są reprezentowane w postaci grafu. Ich elementami składowymi są wierzchołki, reprezentujące stany, i łączące je łuki, reprezentujące przejścia. Stany mogą być zagnieżdżane. Wyróżnia się zagłębienie stanów sekwencyjnych i stanów równoległych. Oba rodzaje zagłębienia stanów pochodzą od map stanów Harela i stanowią istotny mechanizm redukcji liczby stanów przy opisie zachowania obiektu opartego na wykorzystaniu maszyn stanowych.

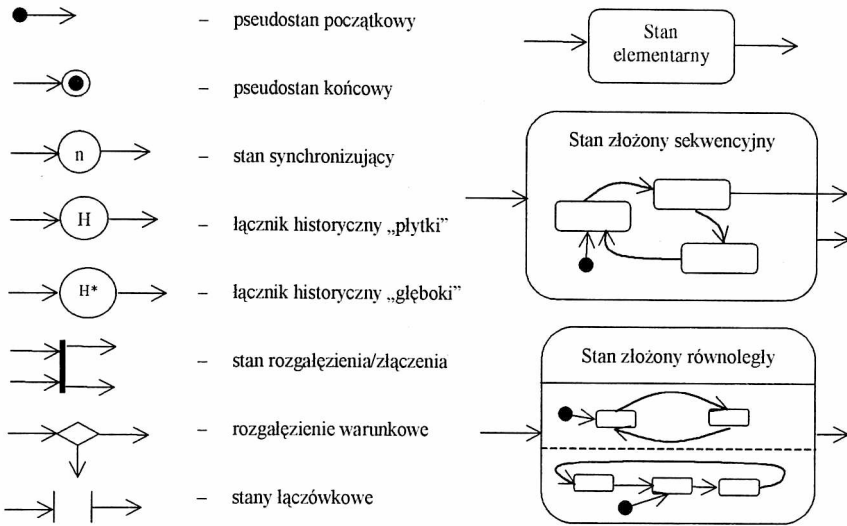
Wyróżnia się następujące rodzaje wierzchołków:

- pseudo-stany, wśród których są stany początkowe, łączniki historyczne płytkie i głębokie, stany równoległego rozgałęzienia, złączenia i synchronizacji oraz stany rozgałęzienia warunkowego;
- stany synchronizujące;
- stany właściwe, które mogą być:
 - stanami złożonymi, tzn. stanami, w których są zagnieżdżone inne stany; gdy jego podstanami są stany równoległe, inaczej regiony, to jest on nazywany stanem równoległym;
 - stanami elementarnymi, tzn. stanami nie zawierającymi innych stanów,
 - stany końcowe;
- opcjonalnie, tylko w przejściowym okresie wyznaczania mapy stanów, używa się tzw. stanów łączówkowych.

Wymienione rodzaje stanów zestawiono na rys. 17.

Pseudostany pełnią rolę pomocniczą. Niektórych z nich można by nie używać kosztem skomplikowania diagramu stanów. Pseudostan początkowy wskazuje tylko na właściwy stan początkowy. Rola łączników historycznych uwidacznia się podczas

powtórznego przechodzenia do wskazywanych przez nie stanów właściwych.



Rys. 17. Notacja graficzna stanów

Równoległe rozgałęzienie i złączenie oraz stan synchronizujący pozwalają na tworzenie, łączenie i synchronizację podstanów (regionów) w obrębie stanów złożonych.

Stan łączówkowy reprezentuje inną maszynę stanową. Wejście do tego stanu oznacza rozpoczęcie zachowania zgodnego z tą maszyną. Po zakończeniu jej działania następuje wyjście ze stanu łączówkowego.

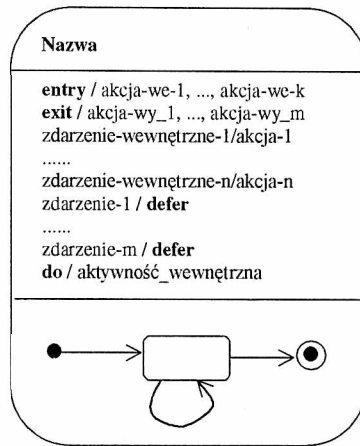
Wierzchołki i łuki mogą być etykietowane – są z nimi związane dodatkowe informacje. Z wierzchołkami jest związana informacja o akcjach jakie można wykonywać podczas przebywania w danym stanie, zaś z łukami – informacja o warunkach i zdarzeniach jakie muszą wystąpić, aby nastąpiło przejście pomiędzy wierzchołkami, a także informacja o dodatkowych akcjach, które mogą wystąpić podczas takich przejść.

Poza nazwą i ewentualnie stanami zagnieżdżonymi, z danym stanem właściwym wiążą się:

- lista akcji wejściowych, wykonywanych bezpośrednio po przejściu do danego stanu,
- aktywność wewnętrzna, wykonywana podczas przebywania danym stanie, przy czym rozpoczyna się ją po wykonaniu akcji wejściowych; jeżeli aktywność zakończy się, to może to spowodować przejście do innego stanu, pod warunkiem, że istnieje odpowiednie przejście wyzwalane przez zdarzenie reprezentujące zakończenie aktywności; jeżeli nastąpi zdarzenie wyprowadzające z danego stanu przez zakończeniem aktywności, to jest ona przerywana,
- lista akcji wyjściowych, wykonywane bezpośrednio przy wychodzeniu z danego stanu, przy czym rozpoczyna się je dopiero po zakończeniu aktywności aktualnie wykonywanych w obrębie stanu,
- lista zdarzeń odroczonej – zdarzeń, które są rejestrowane podczas przebywania danym stanie, ale ich obsługa (reakcja na te zdarzenia) jest odkładana, aż do momentu przejścia do następnego stanu, w których nie są już dalej odraczane,
- lista zdarzeń wewnętrznych, których zajście nie powoduje wyjścia z danego stanu; zdarzeniom wewnętrznym nie towarzyszy więc wykonywanie akcji wyjściowych i wejściowych; lista zdarzeń wewnętrznych związana z danym stanem odnosi się również do stanów zagnieżdżonych w tym stanie.

Stan właściwy może mieć reprezentację graficzną pokazaną na rys. 18.

Stan synchronizujący może być etykietowany liczbą. Stan synchronizujący zlicza różnicę pomiędzy liczbą wejść a liczbą wyjść ze stanu. W początkowej konfiguracji maszyny (p. dalej) licznik ten jest równy zero. Każde wejście do stanu zwiększa wartość licznika o jeden pod warunkiem, że nie przekroczy to liczby, którą jest etykietowany. Każde wyjście ze stanu zmniejsza wartość licznika o jeden pod warunkiem, że jest ona większa od zera.



Rys. 18. Pełna postać opisu stanu właściwego

Łącznik rozgałęziający jest etykietowany formułą, a wyjścia są etykietowane wartościami logicznymi. W momencie wejścia do łącznika oblicza się wartość formuły i jest wskazywane odpowiednie wyjście

Zdarzenie jest obserwowalnym wystąpieniem w czasie (zdarzenia są natychmiastowe). Wyróżnia się cztery rodzaje zdarzeń:

- odbiór sygnału (wiadomości wysyłanej w trybie asynchronicznym),
- odbiór wywołania operacji (wiadomości wysyłanej w trybie synchronicznym),
- zmiany stanu (zmiany wartości pewnego predykatu z wartości *false* na *true*),
- upływ określonego momentu przeterminowania (wyznaczonego względnie lub bezwzględnie).

Zdarzenia odbioru sygnału i wywołania operacji mogą mieć parametry, służące do przekazywania danych.

Z danym łukiem wiążą się zawsze:

- wierzchołek źródłowy, z którego łuk wychodzi,
- wierzchołek docelowy, do którego łuk prowadzi.

Ponadto, z łukiem mogą być powiązane dodatkowe informacje:

- zdarzenie wyzwalające, które inicjuje przejście pomiędzy wierzchołkami (w odróżnieniu od klasycznych map stanów dopuszcza się tylko jedno zdarzenie wyzwalające),
- dozór logiczny, określony na zbiorze wartości atrybutów obiektów należących do modelu systemu, którego prawdziwość warunkuje przejście,
- opcjonalna akcja, która jest wykonywana, gdy następuje przejście,

albo

- zdarzenie czasowe oznaczające upływ czasu określonego przez pewne wyrażenie.

Brak dodatkowych informacji oznacza, że przejście z wykorzystaniem danego łuku następuje autonomicznie, po zakończeniu aktywności wewnątrz stanu, z którego łuk wychodzi.

Etykieta łuku – przejście – może przyjąć jedną podanych niżej postaci:

Nazwa-zdarzenia(Lista-parametrów) [Dozór] / Lista-akcji

after(*Wyrażenie-czasowe*) [*Dozór*] / *Lista-akcji*

when(*Warunek-czasowy*) [*Dozór*] / *Lista-akcji*

gdzie poszczególne elementy *Nazwa-zdarzenia*, *Lista-parametrów*, *Dozór*, *Lista-akcji*, oczywiście w odpowiednich zestawach, są elementami opcjonalnymi. *Wyrażenie-czasowe* określa długość odcinka czasu, np. 10 sek. *Warunek-czasowy* jest wyrażeniem, którego wartość logiczna zależy od bieżącego czasu, np. data = 99.12.31. Dwie ostatnie formy etykiety łuku określają zdarzenia czasowe, nazywane też zdarzeniami przeterminowaniami.

W ustalonej chwili czasu mapa stanów, albo – wygodniej – maszyna stanów znajduje się w pewnej konfiguracji stanów. Konfiguracja stanów określa pewien podzbiór wierzchołków reprezentujących stany aktywne – stany, w których przebywa maszyna. W przypadku, gdy maszyna przebywa w stanie złożonym dla określenia konfiguracji nie wystarczy tylko wskazanie tego stanu jako stanu aktywnego, ale

również wszystkich jego podstanów aktywnych. Konfiguracja stanowi więc pewne drzewo wierzchołków, którego korzeniem jest wierzchołek reprezentujący najbardziej zewnętrzny stan, a jego liśćmi są wierzchołki reprezentujące stany elementarne.

Semantykę maszyny stanów wyraża się przez określenie przejść pomiędzy konfiguracjami maszyny stanów. Przejścia pomiędzy konfiguracjami są złożeniem (sekwencyjnym lub równoległym) dwóch rodzajów przejść pomiędzy wierzchołkami: przejść niskopoziomowych (pojedynczych), związanych z łukami prowadzącymi od wierzchołków reprezentujących stany elementarne, oraz – wysokopoziomowych (grupowych), związanych z łukami prowadzącymi od lub do wierzchołków reprezentujących stany złożone.

Początkową konfigurację wyznaczają początkowe stany domyślne. Są to stany wskazane przez łuki prowadzące od pseudostanów początkowych. Jeżeli tak wskazywany stan SS jest stanem złożonym sekwencyjnym, to do konfiguracji początkowej należy także jeden z jego podstanów S wskazywany jako domyślny stan początkowy w obrębie podstanów stanu SS. Jeżeli wskazywany stan SR jest stanem złożonym równoległym, to do konfiguracji początkowej należą wszystkie jego podstany $S_1, \dots, S_n$ wskazywane jako domyślne stany początkowe w obrębie podstanów równoległych stanu SR. Z każdym stanem wiąże się więc zbiór jego domyślnych podstanów. Zbiór ten, podobnie jak cała konfiguracja maszyny stanów, ma strukturę drzewa.

Przejścia pomiędzy konfiguracjami są wyznaczone przez następujące zasady.

Wyjście ze stanu może nastąpić jeżeli:

- dany stan jest aktywny, tzn. należy do bieżącej konfiguracji maszyny,
- wystąpiło zdarzenie wskazane na łuku wyprowadzającym ze stanu lub nastąpił moment przeterminowania i prawdziwy jest dozór związany z łukiem,
- zakończone zostały wszystkie aktywności w obrębie stanu, a łuk wyprowadzający ze stanu nie jest etykietowany zdarzeniem oraz prawdziwy jest związany z tym łukiem dozór.

Jeżeli w danej chwili możliwe są różne wyjścia, to wybór wyjścia może być niedeterministyczny lub oparty o dodatkowo wprowadzone reguły, np. w oparciu o nadane priorytety.

Wyjściu ze stanu prostego towarzyszy wykonanie akcji umieszczonych na liście akcji wyjściowych danego stanu, następnie wykonywana jest akcja związana z wybranym łukiem, a następnie wykonywane są akcje wejściowe stanu, do którego następuje przejście. Jeżeli wyjście następuje w momencie trwania aktywności w obrębie stanu, wówczas aktywność taka zostaje przerwana.

Wyjście ze stanu złożonego oznacza również wyjście ze wszystkich jego podstanów. Pociąga to wykonanie ciągu akcji wyjściowych, poczynając od akcji dla stanów najbardziej zagnieżdżonych. Ciąg ten jest wyznaczony jednoznacznie w przypadku zagnieżdżania stanów sekwencyjnych, natomiast w przypadku zagnieżdżania stanów równoległych odpowiadające im akcje mogą się przeplatać.

Wejściu do stanu prostego towarzyszy wykonanie akcji wejściowych związanych z tym stanem, a następnie rozpoczyna się ewentualne wykonywanie związanych z nim aktywności.

Wejście do stanu złożonego pociąga wchodzenie do jego podstanów i towarzyszy temu wykonanie ciągu akcji wejściowych związanych z tymi podstanami. Ciąg ten jest określony jednoznacznie w przypadku wchodzenia do podstanów sekwencyjnych. W przypadku wchodzenia do podstanów równoległych ciąg ten jest przeplotem akcji należących do tych podstanów.

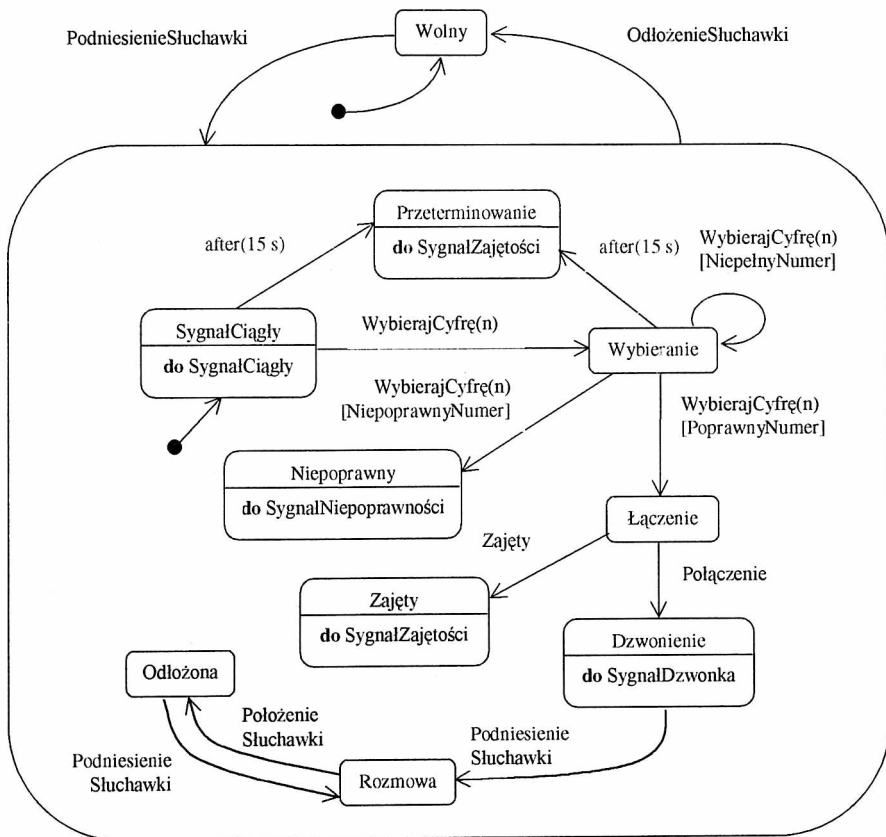
Jeżeli wejście następuje do łącznika historycznego zagnieżdżonego w pewnym stanie S, to aktywnymi w stanie S stają się:

- domyślne podstawy początkowe stanu S, gdy wejście do S następuje po raz pierwszy,
- domyślne stany początkowe stanu wskazywanego przez łącznik historyczny „płytki”, gdy wejście do S następuje po raz kolejny,

- ostatni zestaw stanów aktywnych przed opuszczeniem stanu S, gdy wejście do S następuje po raz kolejny, a łącznik historyczny jest łącznikiem „głębokim”.

Przejścia wysokopoziomowe mają priorytet nad przejściami niskopoziomowymi.

Przejścia pomiędzy stanami, w odróżnieniu od klasycznych map stanów, nie muszą być natychmiastowe. Wynika to z faktu, że przejściom towarzyszy wykonywanie akcji, które w porównaniu do aktywności są czynnościami niepodzielnymi i mogą jednak wymagać pewnego czasu.



Rys. 19. Przykład diagramu stanów

8. Diagramy aktywności

diagram aktywności

Specyficzny rodzaj diagramu stanów, w którym wszystkie lub większość stanów są stanami aktywności, i w którym wszystkie lub większość przejść pomiędzy stanami jest powodowana zakończeniem wykonywania aktywności.

stan aktywności

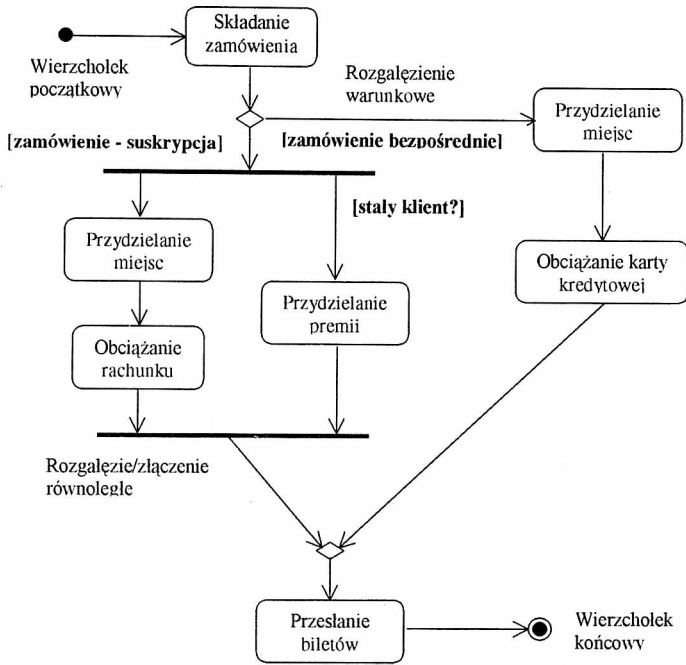
Stan reprezentujący wykonywanie aktywności, na przykład wykonanie operacji.

Stany aktywności odpowiadają stanom wykonywania programu. W przypadku programu współbieżnego stany odnoszą się do stanów poszczególnych procesów składowych całego programu. Pojęcie stanu aktywności należy odróżniać od stanu obiektu. Aktywności mogą na siebie wpływać przez wysyłanie i odbiór zdarzeń.

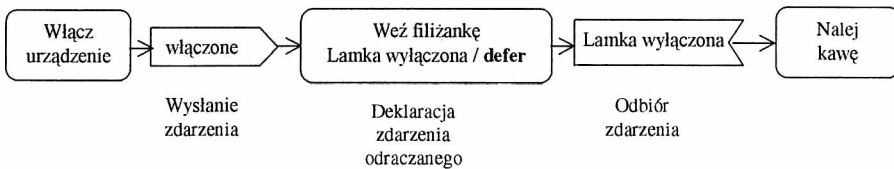
Diagram aktywności ma postać grafu, którego wierzchołkami – poza stanami aktywności – są również stan początkowy, końcowy, rozgałęzienia i złączenia równoległego oraz rozgałęzienia warunkowego i kolekcyjnego. Wierzchołki inne od stanów aktywności reprezentują więc pewne akcje pomocnicze. Wierzchołki reprezentujące stany aktywności są zwykle etykietowane nazwami tych aktywności, a wierzchołki rozgałęzienia warunkowego – wyrażeniami wyboru. Łuki grafu nie są na ogół etykietowane. Nie dotyczy to łuków wychodzących z wierzchołka rozgałęzienia warunkowego, które określają warunki przejścia do następnych stanów. Wyjście z danego stanu i przejście do innego stanu następuje w momencie zakończenia związanej z nim aktywności. Przykład diagramu aktywności przedstawia rys. 20.

Zakończenie aktywności może też być wynikiem reakcji na odbierane zdarzenia. Zdarzenia, które pojawiają się w czasie trwania aktywności mogą być przez nią obsługiwane albo odraczane, tzn. są tylko zapamiętywane z przeznaczeniem do obsługi przez inną aktywność. Jeżeli zdarzenie wygenerowane w danej chwili nie jest obsługiwane ani odraczane przez żadną z aktualnie wykonywanych aktywności, to jest gubione. Natomiast w przypadku przeciwnym, gdy istnieje wiele potencjalnych reakcji na zdarzenie, wybór odpowiedniej reakcji jest niedeterministyczny. Można wybór ten

uściślić przez wprowadzenie dodatkowych zasad. Wysyłanie i odbiór zdarzeń na diagramie aktywności jest reprezentowane przez dodatkowe wierzchołki – rys. 21.



Rys. 20. Przykład diagramu aktywności

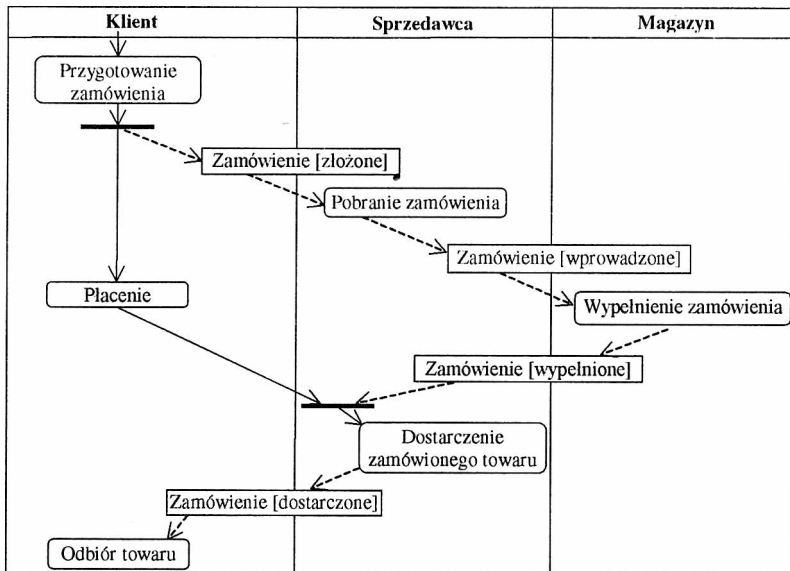


Rys. 21. Wysyłanie i odbiór zdarzeń na diagramie aktywności

Zasadniczo, graf aktywności jest interpretowany jako opis przepływu sterowania pomiędzy aktywnościami. Diagram aktywności umożliwia też opis przepływu danych pomiędzy aktywnościami. Jest to uwidocznione na diagramie w postaci dodatkowych wierzchołków reprezentujących przesyłane obiekty (prostokąty rysowane ciągłą linią) oraz przez dodatkowe łuki (rysowane przerywanymi strzałkami) wskazujące drogi przesyłania obiektów pomiędzy aktywnościami.

Diagram aktywności z przepływem danych jest pokazany na rys. 22.

Dodatkowym elementem na tym diagramie są tzw. linie rozgraniczenia odpowiedzialności wskazujące obszary grafu podzielone na aktywności wykonywane przez różnych wykonawców, w pokazywanym przykładzie przez **Klienta**, **Sprzedawcę** i **Magazyn**. Na diagramie zwracają też uwagę wierzchołki reprezentujące złączenie równoległe; ich łukami wejściowymi i łukami wyjściowymi mogą być zarówno łuki reprezentujące przepływ sterowania jak i łuki reprezentujące przepływ danych.



Rys. 22. Diagram aktywności z przepływem sterowania i przepływem danych

9. Diagramy implementacyjne

diagram komponentów

Diagram pokazujący komponenty i zależności pomiędzy nimi; odzwierciedla statyczny aspekt perspektywy implementacyjnej.

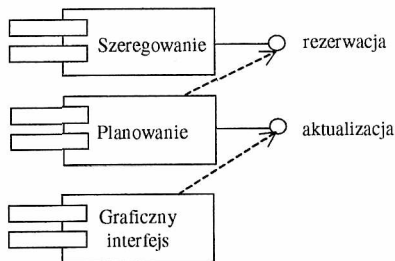
diagram rozmieszczenia

Diagram przedstawiający konfigurację węzłów przetwarzających oraz umieszczonych w nich komponentów. Przedstawia on statyczną strukturę systemu z punktu widzenia perspektywy instalacyjnej.

Komponent reprezentuje zestaw zasobów programowych i informacyjnych. Przykładami takich zasobów są moduły programowe (źródłowe lub wykonywalne), biblioteki procedur, bazy danych lub ich odpowiedniki, np. skrypty lub pliki z komendami. Komponent realizuje pewne usługi, które udostępnia za pośrednictwem swoich interfejsów (interfejs jest zbiorem operacji). Komponent ma w systemie swoją reprezentację fizyczną i może być w systemie przemieszczalny. Szczególną rolę w komponentach pełnią zadania. Zadanie jest to obiekt aktywny, czyli taki, który posiada własne sterowanie (wątek lub proces) niezależne od innych zadań. Zadania są wykonywane równolegle.

Pomiędzy komponentami mogą zachodzić relacje zależności, generalizacji, asocjacji i realizacji.

Przykład diagramu komponentów przedstawia rys. 23.



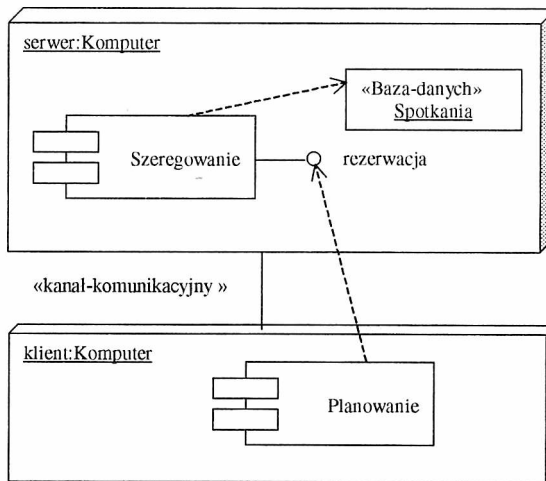
Rys. 23. Przykład diagramu komponentów

Wskazuje się tu pewne relacje (przerywane strzałki) zależności pomiędzy komponentami a ich interfejsami, przy czym bezpośrednio z samego diagramu nie wynika znaczenie tych relacji. Znaczenie takich relacji można ustalić przez wprowadzenie odpowiednich stereotypów.

Diagram rozmieszczenia zawiera węzły i komponenty oraz zachodzące pomiędzy nimi relacji zależności i asocjacji.

Węzeł jest fizycznym obiektem reprezentującym środowisko wykonawcze – sprzęt komputerowy wraz z oprogramowaniem operacyjnym. Węzeł powinien dysponować co najmniej pamięcią, często ma jednostki przetwarzające. Przykładowo, węzłami są komputer z systemem operacyjnym, procesor z pamięcią, pamięć dyskowa.

Przykład diagramu rozmieszczenia przedstawia rys. 24.



Rys. 24. Przykładowy diagram rozmieszczenia

10. Diagramy pakietów

Diagramy pakietów nie należą do standardowych diagramów UML. Są one często przydatne w procesie modelowania i służą do grupowania aertefaktów, na przykład do definiowania podsystemów – części modelowanego systemu.

pakiet

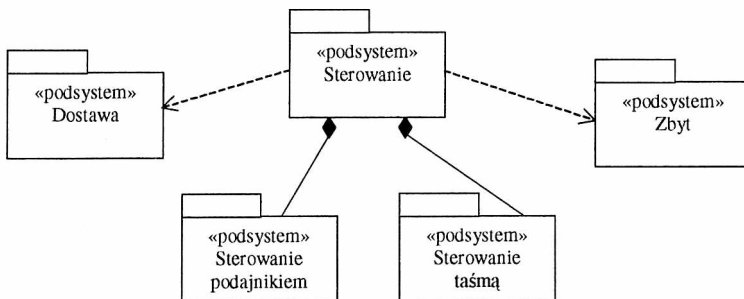
Mechanizm ogólnego przeznaczenia służący grupowaniu elementów. Pakiety mogą być zagnieżdżane w innych pakietach. System może być wyobrażany jako pakiet najwyższego poziomu, w którym jest zawarte wszystko co należy do systemu.

podsystem

Zbiór elementów opisujących zachowanie innych, zagnieżdżonych w nich elementów. Może być modelowany zarówno jako pakiet jak i klasa. Podsystem może posiadać interfejsy, przez które udostępnia usługi pozostałej części systemu.

Diagram pakietów składa się z pakietów oraz relacji zależności, asocjacji i generalizacji.

Przykład diagramu pakietów przedstawia rys. 25., który składa się z trzech podsystemów: *Dostawa*, *Sterowanie* i *Zbyt* powiązanych relacją zależności. Podsystem *Sterowanie* jest systemem złożonym i zawiera dwa podsystemy składowe: *Sterowanie podajnikiem* oraz *Sterowanie taśmą*.



Rys. 25. Przykładowy diagram pakietów

11. Wybrane standardowe elementy UML

UML jest językiem rozszerzalnym, co pozwala na jego adaptację do różnych, z góry nieprzewidywalnych sytuacji projektowych. Mechanizmami rozszerzeń są stereotypy, wartości znakowane (stałe) oraz ograniczenia. Pewne rozszerzenia są w UML zdefiniowane jako standardowe.

stereotyp

Nowy rodzaj elementu modelującego, który rozszerza semantykę metamodelu. Stereotyp musi bazować na wybranych typach i klasach istniejących w metamodelu. Stereotypy mogą tylko rozszerzać semantykę (uściślać) lecz nie naruszać istniejących (predefiniowanych) typów i klas. Pewne stereotypy są w UML zdefiniowane jako standardowe.

Stereotyp oznacza nowy rodzaj elementu modelującego, bazującego na wybranych typach i klasach istniejących w metamodelu. Definicja stereotypu polega na uściśleniu semantyki wybranego elementu. Każdy stereotyp ma nazwę, dla wyróżnienia zapisywaną w cudzysłowach postaci « ». Zamiast nazwy można też wprowadzać specyficzną ikonę. Nazwa stereotypu jest umieszczana przy symbolu graficznym elementu, który był podstawą do jego definicji. Poniżej przedstawiono w tabeli tylko kilka spośród ponad sześćdziesięciu standardowych stereotypów.

Niektóre przedstawiane stereotypy, w ścisłym sensie nie są stereotypami. Są tylko słowami kluczowymi, które występują w metamodelu, natomiast ich rola przy tworzeniu modelu jest taka sama jak stereotypów w ścisłym sensie. Dlatego odróżnia się je zapisując ich nazwy czcionką pochyloną, podczas gdy nazwy pozostałych zapisuje się czcionką prostą. Różnica pomiędzy tymi dwoma kategoriami stereotypów jest więc subtelna i z praktycznego punktu widzenia można jej nie dostrzegać.

Nazwa / ikona	Element odniesienia	Znaczenie
«actor»	klasa	Określa spójny zbiór ról, które reprezentuje użytkownik przypadku użycia.
«call»	relacja zależności	Określa, że jedna operacja wywołuje drugą operację.
«exception»	klasa	Oznacza zdarzenie, które może być przez operację wygenerowane bądź przejęte do obsługi.
«executable» 	komponent	Oznacza komponent wykonywalny – komponent może być wykonywalny w węźle.
«precondition»	ograniczenie	Oznacza ograniczenie, które musi być spełnione przez wywołaniem operacji.
«responsibility»	notka	Oznacza kontrakt lub zobowiązanie związane z klasą.
«subsystem»	pakiet	Oznacza kolekcję elementów, z których pewne elementy specyfikują zachowanie innych elementów należących do tej kolekcji.
«system»	pakiet	Oznacza pakiet reprezentujący całość modelowanego systemu.
«type»	klasa	Oznacza abstrakcyjną klasę używaną tylko do specyfikacji struktury lub zachowania, ale nie do implementacji zbioru obiektów.
«use»	relacja zależności	Oznacza, że semantyka jednego elementu zależy od semantyki części publicznej drugiego elementu.

wartość znakowana

Bezpośredni opis właściwości w postaci pary nazwa-wartość.

Wartość znakowana (stała) jest parą złożoną z etykiety (nazwy) oraz przypisanej jej wartości. Nazwa może być dowolnym tekstem, a wartość ma również formę tekstową i może być elementem dowolnie ustalonego zbioru. Parę taką zapisuje się w postaci *etykieta = wartość* i można ją przypisać dowolnemu elementowi modelu. Znaczenie stałej zależy od kontekstu. Nie należy stałej traktować jako atrybutu elementu, np. obiektu. Standardowo wyróżniono cztery etykiety opisane w tabeli poniżej.

Nazwa stałej	Element odniesienia	Znaczenie
documentation	wszystkie elementy	Oznacza komentarz, opis lub wyjaśnienie dotyczące elementu, z którym jest związana
location	większość elementów	Oznacza węzeł lub komponent, w którym jest umieszczony dany element
persistence	klasa asocjacja atrybut	Oznacza, że po zakończeniu procesu tworzenia danej instancji jej stan zostaje zachowany
semantics	klasa operacja	Określa znaczenie klasy lub operacji

ograniczenie

Semantyczny warunek lub zawężenie.

Ograniczenie ma postać tekstu, pisanego w języku naturalnym lub formalnym, na przykład w języku OCL (*Object Constraint Language*), który jest językiem predykatów przystosowanym do definiowania metamodelu UML. Wyrażenie takie może być związane z jednym lub wieloma elementami modelu i uściśla lub ogranicza semantykę tych elementów. Wyrażenie ma postać komentarza (notki) połączonego linią przerywaną z elementami których dotyczy, może też mieć postać wyrażenia ujętego w nawiasy kłamrowe, umieszczonego obok elementu, którego dotyczy. W tabeli poniżej przedstawiono trzy spośród dziewięciu standardowych ograniczeń.

Ograniczenie	Element odniesienia	Znaczenie
complete	relacja generalizacji	Oznacza, że w modelu wyspecyfikowano wszystkich potomków danej klasy
or	asocjacja	Oznacza, że w zbiorze asocjacji wiążących klasę z innymi klasami, mogą w konkretnym powiązaniu wystąpić element należące tylko do jednej z tych asocjacji.
transient	instancja powiązanie	Oznacza, że instancja lub powiązanie jest tworzone okresowo podczas trwania pewnych interakcji

12. Elementy metamodelu UML

Opis języka UML należy rozpatrywać w kontekście wyznaczonym przez czteropoziomową architekturę metamodelowania. Są to poziomy:

- meta-metamodelu,
- metamodelu,
- modelu abstrakcyjnego,
- modelu użytkowego.

Definicja UML mieści się na poziomie metamodelu. Modelowi abstrakcyjnemu odpowiada pewien model napisany w UML, a modelowi użytkowemu odpowiada skompilowana postać tego modelu. Natomiast najbardziej abstrakcyjny poziom meta-metamodelu obejmuje swym zasięgiem język, w zasadzie metajęzyk, użyty do definicji języka UML.

Podobny podział pojęć odnoszących się do modelowania obiektowego został przyjęty przez OMG w ramach architektury *Meta-Object Facility*.

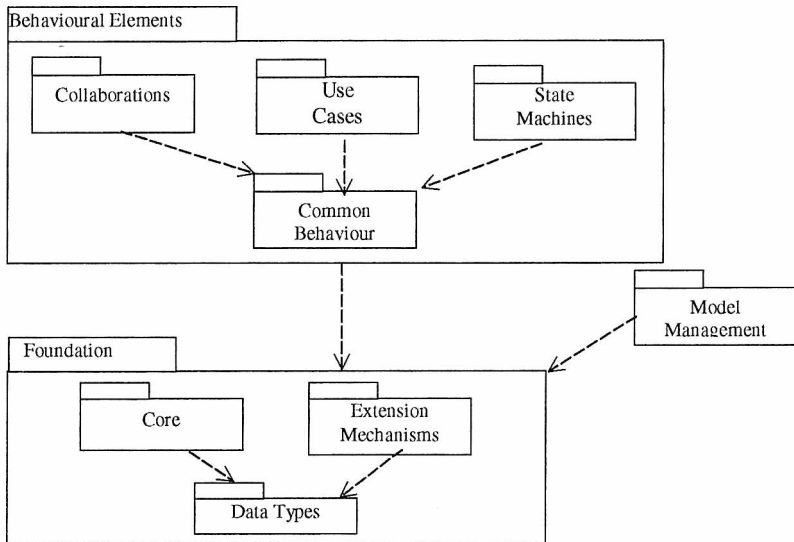
Mówiąc o standardzie języka UML mamy na uwadze metamodel. Metamodel grupuje wszystkie elementy języka w trzy podstawowe grupy obejmowane przez trzy podstawowe pakiety pokazane na rys. 26, z których dwa zawierają jeszcze podpakiety.

Pakiet *Foundation* zawiera trzy podpakiety.

Podpakiet *Core* zawiera główne elementy strukturalne: klasyfikatory (klasy, komponenty, węzły), ich zawartość (atrybuty, operacje, metody, parametry) oraz relacje (generalizacji, asocjacji, zależności).

Podpakiet *Data Types* opisuje typy danych wykorzystywane w metamodelu.

Podpakiet *Extension Mechanisms* opisuje stereotypy, ograniczenia i stałe (wartości znakowane).



Rys. 26. Struktura pakietów metamodelu UML

Pakiet *Behavioural Elements* zawiera cztery podpakiety.

Podpakiet *Common Behaviour* opisuje sygnały, operacje i akcje.

Podpakiet *Collaborations* opisuje grupy współdziałania, interakcje, komunikaty, role klasyfikatorów oraz asocjacje.

Podpakiet *Use Cases* opisuje aktorów i przypadki użycia.

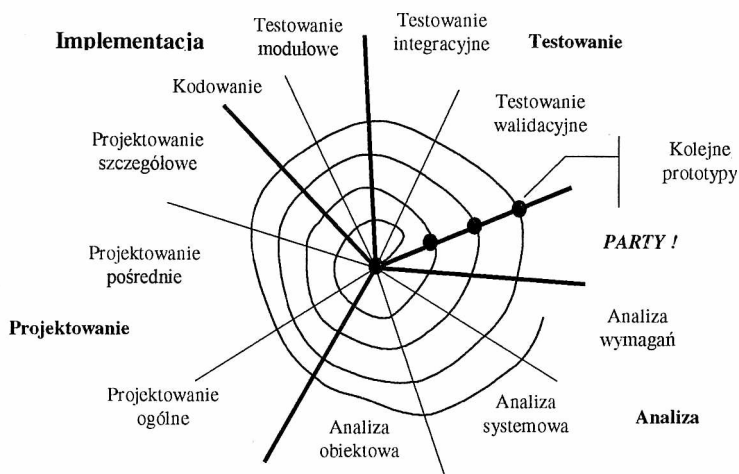
Podpakiet *State Machines* opisuje strukturę maszyn stanowych – stany zdarzenia, sygnały, tranzyje pomiędzy stanami, oraz modele aktywności.

Pakiet *Model Management* opisuje pakiety, modele oraz podsystemy.

13. Metodyka ROPES

UML jest niezależny od przyjętej metody wytwarzania oprogramowania i dlatego jest wykorzystywany w wielu metodach, np. *Rational Unified Process*, *Unified Software Development Process*, *Catalysis*, *ROPES*.

W projektowaniu systemów czasu rzeczywistego godną polecenia jest metoda *ROPES* (*Rapid Oriented Process for Embedded Systems*), która wyróżnia iteracyjnie powtarzane fazy: analizy, projektowania, implementacji i testowania. Fazy dzieli się jeszcze na podfazy jak pokazano na rys. 27.



Rys. 27. Spiralny proces wytwarzania oprogramowania według metody ROPES

Celem pojedynczego cyklu jest wytworzenie pewnego prototypu systemu. Cykle powtarza się aż uzyskany prototyp całkowicie spełnia oczekiwania użytkownika. Celem każdej z faz jest zbudowanie pewnego modelu budowanego systemu. Model otrzymany w jednej fazie jest uściślany lub uzupełniany w fazie następnej.

13.1. Analiza

Analiza wymagań

Celem analizy wymagań jest określenie wymagań funkcjonalnych i niefunkcjonalnych systemu. Wymagania funkcjonalne mają określić usługi wytwarzanego systemu, a wymagania niefunkcjonalne mają, między innymi, określić jakość tych usług. Miarą jakości usług mogą być dostępność,

niezawodność, bezpieczeństwo, zabezpieczenie przed niepowołanym dostępem, i wreszcie – wymagania czasowe. UML nie oferuje szczególnych mechanizmów do wyrażania wymagań niefunkcjonalnych. Można je oczywiście wyrazić za pomocą komentarzy w języku naturalnym.

Przy stosowaniu podejścia obiektowego analiza wymagań jest oparta na przypadkach użycia. Zbiór przypadków użycia związanych z danym systemem wyraża się za pomocą diagramów przypadków użycia.

W początkowej fazie analizy tymi obiektami są: system traktowany jako całość oraz obiekty zewnętrzne. W dalszej analizie, gdy system zostaje zdekomponowany na obiekty wewnętrzne, dekompozycji ulegają także przypadki użycia.

Pojedyncze scenariusze – instancje przypadku użycia – przedstawia się za pomocą diagramów interakcji, a zbiór wszystkich scenariuszy związany z przypadkiem użycia wyraża odpowiednia maszyna stanów.

Model systemu, który powstaje w wyniku analizy wymagań zawiera diagram przypadków użycia, diagramy interakcji i diagramy maszyn stanów, a także może zawierać inne diagramy, np. diagramy klas.

Model systemu ma służyć określeniu podstawowych funkcji oraz protokołów komunikacji pomiędzy systemem a jego otoczeniem, a także określeniu podstawowych czasowych charakterystyk wymiany informacji pomiędzy systemem a otoczeniem.

Diagram przypadków użycia pozwala na charakterystykę strumieni komunikatów napływających od aktorów do systemu. Mogą one dotyczyć intensywności napływu komunikatów i sposobu synchronizacji aktorów z przypadkami użycia. Charakterystyki te mają postać stereotypów specjalnie zdefiniowanych dla systemów czasu rzeczywistego. Przykładem standardowych stereotypów są aktorzy, relacje zależności «include» oraz «extend» pomiędzy przypadkami użycia, zaś przykładami stereotypów specyficznych są charakterystyki strumieni komunikatów «periodic» oraz «aperiodic» występujący na diagramie przypadków użycia.

Diagram sekwencji pozwala na informowanie o związkach czasowych pomiędzy napływającymi komunikatami a odpowiadającymi im reakcjami, czy też – ogólniej – pomiędzy wyróżnionymi zdarzeniami. Relacje te wyraża się w postaci formuł, w których występują odwołania do momentów występowania wyróżnionych zdarzeń. Diagramy stanów także wyrażają własności czasowe, ale czynią to z punktu widzenia obiektu, który w trakcie swego życia ma reagować na zachodzące zdarzenie, w tym na upływ czasu. Najczęściej ograniczenia czasowe wyrażają się przez okresy przeterminowania związane z oczekiwaniem w danym stanie na zajście określonych zdarzeń.

Charakterystyka strumienia komunikatów

Komunikaty ze źródła do miejsca odbioru mogą napływać periodycznie (stereotyp «periodic») lub aperiodycznie (stereotyp «aperiodic»). W przypadku napływu periodycznego komunikaty napływają regularnie w zadanych odstępach czasu, ale odstęp ten może się zmieniać w pewnym zakresie – może mieć fluktuacje. Charakterystyka napływu aperiodycznego (sporadycznego) jest bardziej urozmaicona. Napływ może być nieregularny, ale mogą być scharakteryzowane okresy pomiędzy kolejnymi komunikatami. Charakterystyką taką może być:

- maksymalna liczba komunikatów w pewnym okresie czasu,
- minimalny odstęp czasu pomiędzy kolejnymi komunikatami,
- probabilistyczny rozkład czasu pomiędzy komunikatami.

Gdy napływ komunikatów jest lawinowy (wiązkowy) wówczas odstępy czasu pomiędzy kolejnymi komunikatami mogą być dowolnie małe, ale powinna być znana maksymalna liczba komunikatów w pojedynczej wiązce. Gdy napływ jest ograniczony, wówczas jego charakterystyka jest określona przez minimalny lub przez średni odstęp czasu pomiędzy kolejnymi komunikatami.

Sposób współdziałania komunikujących się obiektów

Ogólnie synchronizacja może dotyczyć grupy obiektów. UML przyjmuje możliwość synchronizacji tylko pomiędzy dwoma obiektami: obiektem-nadawcą i

obiektem-odbiorcą komunikatu. W języku UML wymiana komunikatów pomiędzy parą współpracujących obiektów może opierać się na następujących schematach.

Współpraca synchroniczna polega na tym, że obiekty na okres wykonania operacji ustalają pomiędzy sobą spotkanie. Możliwe są następujące trzy schematy takiej współpracy.

- Obiekt-usługobiorca wywołuje operację i czeka tak długo, aż obiekt-usługodawca zacznie wykonywać operację i przekaze obliczone wyniki.
- Obiekt-usługobiorca wywołuje operację i czeka tak długo, aż obiekt-usługodawca potwierdzi wywołanie operacji.
- Obiekt-usługobiorca wywołuje operację i czeka na potwierdzenie wywołania operacji przez zadany okres przeterminowania. Jeżeli potwierdzenie to nie zostanie dostarczone do obiektu-usługobiorcy przed upływem okresu przeterminowania, obiekt ten przerywa połączenie i wznowia swoją aktywność.

Współpraca asynchroniczna polega na tym, że jeden obiekt wysyła sygnał do innego obiektu (lub do grupy obiektów) i nie czekając na odbiór tego sygnału kontynuuje swą aktywność.

Analiza systemowa i obiektowa

Analiza systemowa polega na dekompozycji systemu na części składowe (mechaniczne, elektroniczne, programowe) i przypisaniu im odpowiednich funkcji i zachowań. Analiza taka występuje zwłaszcza przy projektowaniu systemów wbudowanych i polega, między innymi, na wyodrębnieniu sensorów i urządzeń wykonawczych. Bardzo często z uwagi na prostą strukturę systemów, analizę tę można pominąć. Analizę obiektową dzieli się na analizę strukturalną – której celem jest zidentyfikowanie jednostek strukturalnych (klas i obiektów oraz grupujących je pakietów), oraz na analizę behawioralną – której celem jest określenie zachowań poszczególnych jednostek strukturalnych.

Z punktu widzenia wymagań czasowych odnoszących się do całego budowanego systemu istotna jest dekompozycja tych wymagań na wyłonione składowe i obiekty. Dla

każdego z tych elementów należy scharakteryzować własności czasowe otoczenia – własności czasowe strumieni komunikatów, schemat synchronizacji oraz wymagane czasy odpowiedzi. Wymagania te można wyrazić w UML w taki sam sposób jak w przypadku analizy wymagań, czyli przez diagramy interakcji oraz diagramy stanów.

13.2. Projektowanie

W projektowaniu wyróżnia się trzy podfazy różniące się stopniem szczegółowości podejmowanych decyzji. Są to kolejno: projektowanie ogólne (ponad-obiektowe), projektowanie pośrednie (między-obiektowe) i projektowanie szczegółowe (wewnątrz-obiektowe). Rys. 28. symbolicznie ilustruje trzy poziomy szczegółowości odpowiadające trzem wymienionym podfazom.

Projektowanie ogólne (ponad-obiektowe)

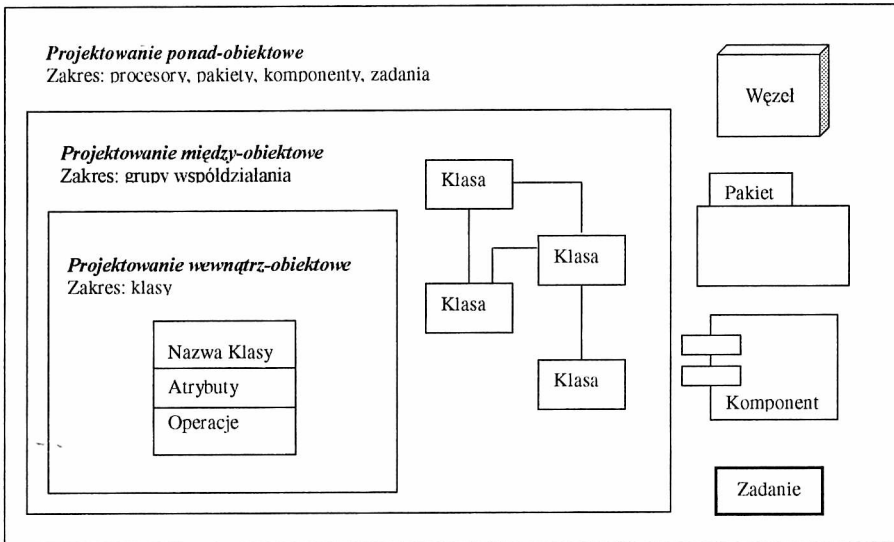
Celem projektowania ogólnego jest określenie architektury systemu, czyli składowych systemu i wzajemnego ich powiązania. Wyróżnia się trzy rodzaje składowych: zadania, komponenty i węzły. Za pomocą pakietów można je dowolnie grupować ze sobą.

Zadanie jest to obiekt aktywny, czyli taki, który posiada własne sterowanie (wątek lub proces) niezależne od innych zadań. Zadania są wykonywane równolegle.

Komponent reprezentuje zestaw zasobów programowych i informacyjnych.

Projektowanie ogólne polega na określeniu struktury środowiska wykonawczego oraz na rozmieszczeniu w tym środowisku tworzonego oprogramowania. Wynikiem projektowania jest wydzielenie zadań, przypisanie ich wraz z obiektami biernymi do komponentów, a następnie rozlokowanie komponentów w węzłach systemu.

Węzeł jest fizycznym obiektem reprezentującym środowisko wykonawcze – sprzęt komputerowy wraz z oprogramowaniem operacyjnym.



Rys. 28. Trzy poziomy projektowania

Właściwy podział na zadania ma krytyczny wpływ na wydajność systemu. W zasadzie wszystkie obiekty wyłonione w fazie analizy są równoległe względem siebie. W praktycznie spotykanych sytuacjach nie ma potrzeby, aby wszystkim obiektom przypisywać rolę zadań, gdyż zwiększa to stopień równoległości systemu i na ogół pociąga wzrost kosztów. Niektóre obiekty, ze względu na swą funkcję muszą być zadaniami. Przykładowo dotyczy to obiektów, które są odpowiedzialne za komunikację z otoczeniem systemu, czy też obiekty obsługujące niezależne urządzenia. Podział na zadania można więc traktować jako wynik analizy polegającej na tym, czy można dany obiekt, bez istotnego wpływu na wydajność systemu, pozbawić własności współbieżności i dołączyć do pewnej wspólnej puli obiektów. Z takiej puli, w danej chwili, na skutek zewnętrznej decyzji, może być wykonywany co najwyżej jeden obiekt. Wiadomo bowiem, że zwiększenie stopnia równoległości może tylko do pewnego stopnia może zwiększyć wydajność całego systemu.

Zadania i sposób współpracy zadań może być przedstawiony przez diagram obiektów (aktywnych) lub przez diagram stanów równoległych. Komponenty i zachodzące między nimi relacje oraz węzły są ukazane na diagramach implementacyjnych – diagramie komponentów, oraz diagramie rozmieszczenia, który ukazuje rozlokowanie komponentów w węzłach. Poszczególnym komponentom można przyporządkować przypadki użycia. Zestawienie wymienionych diagramów wyznacza model architektury systemu.

Dodatkowymi aspektami, które są brane pod uwagę w projektowaniu ogólnym, i które powodują poszerzenie funkcji systemu, są: obsługa sytuacji wyjątkowych, bezpieczeństwo (zapobieganie sytuacjom hazardowym) oraz tolerowanie błędów obliczeń (niezawodność obliczeń).

Z punktu widzenia wymagań czasowych stawianych projektowanemu systemowi konieczne jest sprawdzenie, czy podjęte decyzje zapewnią spełnienie wymagań wyrażonych w modelu w fazie analizy. Ze względu na złożoność analiza taka nie jest tu omawiana. Warto tylko zaznaczyć, że podstawowe potrzebne charakterystyki to wydajność przetwarzania węzłów i przepustowość wiążących ich łączy. Wydajność przetwarzania węzłów należy oszacować na podstawie wymagań czasowych zadań umieszczanych w węzłach. Przepustowość łączy pomiędzy węzłami szacuje się na podstawie charakterystyk strumieni komunikatów generowanych przez umieszczone w węzłach zadania. Dodatkowo, w odniesieniu do pojedynczych węzłów należy zaprojektować szeregowanie wykonywanych w nim zadań.

Projektowanie pośrednie (między-obiektowe)

Celem projektu ogólnego jest zdekomponowanie przypadków użycia na grupy współdziałania. Grupa współdziałania jest zbiorem powiązanych klasyfikatorów (klas, interfejsów, typów danych oraz komponentów) przeznaczonych do realizacji przypadków użycia lub operacji.

Natomiast celem projektu pośredniego jest uściślenie modelu systemu otrzymanego w fazie projektowania ogólnego. Zwykle polega to na wprowadzeniu dodatkowych

obiektów pośredniczących pomiędzy istniejącymi już obiektami. Konieczność pośrednictwa może wynikać z własności środowiska wykonawczego, na przykład dwa obiekty powiązane asocjacją mogą zostać umieszczone w dwóch różnych węzłach, bądź też może wynikać z potrzeby optymalizacji funkcjonowania, na przykład może zaistnieć potrzeba wprowadzenia obiektu zarządzającego pracą węzła.

Specyficznym postępowaniem w tej podfazie, a także w podfazie poprzedniej, jest wykorzystywanie wzorców projektowych. Wzorec jest pewną abstrakcją grupy współdziałania. Konkretyzacja wzorca wyznacza pewną grupę współdziałania. Grupa współdziałania jest elementem języka UML, natomiast nie jest nim pojęcie wzorca projektowego.

Stosowanie wzorców znacznie upraszcza proces projektowania, gdyż sięga się po rozwiązania sprawdzone, często o znanych oszacowaniach ich parametrów wydajnościowych.

Projektowanie szczegółowe (wewnątrz-obiektowe) i implementacja

Jest to faza ściśle związana z implementacją. Projektowanie szczegółowe koncentruje się na obiektach – na sposobach reprezentacji ich atrybutów, przyjęciu strategii realizacji powiązań między obiektami, wyborze konkretnych algorytmów realizacji operacji, przyjęciu zasad obsługi sytuacji wyjątkowych. Implementacja polega na reprezentacji atrybutów obiektu i zapisie opracowanych algorytmów dla jego operacji w wybranym języku programowania.

Uwagi końcowe

Przedstawiono tylko wybrane zagadnienia związane z projektowaniem systemów, w których istotne jest spełnianie warunków czasowych. Spośród diagramów występujących w UML szczególna uwaga, poza diagramem przypadków użycia, powinna być skupiona na diagramach interakcji i diagramach maszyn stanów. Pozwalają one na wyrażanie własności czasowych budowanych systemu.

W fazach analizy i projektowania dwa szczególne problemy wymagają rozwiązania. Pierwszy – to stwierdzanie zgodności pomiędzy diagramami interakcji a

diagramami maszyn stanowych, które należą do tego samego modelu systemu, tzn. modelu na danym poziomie analizy lub projektowania. Drugi problem to stwierdzenie, że model bardziej szczegółowy spełnia wymogi czasowe określone przez model ogólniejszy.

W fazach projektowania szczegółowego i implementacji zasadniczym problemem jest stwierdzenie, czy wydajność węzłów i sieci transmisji danych pozwala na spełnianie warunków czasowych podczas wykonywania zaprojektowanego oprogramowania.

Pozycjami, które stanowią bardzo dobre wprowadzenie zarówno do UML jak i projektowania systemów czasu rzeczywistego są książki Douglassa [4], [5]. Druga z nich dodatkowo zawiera informację o produkcie firmy i-Logix – pakiecie Rhapsody, który jest profesjonalnym narzędziem wspomagającym wytwarzanie oprogramowania czasu rzeczywistego.

Literatura

- [1] Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.
- [2] Booch G., Rumbaugh J., Jacobson I., *The Unified Software Development Process*, Addison-Wesley, 1999.
- [3] Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language. User Manual*, Addison-Wesley, 1999.
- [4] Douglass B.P., *Real-Time UML*, Addison-Wesley, 1998.
- [5] Douglass B.P., *Doing Hard Real. Developing Real-Time Systems with UML*, Addison-Wesley, 1999.
- [6] Ericsson H.-E., Penker M., *UML Toolkit*, John Wiley & Sons, Inc., 1997.
- [7] Górski J. (red.), *Inżynieria oprogramowania w projekcie informatycznym*, Mikom, 1999.
- [8] Hnatkowska B., Huzar Z., *Synchronizacja grupowa w UML*, Materiały II Krajowej Konferencji „Metody i systemy komputerowe w badaniach naukowych i projektowaniu inżynierskim”, 329-334, CCATIE, Kraków 1999.
- [9] Huzar Z., Magott J., *Wymagania czasowe w obiektowym procesie wytwarzania oprogramowania*, Informatyka, grudzień 1999.
- [10] *Open Systems Interconnection – Basic Reference Model of Open Distributed Processing*, ISO/IEC DIS 10746-2.
- [11] *Relationship of the Unified Language to the Reference Model of Open Distributed Processing*, Systemhouse Corporation, 1997, <http://enterprise.Systemhouse.com/UML-ODP>.
- [12] D’Souza D.F., Wills A.C., *Objects, Components and Frameworks with UML. The Catalysis™ Approach*, Addison-Wesley, 1999.
- [13] Subieta K., *Obiektość w projektowaniu i bazach danych*, Akademicka Oficyna Wydawnicza, 1998.
- [14] Szmuc T., Motet G., *Specyfikacja i projektowanie oprogramowania czasu rzeczywistego*, Krakowskie Centrum Informatyki Stosowanej, CCATIE 6, Kraków 1998.
- [15] Szmuc T., *Zaawansowane metody tworzenia oprogramowania czasu rzeczywistego*, Krakowskie Centrum Informatyki Stosowanej, CCATIE 15, Kraków 1998.
- [16] Turner K.J. (ed.), *Using Formal Techniques. Introduction to Estelle, Lotos and SDL*, Wiley, 1993.
- [17] *UML Notation Guide, Version 1.3*, Rational Software Corporation, October 1998.

- [18] *UML Semantics, Version 1.3*, Rational Software Corporation, October 1998.
- [19] *UML Object Constraint Language Specification, Version 1.3*, Rational Software Corporation, October 1998.
- [20] *UML Extension for Objectory Process for Software Engineering, Version 1.1*, Rational Software Corporation, 1997.
- [21] *UML Summary, Version 1.3*, Rational Software Corporation, October 1998.



BW-8



BIBLIOTEKA GŁÓWNA

315707/1
3w-8

ISBN 83-912747-4-8