1. Geneza UML

Metodyka jako zestaw pojęć, języków, modeli formalnych i sposobów postępowania jest wykorzystywana w procesie konstrukcji SI do modelowania pojęciowego, logicznego i fizycznego. Metodyka pozwala na podział procesu konstrukcji na fazy, dla każdej z faz ustala: produkty wejściowe i wyjściowe, scenariusze postępowania, reguły przechodzenia do kolejnej fazy oraz dokumentację, która powinna powstać, a więc umożliwia zdyscyplinowanie procesu konstrukcji pozwalając na w miarę obiektywne rozliczenie jego uczestników. Języki, elementy składowe metodyk, mogą wykorzystywać różne rodzaje notacji, np. tekstową, ustrukturalizowany zapis tekstowy i numeryczny czy też notacje graficzne. Języki, które powinny być proste a zarazem precyzyjne, łatwo zrozumiałe i nadające się do modelowania złożonych zależności, mogą być używane: do rejestrowania efektów pracy członków zespołu projektowego, do komunikacji wewnątrz zespołu, do współpracy z użytkownikiem, a także jako podstawowy środek zapisu dokumentacji projektowej. Język i jego notacja stanowią ważny element składowy każdej metodyki. Wiele osób utożsamia metodykę z wykorzystywaną przez nią notacją (nawet niekoniecznie językiem), ponieważ firmy rzadko stosują się do reguł narzucanych przez metodologów (opracowywując własne, bardziej przystosowane do wykorzystywanego oprogramowania narzędziowego, specyfiki wytwarzanych produktów, struktury organizacyjnej zespołów projektowych czy nabytych doświadczeń), ale wykorzystują notacje, w szczególności graficzne, wspomagane przez narzędzia CASE.

Wzrost popularności obiektowości w informatyce w końcu lat 80-tych i na początku 90-tych spowodował prawdziwą eksplozję metodyk i języków określanych jako „obiektowe": Objectory (Jacobson), OMT (Rumbaugh), OOA/OOD (Coad/Yourdon), OOAD (Booch), OOSA (Shlaer/Mellor), itd. Cechą wspólną jest tu oparcie o pojęcia obiektowości, takie jak: klasy, atrybuty, metody, dziedziczenie, związki, itp. Dość powszechne jest jednak odczucie, że różnice np. między językami w tych metodykach są drugorzędne, głównie syntaktyczne.

Zgodnie z deklaracją twórców: **UML** (Unified Modeling Language), który powstał w rezultacie połączenia trzech metodyk, „jest językiem ogólnego przeznaczenia do specyfikacji, konstrukcji, wizualizacji i dokumentowania wytworów związanych z systemami intensywnie wykorzystującymi oprogramowanie" . Powyższe stwierdzenie nie do końca jest prawdziwe. Każdy język, w tym UML, oprócz semantyki i składni, zawiera też pragmatykę – pragmatyka określa do jakiej konkretnej sytuacji występującej w procesie konstrukcji SI (analizie, projektowaniu, …) dopasować konkretny wzorzec notacyjny, a więc pośrednio prowadzi do określenia elementów pewnej metodyki.

2. Cele UML

UML zgodnie z deklaracją jego twórców, ma dokonać unifikacji języków używanych w innych metodykach obiektowych. Szanse rynkowe, nie dość że wzmocnione przez zaakceptowanie go jako standardu przemysłowego przez ciało standaryzacyjne OMG, są i tak duże, ponieważ autorzy: G. Booch, I. Jacobson i J. Rumbaugh to znani światowi metodolodzy, których metodyki opanowały około 40% rynku zastosowań metodyk obiektowych. Te metodyki to:

- **OOSE** (I. Jacobson): Ukierunkowana na modelowanie użytkowników i cyklu życiowego systemu. Nie przykrywa zarówno aspektu modelowania dziedziny przedmiotowej, jak i aspektu implementacji.
- **OOAD** (G. Booch): Dobrze podchodzi do kwestii projektowania, implementacji i związków ze środowiskiem implementacji. Nie przykrywa wystarczająco dobrze aspektu specyfikowania i analizy wymagań użytkowników.
- **OMT** (J. Rumbaugh): Ukierunkowana na modelowanie dziedziny przedmiotowej. Nie

przykrywa dostatecznie dobrze zarówno aspektu modelowania użytkowników systemu, jak i aspektu implementacji.

Podstawowym celem UML, zgodnie z intencjami autorów, było utworzenie języka do modelowania użytecznego zarówno dla ludzi, jak i dla maszyn – czegoś w rodzaju pomostu między ludzką percepcją i wyobraźnią, a automatyczną generacją fragmentów kodu, szkieletów aplikacji czy wreszcie całych aplikacji. Wydaje się, że zadanie uczynienia z języka do modelowania, uwzględniającego wymagania dotyczące naturalności, poglądowości i wysokiego poziomu abstrakcji, języka algorytmicznego o bardzo precyzyjnej składni i semantyce, jest co najmniej trudne, o ile z góry nie skazane na niepowodzenie i musi prowadzić do zmniejszenia czytelności diagramów, a przez to do zmniejszenia potencjału tego języka, jako środka do modelowania pojęciowego. Dlatego nie wszyscy są zachwyceni UML. Niektórzy uważają go za twór przereklamowany: zbyt obszerny i źle zdefiniowany.

3. Diagramy definiowane w UML

Mnogość aspektów konstruowanych SI z reguły prowadzi do zaistnienia w metodykach wielu modeli, opisywanych za pośrednictwem diagramów, i stanowiących coś w rodzaju „rzutu" projektowanego SI z pewnej perspektywy i na pewnym poziomie szczegółowości.

W UML definiowane są następujące diagramy:

- **Przypadków użycia**: Za główny cel mają odwzorowanie funkcjonalności projektowanego SI z perspektywy przyszłych użytkowników.
- **Klas** i **obiektów**; mają za zadanie odwzorowanie struktury systemu; najważniejsze z diagramów UML.
- **Dynamiczne**: Należące do nich diagramy interakcji (współpracy i sekwencji), diagramy stanu oraz diagramy aktywności stanowią element wspomagający w procesie uzupełniania struktury systemu o operacje umożliwiające realizację funkcjonalności wyspecyfikowanej w modelu przypadków użycia.
- **Implementacyjne**: Należą do nich diagramy komponentów i diagramy wdrożeniowe. Diagramy komponentów pokazują sposób odwzorowania fragmentów projektu na fragmenty kodu (komponenty), a diagramy wdrożeniowe przypisanie komponentów do fizycznych zasobów, czyli konfigurację elementów czasu wykonania.
- **Pakietów**: Wspomagają zarządzanie poprzez umożliwienie podziału modeli systemu na mniejsze elementy, zwane pakietami, i ustaleniu zachodzących między nimi relacji.

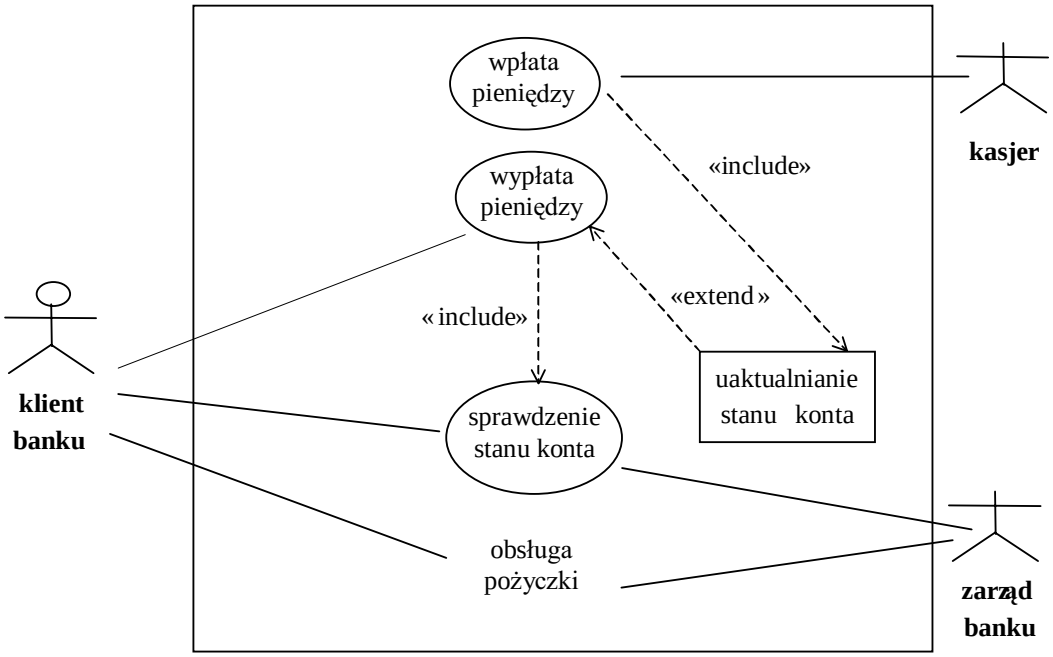
Diagramy przypadków użycia

Przypadki użycia, zwane scenariuszami zanim I. Jacobson wyodrębnił je jako oddzielną metodę projektowania, były intuicyjnie stosowane w konstruowaniu SI na długo przed pojawieniem się metodyk obiektowych.

Diagramy przypadków użycia odwzorowywują funkcjonalność systemu w taki sposób, w jaki będzie widoczna przez klasy przyszłych użytkowników, zwanych tu aktorami. Okazuje się, że dla dużych systemów, o wielu złożonych i wzajemnie powiązanych funkcjach, tego rodzaju podejście ma rzeczywiście duże znaczenie, sprzyja bowiem punktowi widzenia, w którym centralnym ośrodkiem zainteresowania staje się użytkownik systemu, a nie wewnętrzna struktura tego systemu. Przypadki użycia odwzorowywują strukturę systemu tak, jak ją widzą jego użytkownicy. Zadanie prawidłowego określenie wymagań funkcjonalnych na projektowany system staje się łatwiejsze, co jest ważne, bo prawidłowe określenie funkcjonalności systemu uznawane jest za jeden z podstawowych problemów w procesie konstrukcji.

Podsumowywując, model przypadków użycia pozwala na:

- lepsze zrozumienie możliwych sposobów wykorzystania projektowanego systemu (przypadków użycia), co oznacza zwiększenie stopnia świadomości analityków i projektantów co do celów, czyli funkcjonalności tego systemu,
- ustalenie praw dostępu do zasobów,
- ustalenie składowych systemu i związanego z nimi planu konstrukcji systemu,
- dostarczenie podstawy do sporządzenia planu testów systemu,
- lepszą interakcję zespołu projektowego z przyszłymi użytkownikami systemu.



Rys. 1. Przykład diagramu przypadków użycia.

Diagramy obiektów i diagramy klas

Diagram klas jest pojęciem centralnym we wszystkich znanych metodykach obiektowych. Stanowi pewną odmianę diagramów encja-związek. Zasadnicza nowość, to metody przypisane do klas. W UML, diagramy klas zostały praktycznie bez większych zmian przeniesione z OMT.

Diagram obiektów zawiera wyłącznie obiekty – wystąpienia klas. Diagram klas zawiera klasy (ale może też zawierać obiekty) powiązane w sieć zależnościami dwóch podstawowych rodzajów:

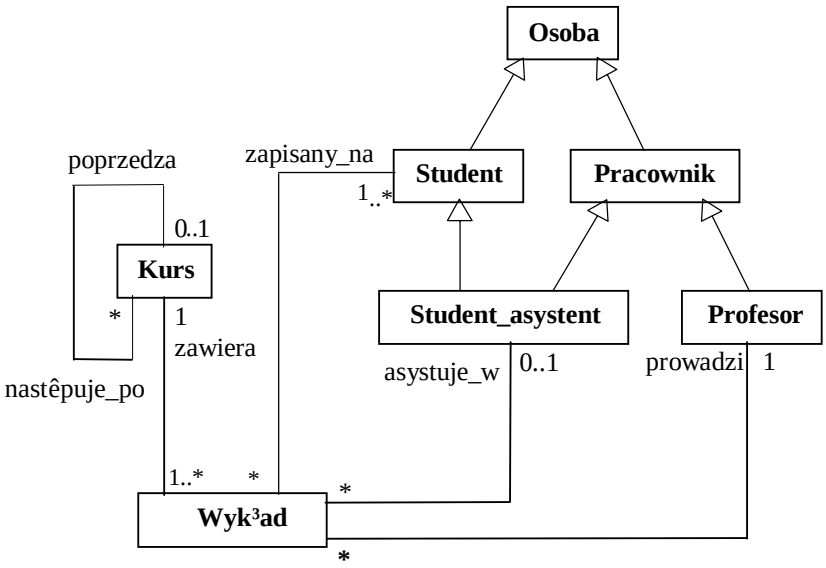
- **związki generalizacji-specjalizacji**, gdzie klasy bardziej wyspecjalizowane dziedziczą inwarianty (atrybuty, metody, zdarzenia, wyjątki i inne) z klas bardziej ogólnych,
- **asocjacje**, czyli dowolne związki między klasami odwzorowywujące zależności zachodzące między bytami w modelowanej dziedzinie przedmiotowej.

Specjalne rodzaje asocjacji to: asocjacja kwalifikowana, agregacja, kompozycja i asocjacja n-arna.

Asocjacja kwalifikowana pozwala na wyróżnienie kwalifikatora (pojedynczego atrybutu lub zestawu atrybutów asocjacji), którego wartość wyznacza w sposób unikalny zbiór obiektów pewnej klasy powiązanych z jednym obiektem klasy, do której „dostawiony" został kwalifikator. Np. *bank + numer konta* wyznaczają w sposób jednoznaczny właściciela konta.

Agregacja jest szczególnym przypadkiem asocjacji wyrażającym zależność część-całość. Pojęcie agregacji jest niejasne i rodzi mnóstwo wątpliwości. Zdania są podzielone nawet wśród światowych guru. Np. P. Coad podaje związek między organizacją a jej pracownikami jako przykład na zastosowanie agregacji, podczas gdy J. Rumbaugh twierdzi, że organizacja nie może być agregacją jej pracowników. Autorzy UML podjęli próbę zmniejszenia mętliku istniejącego wokół pojęcia agregacji poprzez wprowadzenie jej mocniejszej formy, zwanej **kompozycją.** Związek kompozycji oznacza: część nie może być współdzielona, a jej cykl życiowy zawiera się w cyklu życiowym całości. Część nie może więc istnieć bez całości, jej pojawianie się i znikanie zawsze pozostaje w związku z istnieniem pewnej całości, do której jest przypisana. Klasycznym przykładem związku kompozycji jest związek między *zamówieniem* a *pozycją zamówienia*.

Asocjacja n-arna to asocjacja, której wystąpienia łączą n obiektów, będących instancjami co najwyżej n klas: 3-arna – 3 obiekty (inna nazwa dla tej asocjacji to ternarna), 4-arna – 4 obiekty, itd. Dana klasa może pojawić się na więcej niż jednej pozycji w asocjacji. Asocjacja binarna ze swoją uproszczoną notacją i pewnymi dodatkowymi własnościami (takimi jak możliwość ustalania kierunku nawigowania, kwalifikatorów, związków agregacji czy kompozycji) jest specjalnym rodzajem asocjacji n-arnej, dla n=2. Asocjacja binarna i 2-arna są równoważne, nie istnieje między nimi różnica semantyczna, inny jest tylko sposób reprezentowania. Wymienione powyżej dodatkowe własności asocjacji binarnych są zabronione dla tych asocjacji n-arnych, gdzie n > 2.



Rys. 2. Przykład diagramu klas.

Diagramy dynamiczne

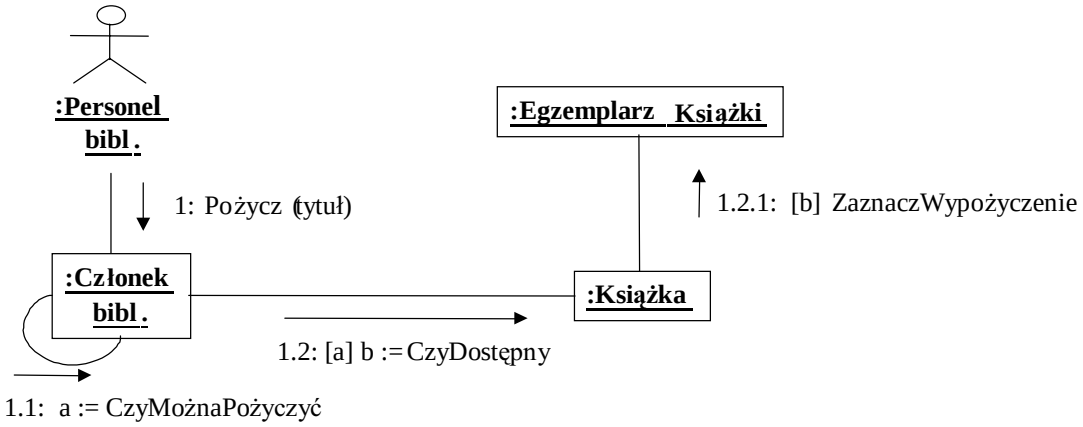
Diagramy: interakcji, stanu, aktywności – środek notacyjny do opisu modelu dynamicznego – mogą stanowić i często stanowią istotne uzupełnienie diagramu klas, ale też w wielu wypadkach ich znaczenie jest pomocnicze i drugorzędne.

Diagramy interakcji: współpracy i sekwencji

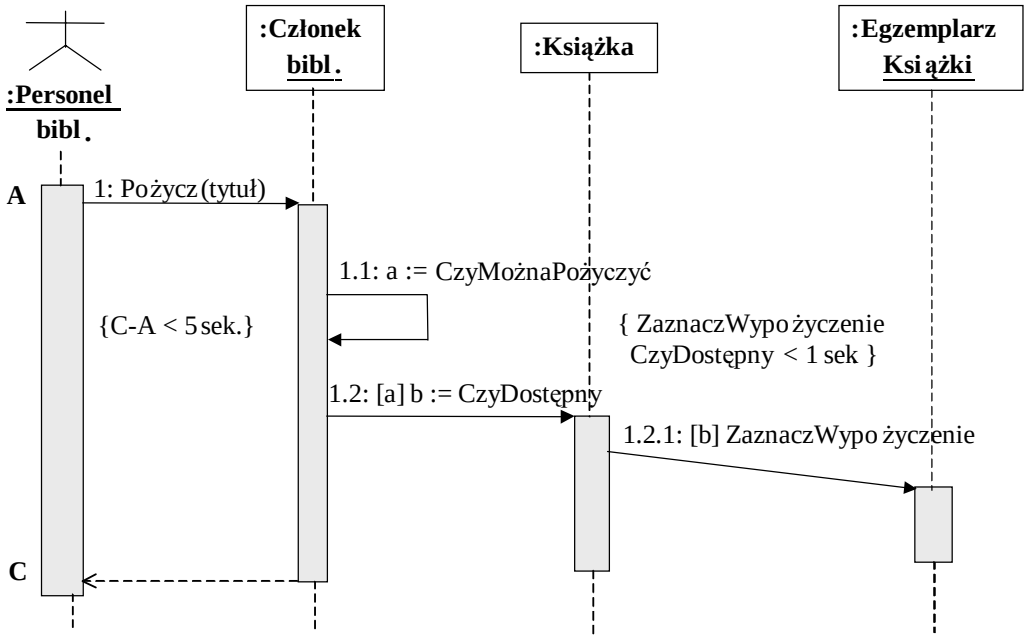
Diagramy interakcji pozwalają na utworzenie opisu współdziałania obiektów systemu podczas

realizacji danego zadania, np.: przypadku użycia czy też jednego konkretnego scenariusza danego przypadku użycia. Nie dla wszystkich przypadków użycia może zachodzić potrzeba konstruowania diagramów interakcji, ale mogą okazać się szczególnie użyteczne np. do komunikacji wewnątrz zespołu projektowego (jak zresztą wszystkie inne rodzaje diagramów) czy też do rozważenia opcjonalnych realizacji w „trudnych przypadkach". Ponadto, niektóre narzędzia CASE potrafią wykorzystać diagramy interakcji do generacji kodu, co może stanowić ważny powód dla ich konstruowania.

UML posiada dwa rodzaje diagramów interakcji: diagramy współpracy (kolaboracji) i diagramy sekwencji. Oba rodzaje diagramów, bazując na danym diagramie klas, pokazują prawie tą samą informację, w nieco inny sposób – diagramy sekwencji umożliwiają specyfikowanie ograniczeń nakładanych na przepływ czasu. Niektóre narzędzia CASE potrafią generować jedne z tych diagramów z drugich. Decyzja, który rodzaj diagramów konstruować, zależy od pożądanego aspektu interakcji.



Rys. 3. Przykład diagramu współpracy.

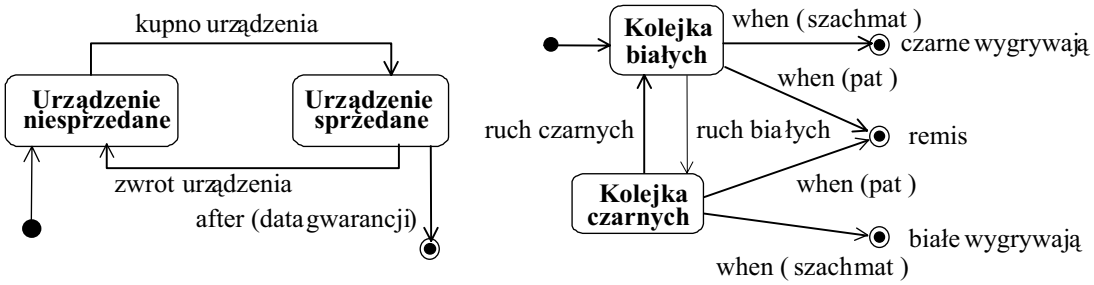


Rys. 4. Przykład diagramu sekwencji.

Diagramy stanu

Obiekt, w świetle swoich własności (unikalna tożsamość, stan i zachowanie) może być traktowany jako automat o skończonej liczbie stanów, czyli pewną maszynę, która może znajdować się w danym momencie w jednym z wyróżnionych stanów, a także może oddziaływać na otoczenie i vice-versa. Maszyna stanu jest grafem, którego wierzchołki stanowią stany obiektu, a łuki opisują przejścia między stanami. Przejście między stanami jest odpowiedzią na zdarzenie. Zwykle, maszyna stanu jest przypisana do klasy i specyfikuje reakcje wystąpień danej klasy na zdarzenia, które do nich przychodzą, stanowiąc w ten sposób model historii życia dla obiektu tej klasy. Takie podejście, separujące obiekt od reszty świata (innych obiektów w systemie czy poza nim), stanowiące podstawę do konstruowania diagramów stanu, pozwala na analizę zachowań pojedynczego obiektu, ale może nie być najlepszym sposobem na zrozumienie działania systemu jako całości.

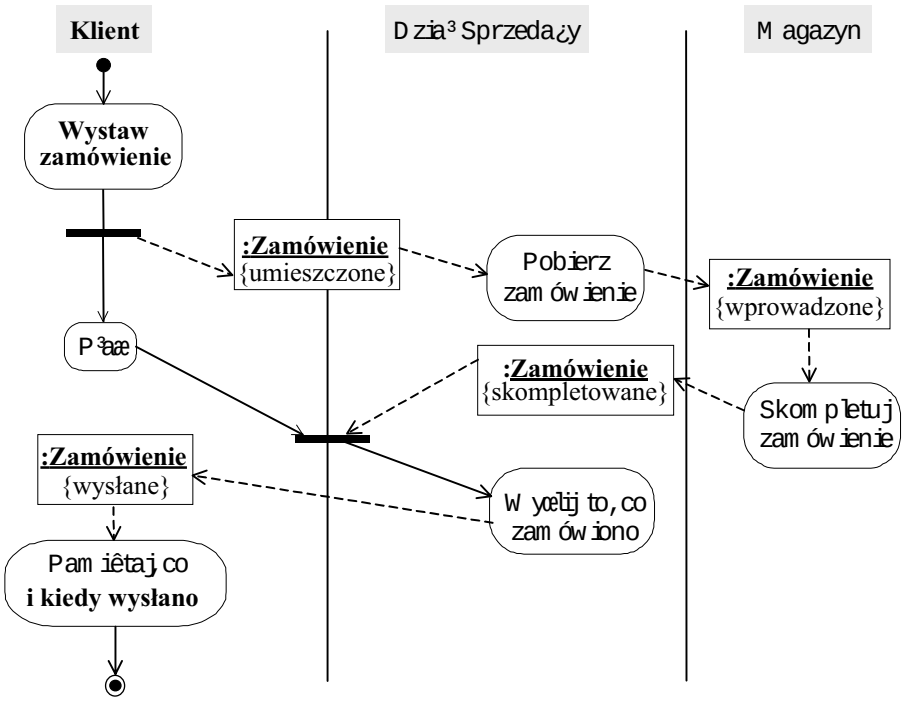
Maszynę stanu można wykorzystać do analizy przypadków użycia, przepływu operacji, czy współpracy obiektów, ale w tym znaczeniu – przepływu sterowania – UML dostarcza inne środki, np. diagramy interakcji czy aktywności.



Rys. 5. Diagramy stanu w postaci: „cykl życia obiektu” oraz „przepływ sterowania”.

Diagramy aktywności

Diagramy aktywności nie posiadające wyraźnego pierwowzoru w poprzednich pracach „trzech przyjaciół” (Jacobson’a, Booch’a i Rumbaugh’a), łączą idee pochodzące z trzech źródeł: diagramów zdarzeń J. Odell’a, technik modelowania stanów i sieci Petriego. Są szczególnie użyteczne przy modelowaniu przepływów operacji czy też w opisie zachowań z przewagą przetwarzania współbieżnego.



Rys. 6. Przykład diagramu aktywności.

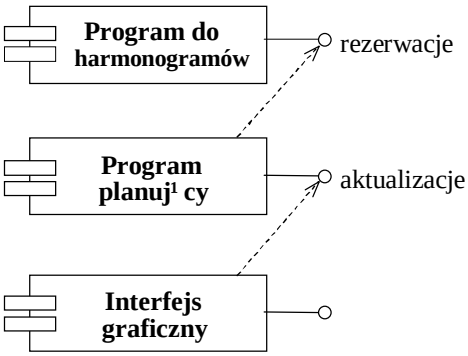
Graf aktywności to maszyna stanu, której podstawowym zadaniem nie jest analiza stanów obiektu, ale modelowanie przetwarzania (przepływów operacji). Stany grafu aktywności, zwane aktywnościami, odpowiadają stanom wyróżnialnym w trakcie przetwarzania, a nie stanom obiektu. Aktywność może być interpretowana różnie, w zależności od perspektywy: jako zadanie do wykonania i to zarówno przez człowieka, jak i przez komputer (z perspektywy pojęciowej) czy też np. jako pojedyncza metoda (z perspektywy projektowej). Podobnie, przejścia między stanami raczej nie są tu związane z nadejściem zdarzenia, ale z zakończeniem przetwarzania wyspecyfikowanego dla danego stanu.

Diagramy implementacyjne: komponentów i wdrożeniowe

Diagramy implementacyjne pokazują niektóre aspekty implementacji SI, włączając w to strukturę kodu źródłowego oraz strukturę kodu czasu wykonania (run-time). UML wprowadza dwa rodzaje takich diagramów: diagramy komponentów pokazujące strukturę samego kodu oraz diagramy wdrożeniowe pokazujące strukturę systemu czasu wykonania.

Diagramy komponentów

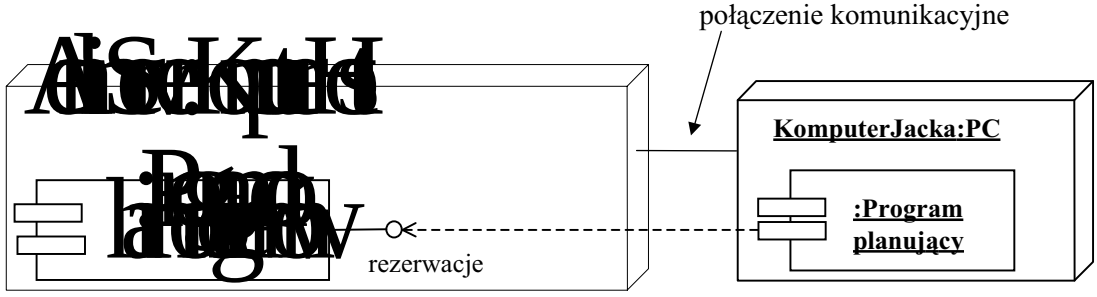
Diagramy komponentów pokazują zależności pomiędzy komponentami oprogramowania: komponentami kodu źródłowego, kodu binarnego oraz kodu wykonywalnego. Komponenty mogą istnieć w różnym czasie: w czasie kompilacji, w czasie konsolidacji czy też w czasie wykonania. Diagram komponentów pokazuje sieć wzajemnych zależności: jest przedstawiany jako graf, gdzie węzłami są komponenty, zaś łuki prowadzą od klienta do dostawcy pewnej informacji.



Rys. 7. Przykład diagramu komponentów.

3.1.1. Diagramy wdrożeniowe

Diagram wdrożeniowy jest grafem, z wierzchołkami zwanymi tu węzłami, połączonymi przez łuki odwzorowywujące połączenia komunikacyjne. Węzły oznaczają fizyczne jednostki, które posiadają co najmniej pamięć, a często i możliwości obliczeniowe. Węzły i połączenia komunikacyjne, mogą być opatrzone stereotypami, np.: «CPU», «pamięć», «urządzenie jakieś». Węzły przechowują wystąpienia obiektów i komponentów.

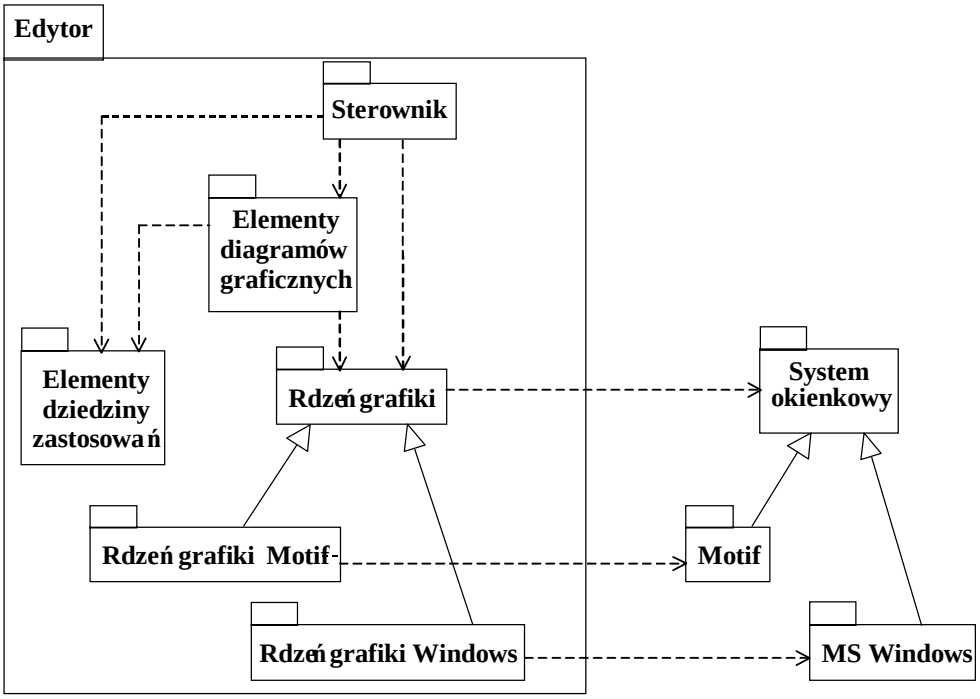


Rys. 8. Przykład diagramu wdrożeniowego.

Diagramy pakietów

Pakiety, wspomagające zarządzaniem projektem, stanowią fragment modelu: oprócz zestawu elementów składowych tego modelu zawierają też sieć ich wzajemnych zależności. Każdy element modelu musi być przypisany do jednego pakietu (home package). Dany model może być więc opisany przez zbiór pakietów. Podział modelu na pakiety jest arbitralny, ale u jego podstaw powinny leżeć racjonalne przesłanki, np. wspólna funkcjonalność, mocne skojarzenia, itp. Pakiety zawierają elementy tzw. wysokiego poziomu, takie jak np.: klasy i ich związki, maszyny stanu, grafy przypadków użycia, itp. To, co jest zawarte w elementach wysokiego poziomu, np. atrybuty, operacje, stany zagnieżdżone, obiekty, linie życia, komunikaty, itp. z reguły nie pojawia się jako bezpośrednia zawartość pakietu. Pakiety mogą być zagnieżdżane i mogą brać udział w związkach dziedziczenia.

Diagramy pakietów są istotne przede wszystkim dla dużych projektów, składających się z wielu modułów funkcjonalnych, ze złożonymi wzajemnymi zależnościami, nie tylko w celu ukrycia mniej istotnych elementów (co zawsze ułatwia zrozumienie), ale także dla ułatwienia podziału prac w ramach grupy osób współpracujących przy konstruowaniu SI.



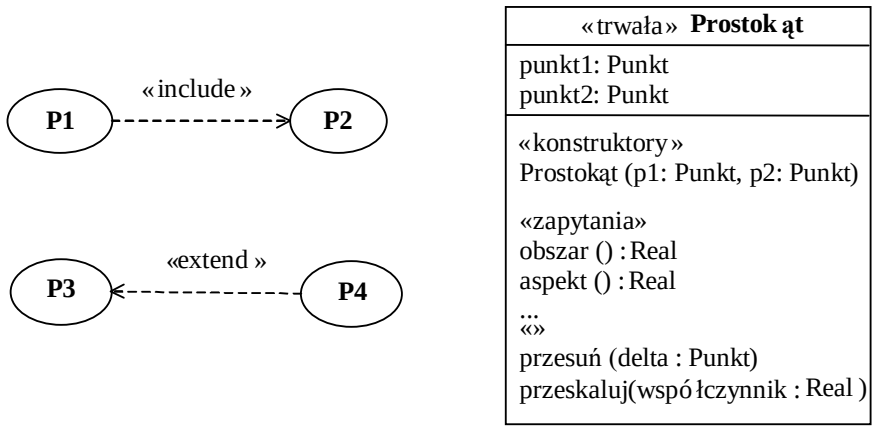
Rys. 9. Przykład diagramu pakietów.

4. Mechanizmy rozszerzalności

UML posiada trzy rodzaje mechanizmów pozwalających na rozszerzenie semantyki metamodelu: stereotypy, wartości etykietowane, ograniczenia. Rozszerzenia z definicji stanowią odstępstwo od standardów UML i w naturalny sposób prowadzą do utworzenia pewnego dialektu UML, co z kolei w efekcie może spowodować problemy z przenaszalnością. Zawsze trzeba rozważać zyski i straty poniesione dzięki korzystaniu z tych mechanizmów, szczególnie wtedy, gdy „stare" – standardowe mechanizmy – pracują wystarczająco dobrze.

Stereotypy

Stereotypy są środkiem umożliwiającym metaklasyfikację elementów modelu. Dla każdego rodzaju elementów modelu, czyli dla każdego elementu metamodelu UML, istnieje lista stereotypów predefiniowanych, ale użytkownik może też definiować stereotypy własne. Dany element modelu (np. relacja między przypadkami użycia, konkretna klasa czy metoda) może być oznaczona przez co najwyżej jeden stereotyp.



Rys. 10. Przykłady zastosowania stereotypów.

Wartości etykietowane

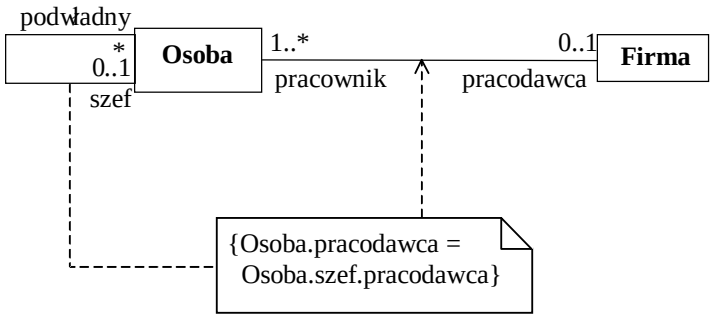
Wartości etykietowane są używane do skojarzenia arbitralnej informacji z pojedynczym elementem modelu:

- Wartość etykietowaną stanowi ciąg znaków o postaci: **słowo kluczowe = wartość**.
- Dowolny łańcuch znaków może być użyty jako słowo kluczowe.
- Są słowa kluczowe predefiniowane, ale użytkownik może też definiować własne.
- Dowolny element modelu może być skojarzony nie tylko z listą wartości etykietowanych, ale w bardziej ogólnym sensie, z łańcuchem własności, w postaci: {dowolny łańcuch znaków}.

Przykład: {autor = „Jan Nowak”, termin zakończenia = „31 Maja 1999”, status = analiza}

Wartości etykietowane są szczególnie przydatne do przechowywania informacji związanych z zarządzaniem projektem (jak w przykładzie powyżej) czy ze szczegółami implementacyjnymi.

Ograniczenia



Rys. 11. Przykład zastosowania ograniczeń.

Ograniczenia specyfikują restrykcje nakładane na elementy modelu. Mogą stanowić wyrażenia języka naturalnego, języka formalnego (np. OCL w UML), mogą przyjmować postać formuły matematycznej lub fragmentu kodu czy też pseudokodu.

5. Podsumowanie

Wadą wszystkich metodyk używanych w procesie konstrukcji SI jest ich arbitralność oraz idealizacja sytuacji projektowych, często niezbyt zgodnych z sytuacjami realnymi. Każda metodyka jest obciążona wysokim stopniem ryzyka, bowiem jej stosowalność jest silnie uzależniona od rodzaju i skali przedsięwzięcia. Udowodnienie, formalne czy empiryczne (na bazie rzeczywistych, dużych projektów), że zastosowanie danej metodyki może znacząco usprawnić proces konstrukcji SI, jest – głównie ze względu na rozmiary współczesnych przedsięwzięć – praktycznie niemożliwe do przeprowadzenia. Nawet najlepsza metodyka nie jest w stanie zapewnić osiągnięcia przez produkt informatyczny odpowiednio wysokiej jakości. Kluczem jest tu zaistnienie zespołu kompetentnych i zaangażowanych osób. Niemniej jednak metodyki są ważne – stanowią drugorzędne, aczkolwiek istotne wspomaganie w procesie konstrukcji SI. W ostateczności, lepsza jest dowolna, lepiej czy gorzej przemyślana metoda, niż żadna.

6. Bibliografia

[Fow1997] M. Fowler, K. Scott. **UML Distilled Applying the Standard Object Modeling Language.** Addison Wesley Longman, Inc., 1997

[Mar2000] Ch. Marshall. **Enterprise Modeling with UML.** Addison Wesley Longman, Inc., 2000

[Mul2000] R.J. Muller. **Bazy danych język Uml w modelowaniu danych.** Wydawnictwo Mikom, 2000

[OMG1999] **Standard OMG UML 1.3**. http://www.omg.org/cgi-bin/doc?formal/00-03-01.pdf, 1999

[Qua2000] T. Quatrani. **Visual Modeling with Rational Rose 2000 and UML.** Addison Wesley Longman, Inc., 2000

[Rum1998] J. Rumbaugh, I. Jacobson, G. Booch. **The Unified Modeling Language Reference Manual.** Addison-Wesley Object Technology Series, 1998

[Ste2000] P. Stevenson, R. Pooley. **Using UML Software Engineering with Objects and Components.** Harlow Pearson Education Ltd., 2000

[Sub1998] K. Subieta. **Obiektowość w projektowaniu i bazach danych.** Akademicka Oficyna Wydawnicza PLJ, 1998