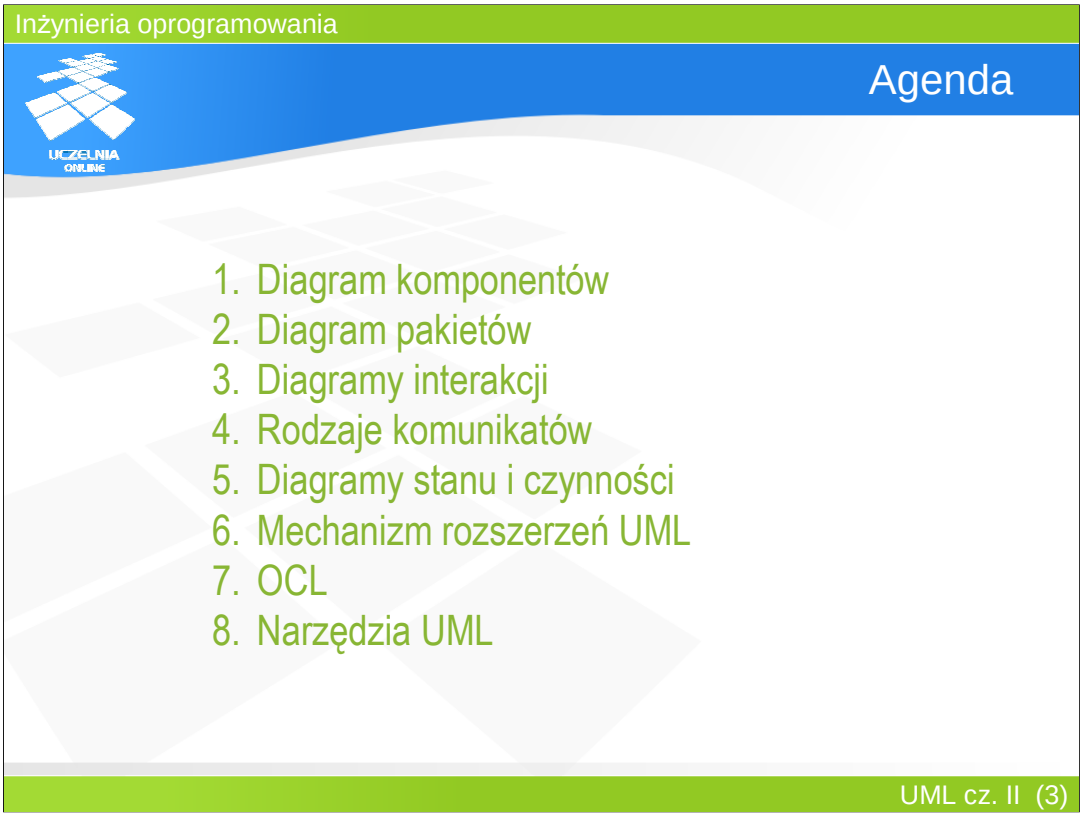

Inżynieria oprogramowania
UML cz. II
Prowadzący: Bartosz Walter
UCZELNIA
ONLINE

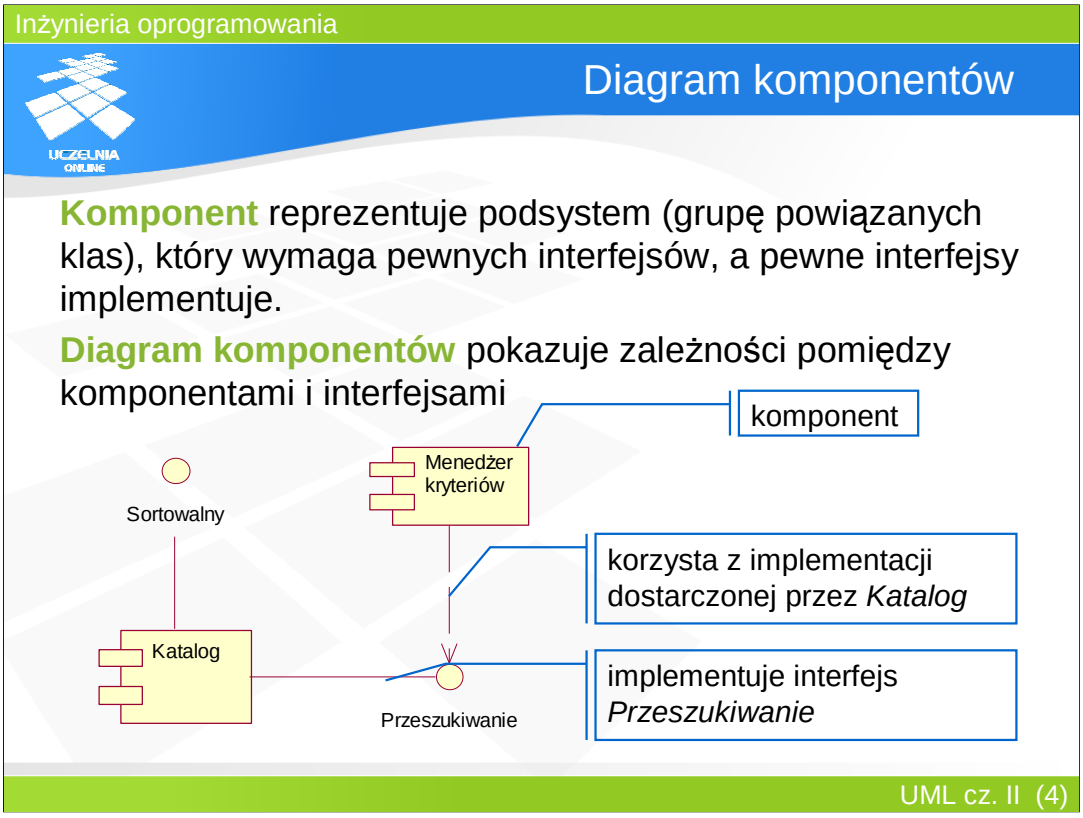
Wykład ten stanowi dalszą część przeglądu najważniejszych elementów języka modelowania UML.



Podczas bieżącego wykładu zostaną przedstawione pozostałe diagramy struktury: diagram komponentów oraz diagram pakietów. Następnie omówione będą diagramy interakcji, prezentujące dynamiczny aspekt modelu systemu, przede wszystkim – diagramy sekwencji i współdziałania. Kolejna część wykładu będzie poświęcona diagramom czynności oraz stanu, opisującym wewnętrzne zachowanie klas, komponentów i podsystemów.

Po zakończeniu przeglądu diagramów przedstawiony zostanie język OCL służący do formalnego opisu ograniczeń w UML. Mimo, że jest nieobowiązkowy, odgrywa on znaczącą rolę przy wykorzystaniu notacji UML jako podstawy do generowania kodu wykonywalnego.

Ostatnią częścią będzie przedstawienie wybranych narzędzi UML, zarówno komercyjnych, jak i darmowych.



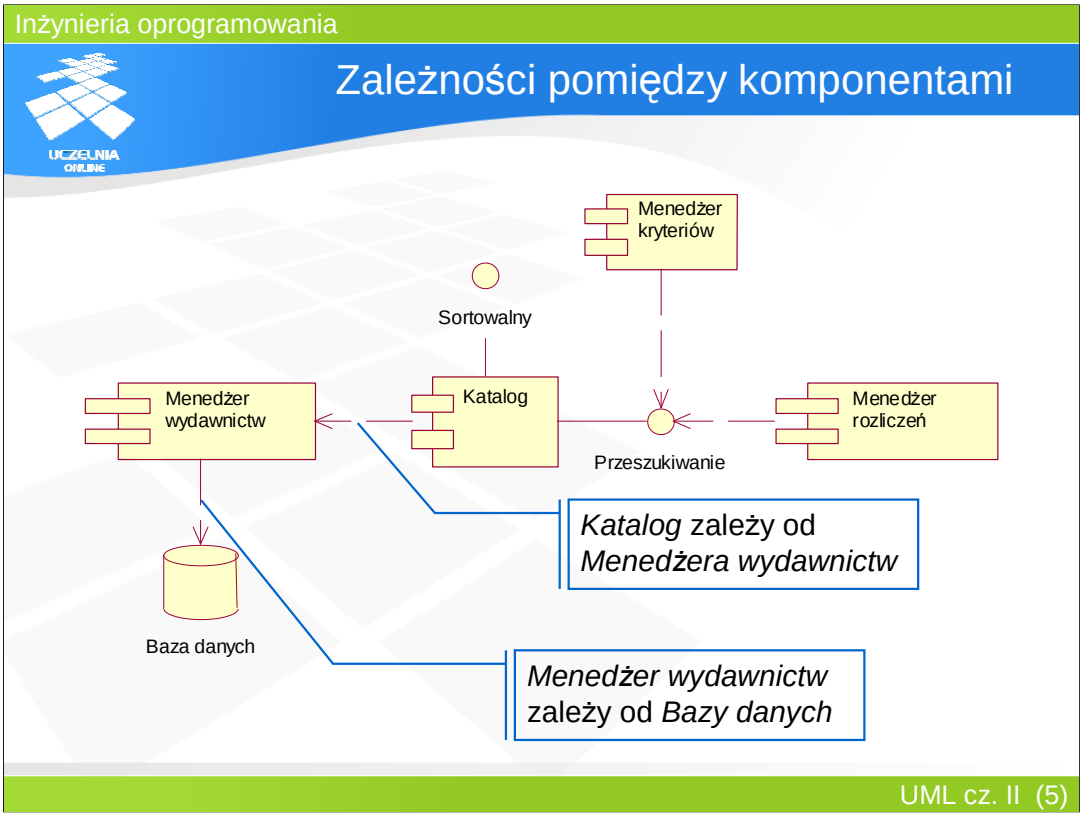
Komponent to wymienny, wykonywalny fragment systemu o hermetyzowanych szczegółach implementacyjnych. Komponenty z natury służą do ponownego wykorzystania poprzez połączenie ich z innymi komponentami, zwykle poprzez ich skonfigurowanie, bez potrzeby rekompilacji.

Funkcjonalność oferowana przez komponent jest dostępna przez interfejsy, które implementuje. Z drugiej strony, komponent może wymagać pewnych interfejsów, które muszą być dostarczone przez inne komponenty.

Diagram komponentów (ang. *component diagram*) przedstawia komponenty, ich interfejsy oraz zależności pomiędzy nimi.

Na powyższym slajdzie komponent Katalog implementuje interfejsy Sortowalny oraz Przeszukiwanie, natomiast interfejs Menedżer kryteriów potrzebuje implementacji interfejsu Przeszukiwanie i korzysta w tym celu z komponentem Katalog.

Przedstawiony diagram stosuje notację znaną z UML 1.x. W przypadku UML 2.0 komponenty są przedstawiane w postaci prostokątów ze stereotypem «component», natomiast interfejsy żądane przez komponent w postaci otwartych "łapek" dopasowanych do "piłeczek" reprezentujących implementowane interfejsy.



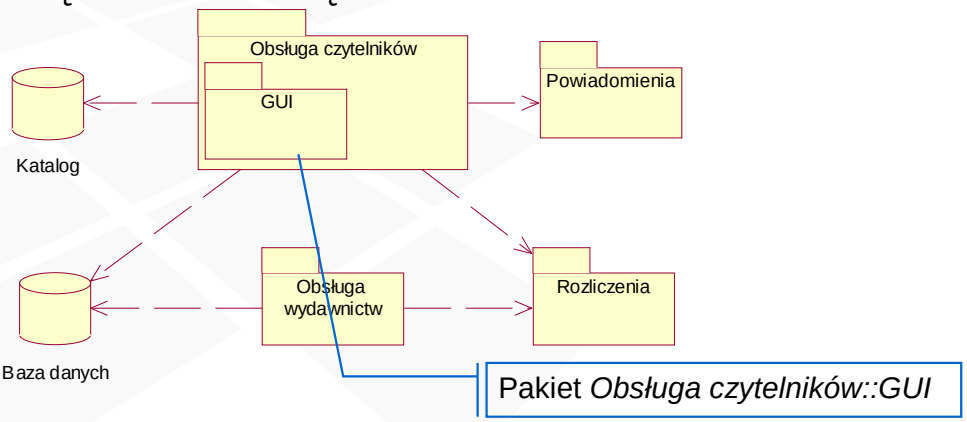
Komponenty są między sobą powiązane relacją zależności, ponieważ wymagają ich do realizacji własnej funkcjonalności. Zależność między A i B oznacza, że komponent A korzysta z komponentu B i zmiana w komponencie B może spowodować konieczność zmiany w A.

Ilość i jakość tych zależności ma duże znaczenie dla oceny jakości modelu i projektu: duża liczba powiązań pomiędzy komponentami, a w szczególności zależności cykliczne, w znacznym stopniu utrudniają wyznaczanie obszarów zmienności i ich hermetyzację, a co za tym idzie – podnoszą koszt pielęgnacji oprogramowania. W odróżnieniu od tego, system o dobrze zdefiniowanych interfejsach komponentów pozwala na ich wymianę bez potrzeby modyfikacji pozostałej części systemu.

Przykład przedstawiony na powyższym slajdzie pokazuje m.in., zależność komponentu Katalog (który implementuje interfejs Sortowalny, jednak nie jest w ten sposób przez żaden inny komponent wykorzystywany) od Menedżera wydawnictw. Komponent Katalog posiada także interfejs Przeszukiwanie, który jest interfejsem wymaganym przez Menedżera rozliczeń i Menedżera kryteriów. Interfejs ten stanowi punkt łączący Katalog z tymi komponentami.

Diagram pakietów

Pakiety w UML służą do przechowywania innych elementów języka, przede wszystkim klas. Dzięki nim można uporządkować strukturę modelu.



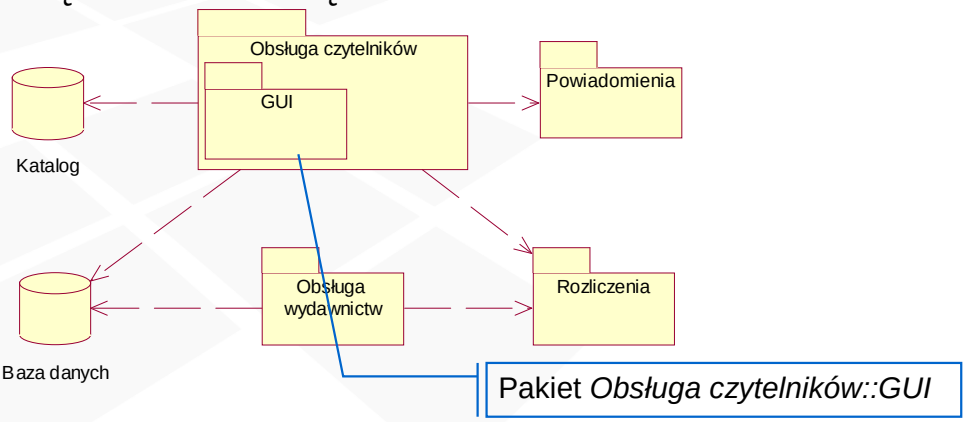
Diagramy pakietów (ang. *package diagrams*) służą do modelowania fizycznego i logicznego podziału systemu. Pakiety są elementem strukturalizującym elementy UML i służą do grupowania ich według dowolnego kryterium. W pakiecie można umieścić praktycznie dowolne elementy: klasy, komponenty, przypadki użycia, a także inne pakiety. W ten sposób przedstawiają one drzewiastą strukturę elementów modelu.

Pakiety doskonale nadają się do wizualizacji podstawowych zależności pomiędzy częściami systemu, dzięki czemu łatwo ocenić jakość i stopień powiązań pomiędzy nimi. Dobra struktura pakietów, w której zależności są jasno uporządkowane oraz nie występują (lub występują tylko na niskim poziomie) zależności cykliczne, wspiera późniejszą rozbudowę systemu. W szczególności przydają się w dużych aplikacjach, podzielonych na wiele podsystemów, ponieważ w prosty sposób obrazują podstawowe zależności pomiędzy nimi.

Pakiet tworzy także jednostkę hermetyzacji: elementy z pakietu odwołują się do elementów zewnętrznych posługując się ich pełnymi kwalifikowanymi (zawierającymi nazwy pakietów) nazwami, zgodnie z ich zakresem widoczności, natomiast wewnątrz pakietu elementy mogą odwoływać się do siebie bezpośrednio.

Diagram pakietów

Pakiety w UML służą do przechowywania innych elementów języka, przede wszystkim klas. Dzięki nim można uporządkować strukturę modelu.



Elementy wewnątrz pakietu mogą mieć jeden z dwóch poziomów widoczności: prywatny lub publiczny. Elementy publiczne są widziane i mogą być użyte poza własnym pakietem, natomiast prywatne – nie. Aby elementy pakietu mogły odwołać się do elementów prywatnych z innego pakietu, muszą go importować. Oznacza to, że elementy te stają się dla importującego pakietu widoczne. Import pakietu oznaczany jest zależnością ze słowem kluczowym «import».

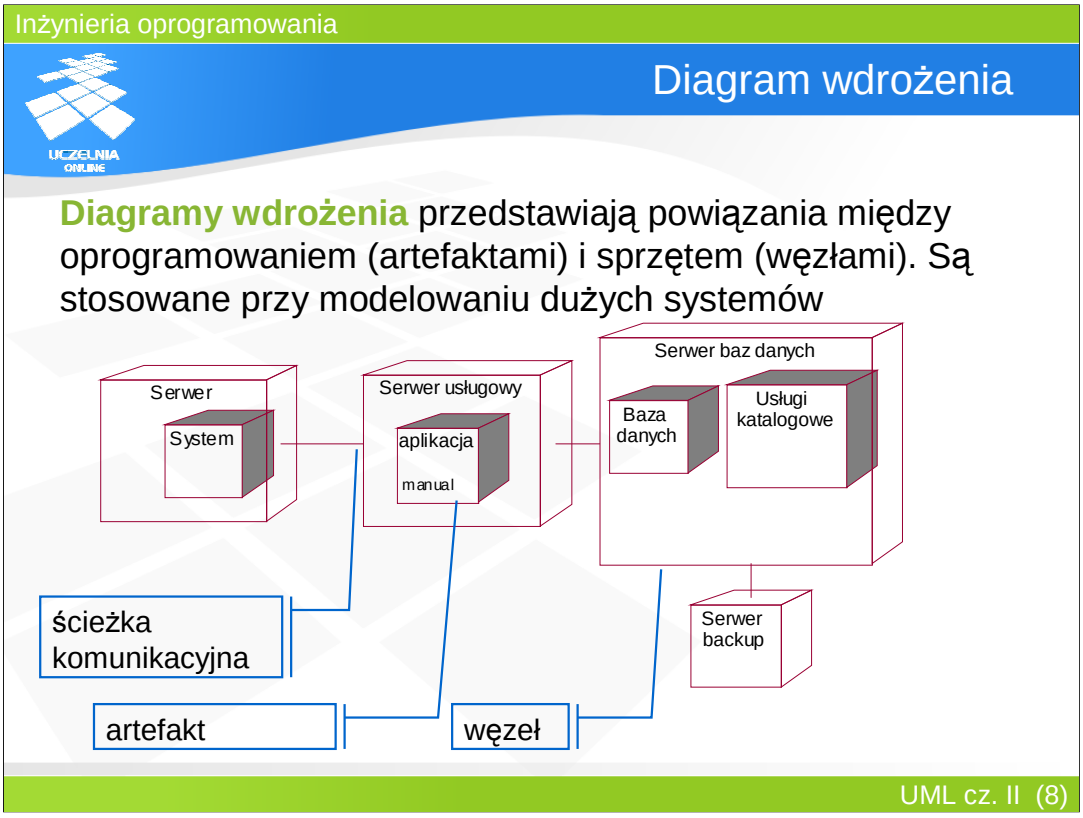
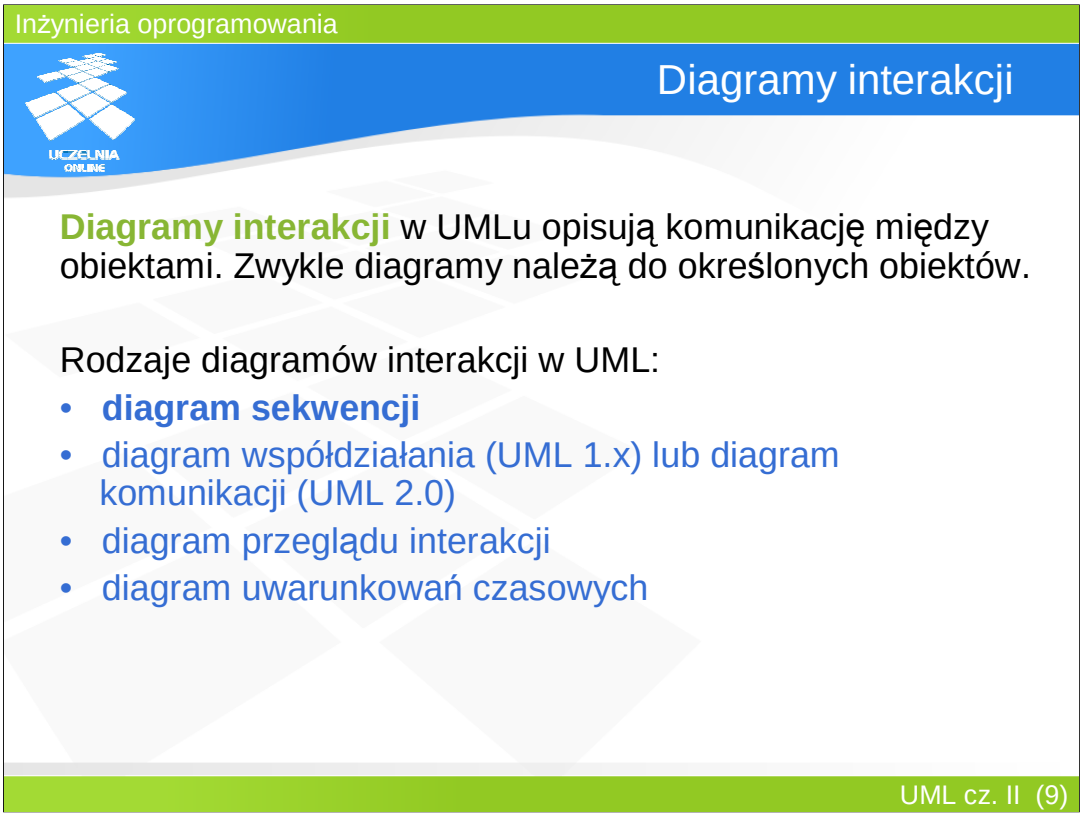


Diagram wdrożenia (ang. *deployment diagram*) odzwierciedla fizyczną strukturę całego systemu, z uwzględnieniem oprogramowania i sprzętu. Jednostki oprogramowania są reprezentowane przez artefakty (czyli skompilowane wersje komponentu, który można uruchomić), dane i biblioteki. Stronę sprzętową reprezentują węzły, czyli poszczególne urządzenia obliczeniowe, komunikacyjne i przechowujące, powiązane ścieżkami komunikacyjnymi (np. połączeniem TCP/IP).

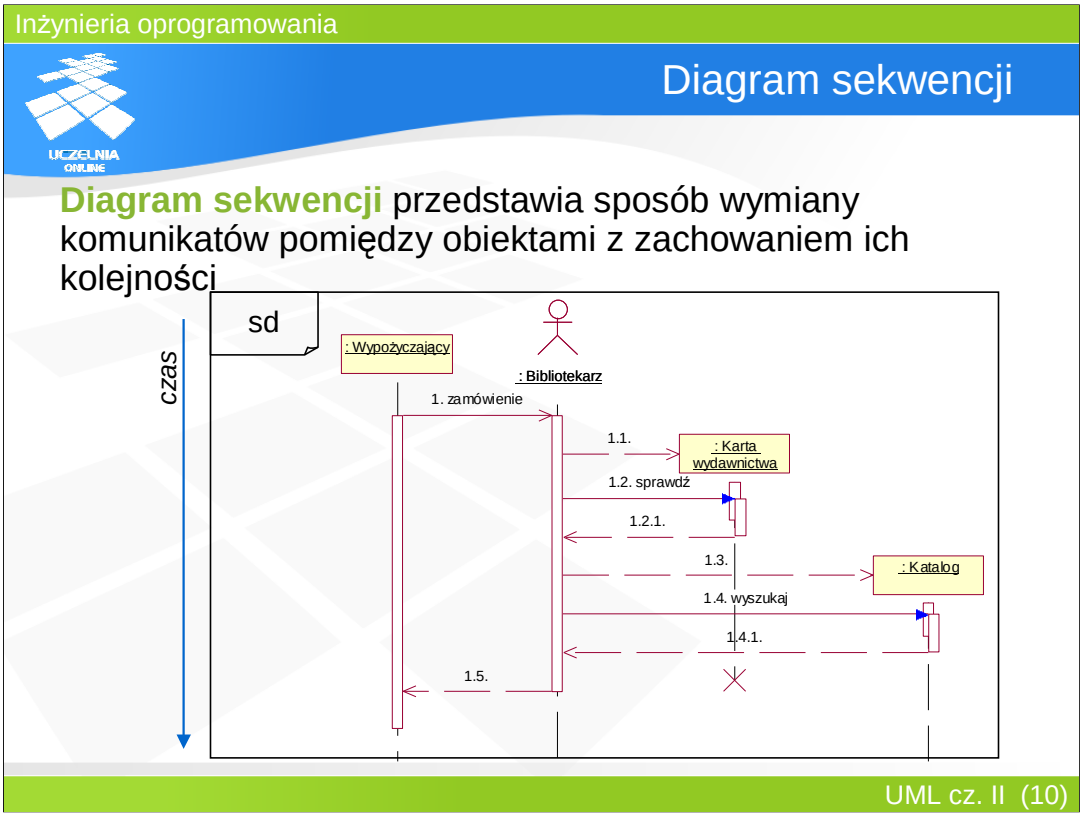
Diagramy te są rzadko używane przy modelowaniu mniejszych i średnich systemów, dlatego zwykle ich rola jest ograniczona. Ponieważ posługują się zaledwie kilkoma symbolami, dlatego kluczową rolę odgrywają stereotypy nadawane poszczególnym elementom. Pozwalają one doprecyzować znaczenie i funkcję oprogramowania oraz sprzętu.

Diagramy wdrożenia istotną rolę odgrywają przy wdrażaniu dużych, rozproszonych systemów.



Opisywanie jedynie struktury logicznej systemu jest zwykle niewystarczające. Konieczne jest także pokazanie, jak obiekty ze sobą się komunikują, jakie informacje przesyłają, aby dostarczyć określoną funkcjonalność. Wszystkie wersje UML posiadają bogaty wachlarz diagramów dotyczących tego aspektu modelowania.

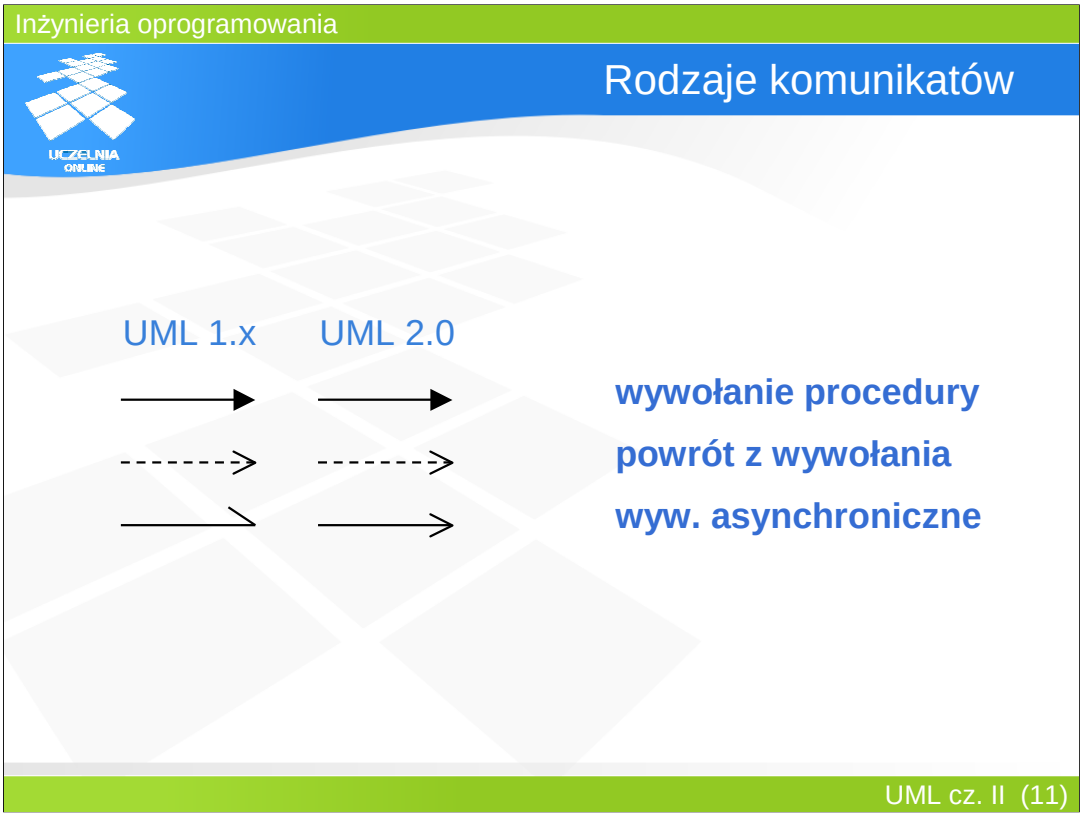
Spośród nich najbardziej znanym i najczęściej wykorzystywanym jest diagram sekwencji, pokazujący przepływ komunikatów między obiektami w kontekście czasu. Pozostałe diagramy – komunikacji, przeglądu interakcji czy uwarunkowań czasowych – zwykle odgrywają mniejszą rolę, ale warto o nich wspomnieć.



Diagramy sekwencji (ang. *sequence diagrams*) intuicyjnie prezentują kolejność wywołań operacji, przepływ sterowania pomiędzy obiektami oraz szablon realizowanego algorytmu. Pomijają natomiast całkowicie aspekt dostępu i operacji na danych, związany z komunikacją. Uczestnikami diagramów sekwencji są obiekty, opisane nazwą obiektu i jego klasą, które wymieniają między sobą komunikaty.

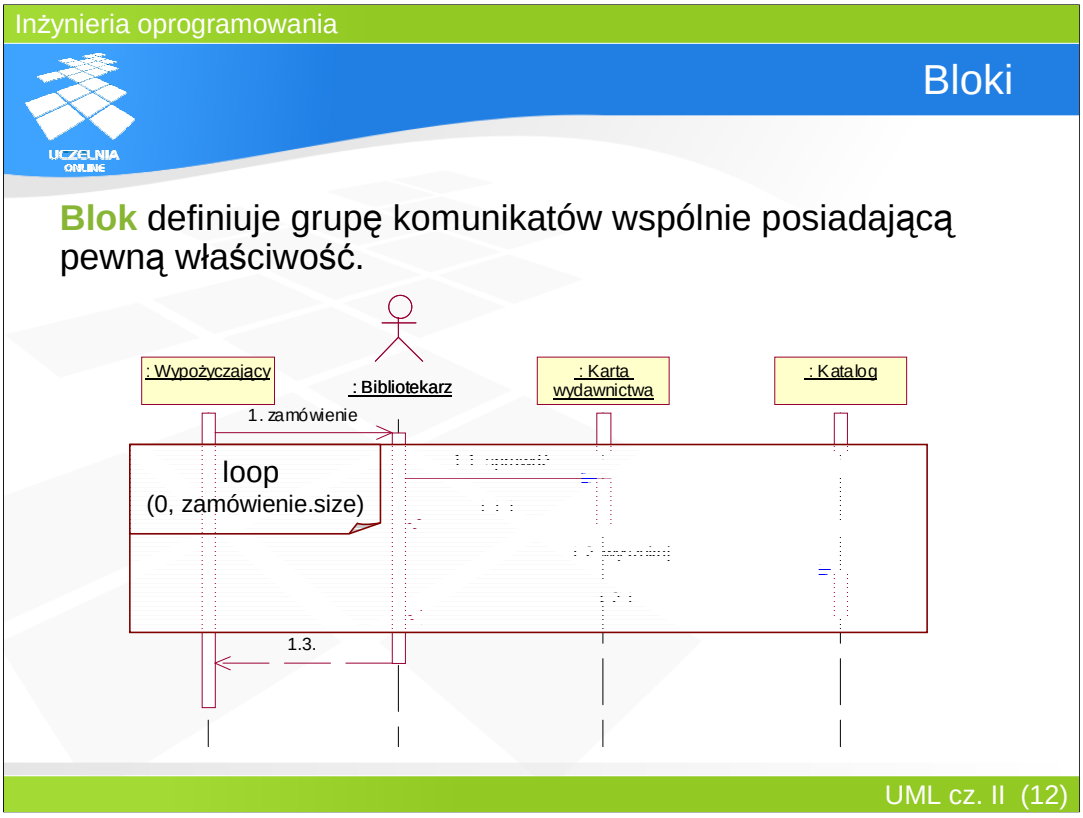
Diagram sekwencji jest zapisany w prostokącie oznaczonym operatorem sd (od angielskiej nazwy diagramu) i składa się z pionowych linii życia (ang. *lifelines*) poszczególnych obiektów uczestniczących w interakcji oraz wymienianych między nimi komunikatów (wywołań operacji). Białe prostokąty umieszczone na linii życia obiektu oznaczają, że obiekt jest zajęty wykonywaniem pewnej czynności (natomiast nie mają bezpośredniego związku z istnieniem obiektu). Czas jest reprezentowany w postaci pionowej osi diagramu.

Linia życia obiektu to czas, w którym konkretna instancja obiektu jest w stanie przyjmować komunikaty i wysyłać je. Innymi słowy, obejmuje ona czas istnienia obiektu w systemie. Obiekt jest tworzony poprzez wysłanie do niego komunikatu-konstruktora (Bibliotekarz tworzy obiekt klasy Karta Wydawnictwa), natomiast niekoniecznie jest fizycznie usuwany na końcu linii życia – raczej przestaje być istotny. Fizycznie usunięcie obiektu można wprost oznaczyć jako znak X na linii życia (na przykład obiekt Karta wydawnictwa).



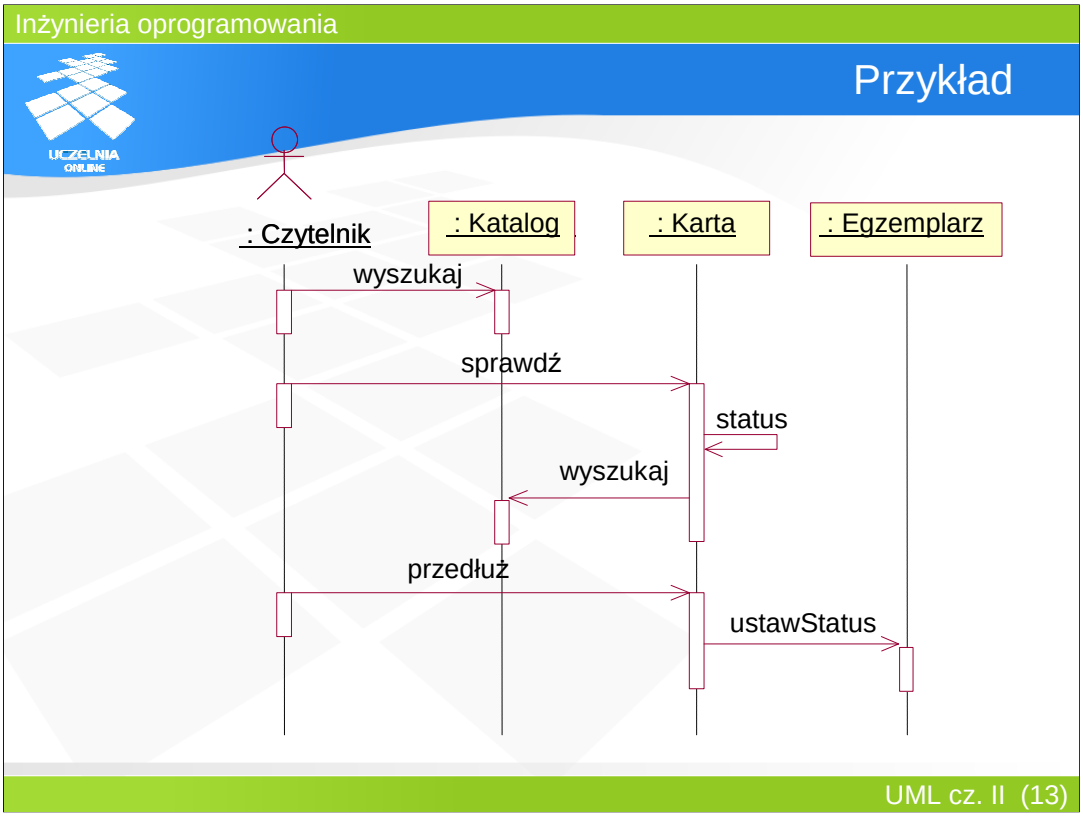
Komunikat to forma kontaktu pomiędzy obiektami, której efektem ma być podjęcie przez docelowy obiekt pewnej akcji. Otrzymanie komunikatu przez obiekt wiąże się z wykonaniem przez niego jego własnego kodu lub wysłaniem kolejnego komunikatu do innego obiektu w celu wykonania przez niego pewnej akcji.

Komunikaty w UML są reprezentowane przez strzałki łączące linie życia poszczególnych obiektów. Każdy komunikat wewnątrz interakcji opatrzony jest kolejnym numerem, co pozwala na łatwe śledzenie jej przebiegu. Istnieją trzy podstawowe komunikaty, jakie mogą zostać wymienione pomiędzy obiektami: wywołanie procedury, powrót z niej oraz wywołanie asynchroniczne.



Bardzo często zachodzi konieczność wskazania specjalnej własności pewnej części interakcji, np. oznaczenie sekcji krytycznej czy zwyczajnej pętli. Na diagramach sekwencji taką grupę operacji obejmuje się prostokątem, w którego lewym górnym narożniku, w pięciokącie umieszcza się słowo kluczowe lub opis określający znaczenie danego bloku (tzw. operator interakcji), np.:

- **alt** (od *alternative*) – określający warunek wykonania bloku operacji, odpowiadający instrukcji *if-else*; warunek umieszcza się wówczas wewnątrz bloku w nawiasach kwadratowych
- **opt** (od *optional*) – reprezentujący instrukcję *if* (bez *else*)
- **par** (od *parallel*) – nakazujący wykonać operacje równolegle
- **critical** – oznaczający obszar krytyczny
- **loop** – definiujący pętlę typu *for* (o określonej z góry liczbie iteracji) lub *while* (wykonywanej dopóki pewien warunek jest prawdziwy)



Slajd przedstawia przykładowy diagram sekwencji dla przypadku użycia 'Przedłużenie wypożyczenia'. Czytelnik wyszukuje w Katalogu swoją kartę biblioteczną. Po jej znalezieniu sprawdza jej status. Sprawdzenie statusu polega m.in. na wyszukaniu w katalogu wszystkich egzemplarzy książek, które Czytelnik aktualnie ma wypożyczone. Następnie Czytelnik zleca przedłużenie określonego egzemplarza, co powoduje aktualizację statusu w klasie Egzemplarz.

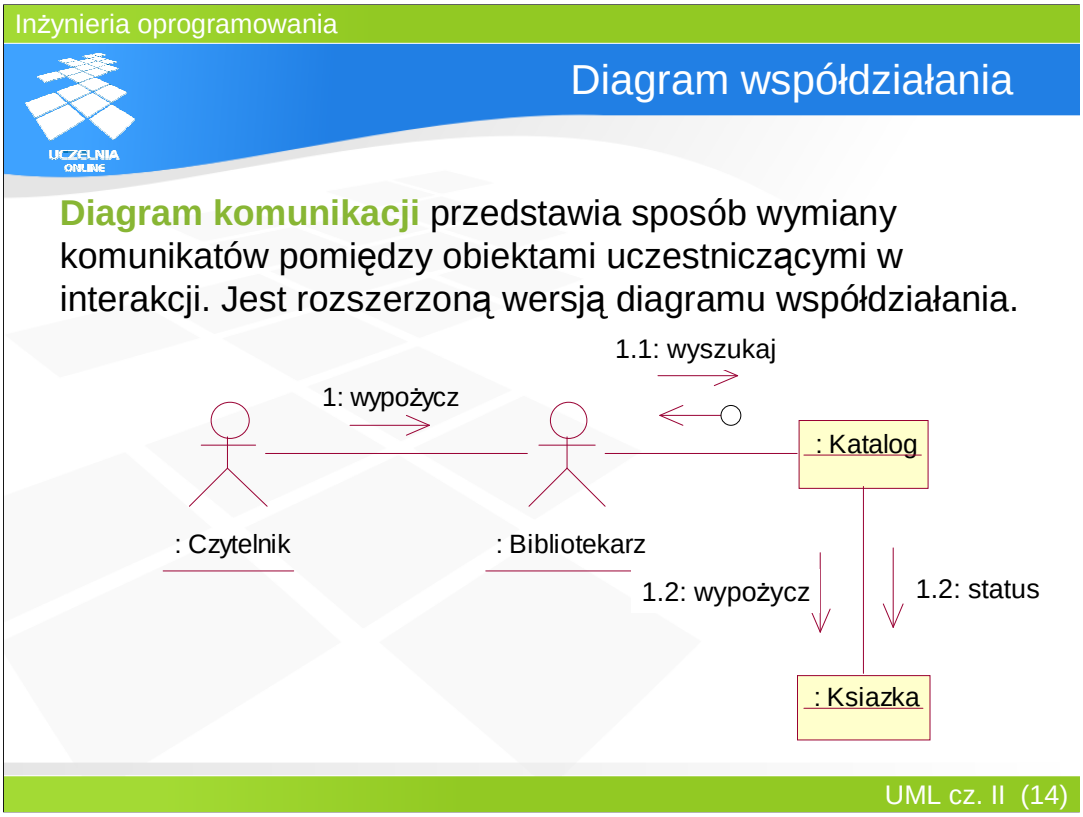


Diagram komunikacji (ang. *communication diagram*) jest rozszerzoną i przemianowaną wersją diagramu współdziałania znanego z UML 1.x. Skupia się on na obiektach wchodzących w skład interakcji i wymienianymi przez nie komunikatach, natomiast w mniejszym stopniu niż diagram sekwencji (choć nadal obecnym) wskazuje na aspekt czasowy. Z tego powodu obiekty na diagramie komunikacji są umieszczone tak, aby łatwo można było opisać ich relacje pomiędzy sobą. Komunikacje są przedstawiane za pomocą linii łączących obiekty, natomiast przesyłane między obiektami komunikaty i dane są umieszczane obok tych linii. Każdy komunikat jest opatrzony etykietą liczbową, wskazującą na kolejność ich wysyłania. Etykieta ta ma postać liczb oddzielonych kropkami. W przypadku rozdzielenia sterowania każdy krok powoduje dodanie do etykiet kroków następnych kolejnych pól z liczbami, np. krok 2 powoduje utworzenie kroków 2.1, 2.2 leżących bezpośrednio za nim. Krok 2.1 posiada kroki 2.1.1 i 2.1.2, itd.

W odróżnieniu od diagramów sekwencji, diagramy komunikacji nie mogą przekazać wielu informacji dotyczących interakcji, np. bloków komunikatów. Z drugiej strony jednak prezentują rzeczywiste powiązania obiektów i ich relacji, co może ułatwić zrozumienie interakcji.

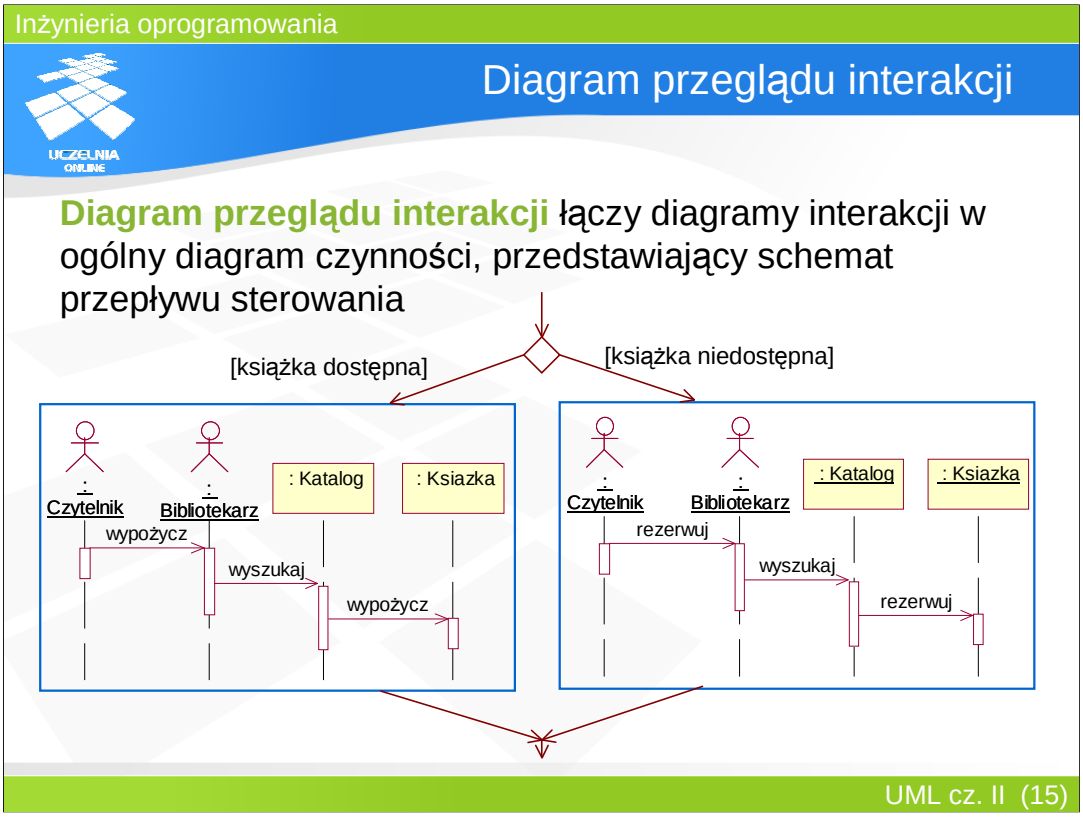
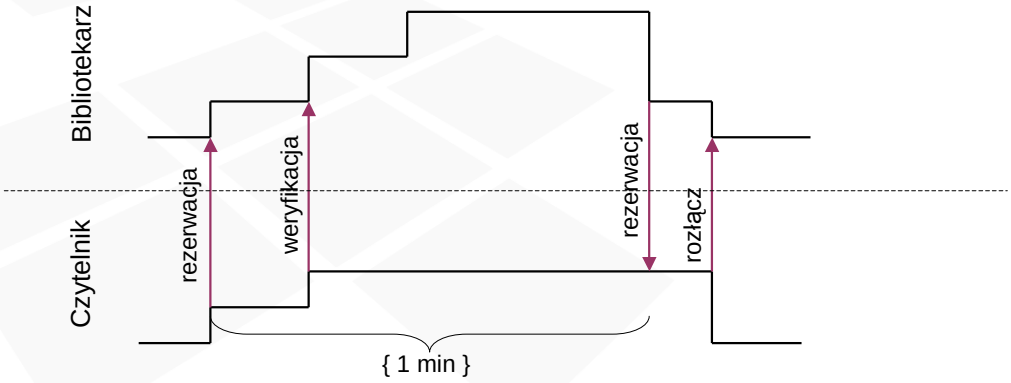


Diagram przeglądu interakcji (ang. *interaction overview diagram*) służy do przedstawiania ogólnego przepływu sterowania w interakcjach pomiędzy obiektami, korzystając z uproszczonej notacji diagramu czynności. Na jednym diagramie pokazany jest przepływ sterowania pomiędzy interakcjami pokazanymi w postaci innych diagramów, np. sekwencji. Diagram ten może korzystać z większości elementów obecnych na diagramach czynności: punktu decyzyjnego, pętli, rozwidlenia i synchronizacji, etc.

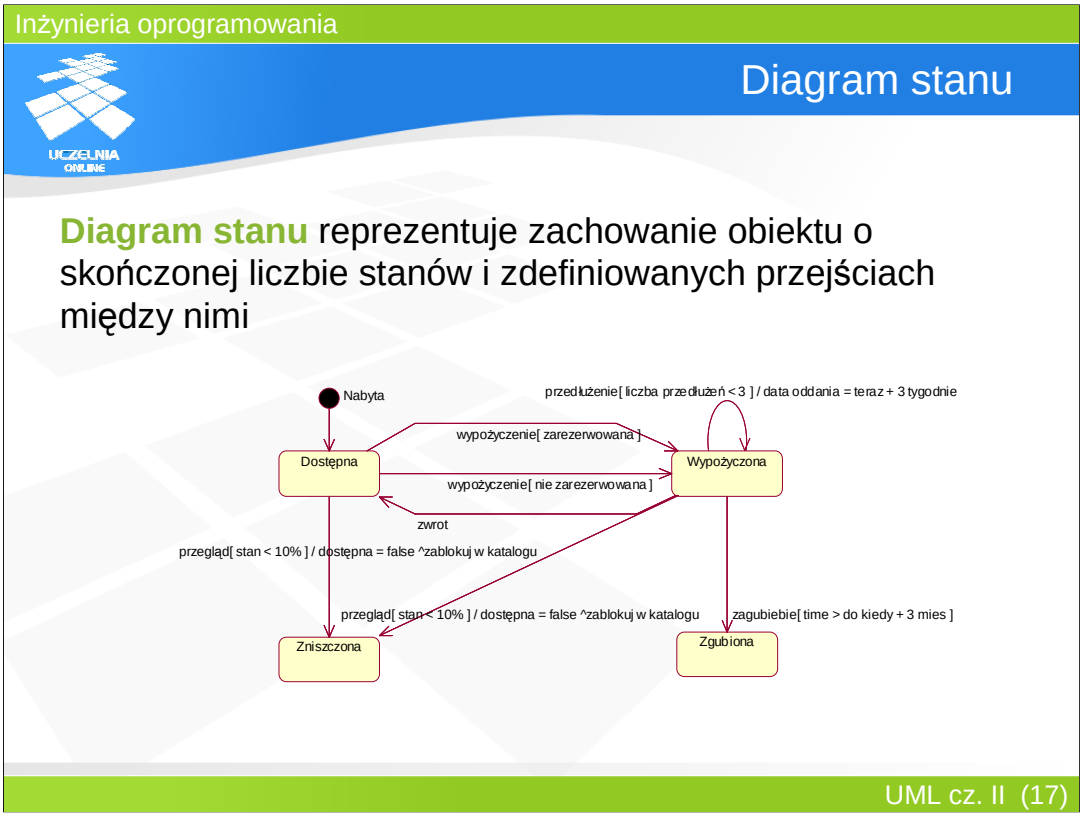
Diagram uwarunkowań czasowych

Diagram uwarunkowań czasowych koncentruje się na zależnościach czasowych w interakcji pomiędzy grupą obiektów



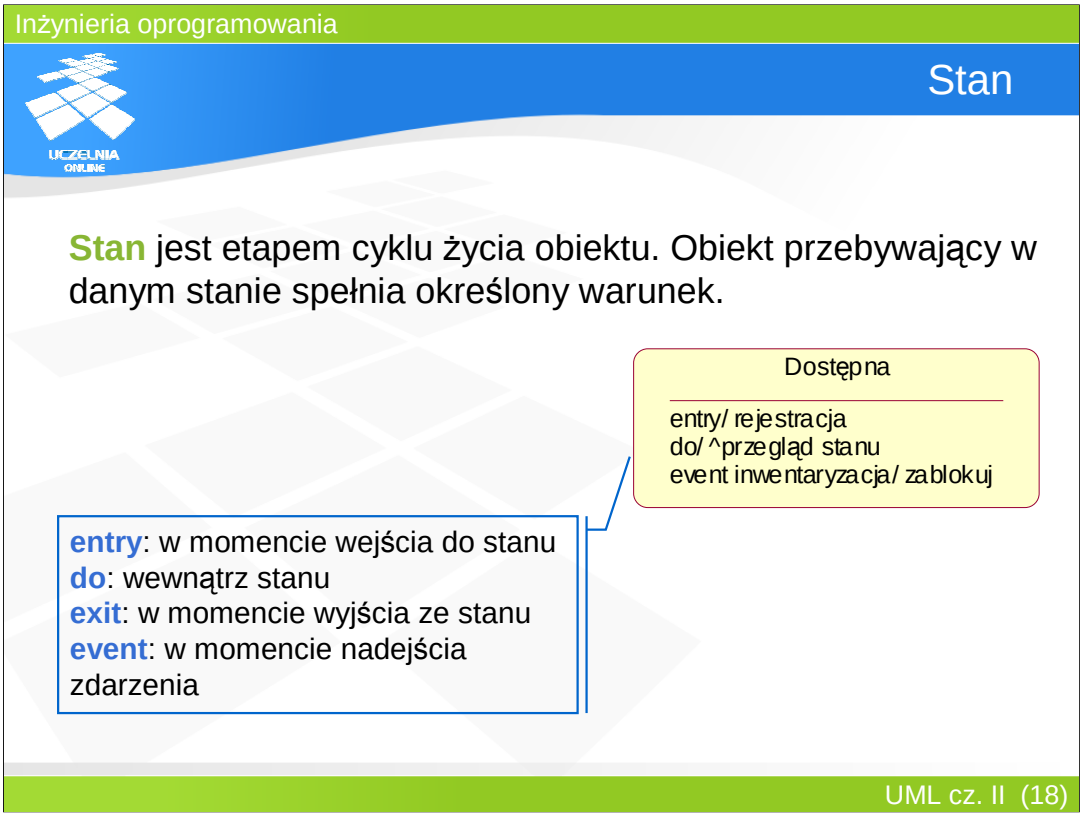
Diagramy uwarunkowań czasowych (ang. *timing diagrams*) są specjalną formą diagramów interakcji przeznaczoną niemal wyłącznie do prezentowania zależności związanych z czasem wykonywania operacji przez obiekt lub grupę obiektów. Linia czasu jest reprezentowana przez poziomą oś diagramu, natomiast oś pionowa przedstawia kolejne obiekty uczestniczące w interakcji oraz ich zmieniające się stany wewnętrzne. Odległości pomiędzy momentami zmian stanów wyznaczają uwarunkowania czasowe.

Diagram ten ma duże znaczenie w modelowaniu systemów czasu rzeczywistego.



Diagramy stanu (ang. *state machine diagram*) reprezentują zachowanie systemu lub jego elementu (zwykle klasy), a w szczególności zmiany tego zachowania.

Podstawowymi elementami diagramu są stany obiektu połączone strzałkami przejść. Obiekt, reagując na nadchodzące zdarzenia, jeżeli spełnione są określone warunki, zmienia swój stan i położenie na diagramie stanu.



Pojedynczy stan reprezentuje moment w zachowaniu obiektu, w którym pewien warunek jest prawdziwy.

Stany są reprezentowane przez prostokąty z zaokrąglonymi narożnikami. Każdy stan ma swoją nazwę.

Ze stanem mogą być związane pewne akcje, wykonywane w określonym momencie:

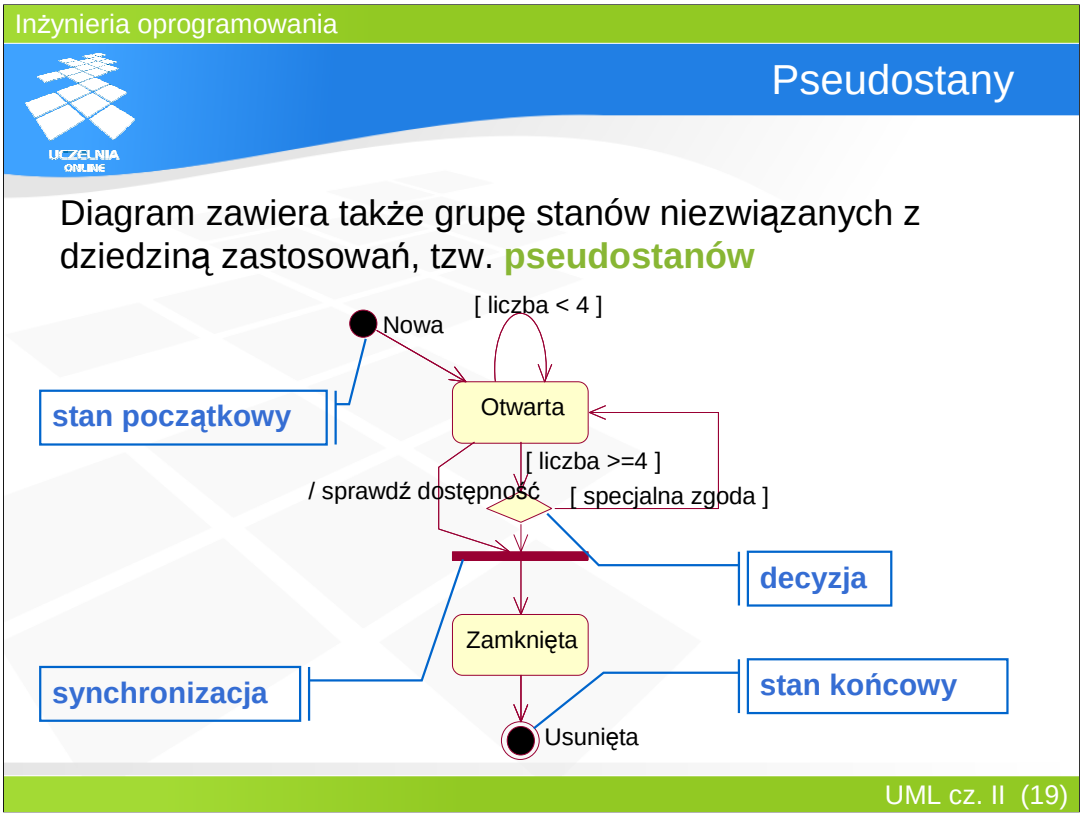
•**entry**: jest akcją wykonywaną w momencie gdy obiekt przyjmuje dany stan; akcja ta jest wykonywana jeden raz i niepodzielnie

•**do**: jest akcją wykonywaną nieprzerwanie w czasie, gdy obiekt przebywa w tym stanie

•**exit**: oznacza (analogicznie do entry) moment opuszczenia stanu; podobnie, akcja taka jest wykonywana tylko raz.

•**event**: reprezentuje akcję wykonywaną w momencie nadejścia zdarzenia określonego typu

Wykonanie każdej z tych akcji może również generować zdarzenie.



Obok stanów reprezentujących własności wynikające z dziedziny zastosowań (np. Dostępna czy Wypożyczona w przypadku Książki), UML definiuje grupę innych stanów pomocniczych, które pozwalają na łatwiejsze modelowanie rozmaitych maszyn stanowych. Są to tzw. pseudostany:

•**początkowy**, który reprezentuje obiekt w momencie jego utworzenia. Każdy diagram stanu może zawierać tylko jeden taki stan. Do stanu początkowego nie dochodzą żadne przejścia.

•**końcowy**, który reprezentuje usunięcie obiektu z systemu. Stan ten jest opcjonalny (nie wszystkie obiekty są usuwane), w systemie także może występować wiele różnych stanów końcowych. Ze stanów końcowych nie można przejść do innych stanów.

•**decyzja**, przedstawiająca wybór pomiędzy dwiema wartościami logicznymi pewnego wyrażenia. Warto zauważyć, że odpowiednio korzystając z warunków przejść można pominąć ten pseudostan, jednak często jego użycie zwiększa czytelność modelu

•**złączenie/rozwidlenie** – powoduje synchronizację stanów (wszystkie dochodzące do niego przejścia muszą być wykonane)

•**historia** (reprezentowana przez literkę H umieszczoną w okręgu wewnątrz stanu) – zapewnia możliwość zapamiętania poprzedniego stanu obiektu i przywrócenie go.

Stany są powiązane ze sobą przejściami. Przejścia definiują warunki, jakie muszą zaistnieć, aby obiekt zmienił swój stan z źródłowego na docelowy. Formalnie opis przejścia składa się z czterech elementów:

•**wyzwalacza** (ang. *trigger*) – zdarzenia, które może spowodować przejście i zmianę stanu

•**dozoru** (ang. *guard condition*) – warunku, jaki musi być spełniony, aby przejście zostało wykonane; warunek ten jest ewaluowany w momencie pojawienia się wyzwalacza

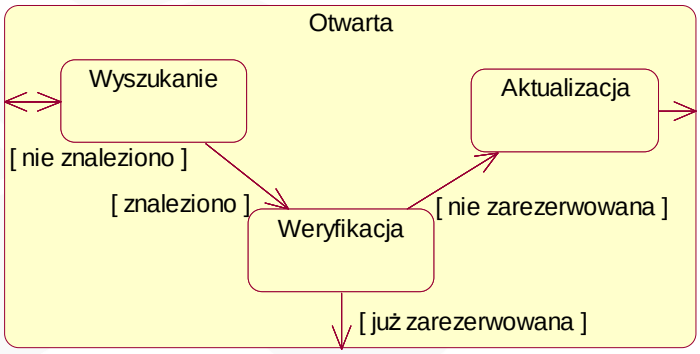
•**akcji** (ang. *action*) – operacji wykonywanej w momencie przejścia ze stanu do stanu; nawet jeżeli akcja przejścia jest złożona z wielu akcji elementarnych, jest ona wykonywana niepodzielnie

•**zdarzenia** (ang. *event*) – wysyłanego w momencie wykonania przejścia.

W podanym przykładzie, reprezentującym dwa stany klasy Książka, zdarzenie Przegląd może spowodować zmianę stanu z Dostępnej na Zniszczoną, jeżeli jej stan zostanie oceniony na mniej niż 10%. Efektem przejścia będzie zmiana atrybutu dostępna na *false* oraz wysłanie zdarzenia zablokowania książki w katalogu.

Stany złożone

Stany złożone posiadają wewnętrzną maszynę stanów. Wejście do stanu jest jej stanem początkowym, a wyjście – końcowym.



Dotychczas była mowa o stanach prostych. Są one niepodzielne – znalezienie się obiektu w takim stanie ma zawsze taki sam efekt i pomija ewentualne zmieniające się zewnętrzne okoliczności.

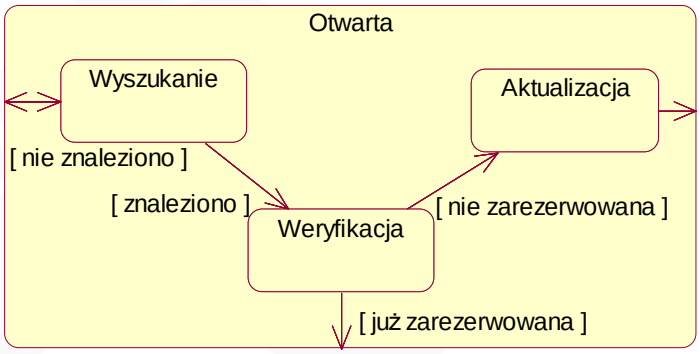
W niektórych sytuacjach wewnątrz stanu można jednakże wyróżnić podstany. Innymi słowy, wewnątrz stanu znajduje się inny diagram stanu.

Diagram podstanów jest przetwarzany w sposób zbliżony do zwykłego diagramu stanu. Jednak w ogólnym przypadku stan złożony dopuszcza także istnienie podstanów współbieżnych, co oznacza, że obiekt znajdując się w jednym stanie jednocześnie znajduje się w kilku podstanach. Wówczas podstany równoległe tworzą niezależne regiony wewnątrz stanu zewnętrznego, w których przejścia następują niezależnie od siebie.

Wejście do stanu powoduje także wejście wszystkich podstanów początkowych we wszystkich regionach. Następnie przejścia są realizowane równolegle i niezależnie we wszystkich regionach, aż do podstanów końcowych. Przejście do stanu końcowego we wszystkich regionach powoduje uruchomienie zdarzenia zakończenia stanu i skojarzonych z nim wyzwalaczy.

Stany złożone

Stany złożone posiadają wewnętrzną maszynę stanów. Wejście do stanu jest jej stanem początkowym, a wyjście – końcowym.



Przykład dotyczy stanu Otwarta, reprezentującego otwarty stan Rezerwacji książek. Rezerwacja może objąć do 4 książek jednocześnie. Stan Rezerwacji pozostaje otwarty w trakcie dodawania kolejnych książek, jednak wyróżniono w nim podstany: Wyszukanie informacji o książce, Weryfikację, czy danej książki już wcześniej nie zarezerwowano oraz Aktualizację danych Rezerwacji. Wszystkie podstany prowadzą do opuszczenia stanu przez Rezerwację, co jest związane np. z próbą dodania do niej nowej książki.

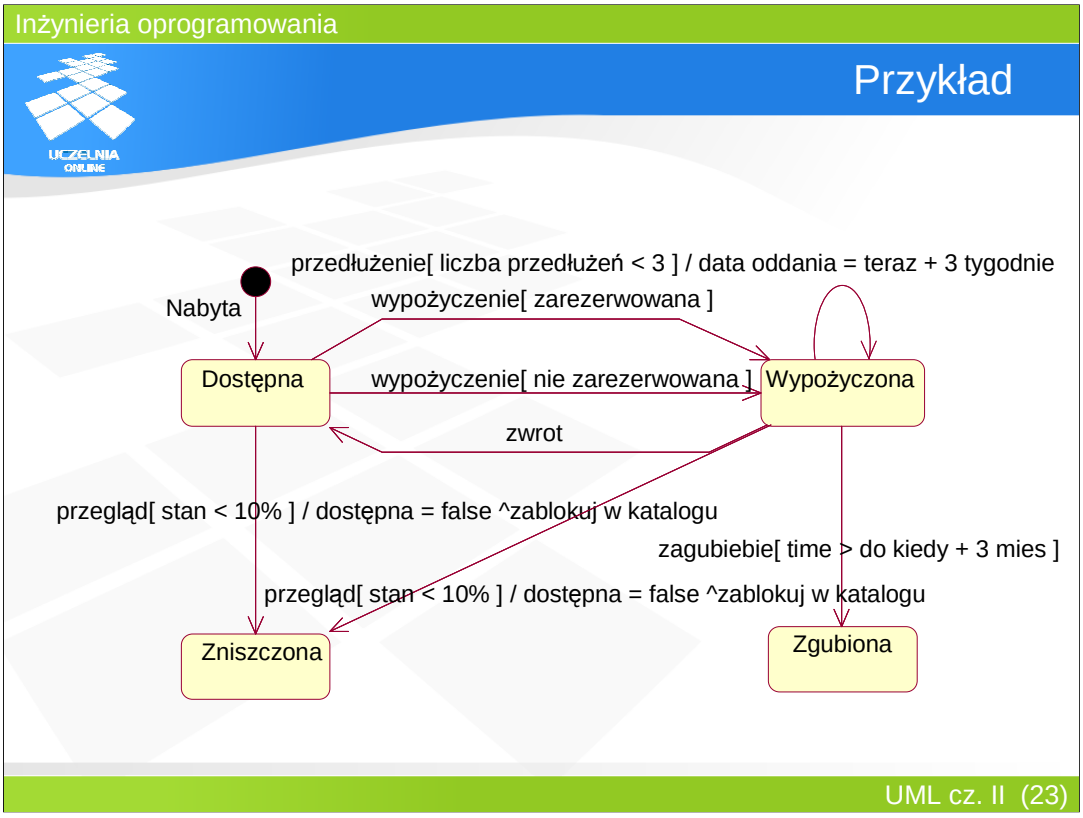
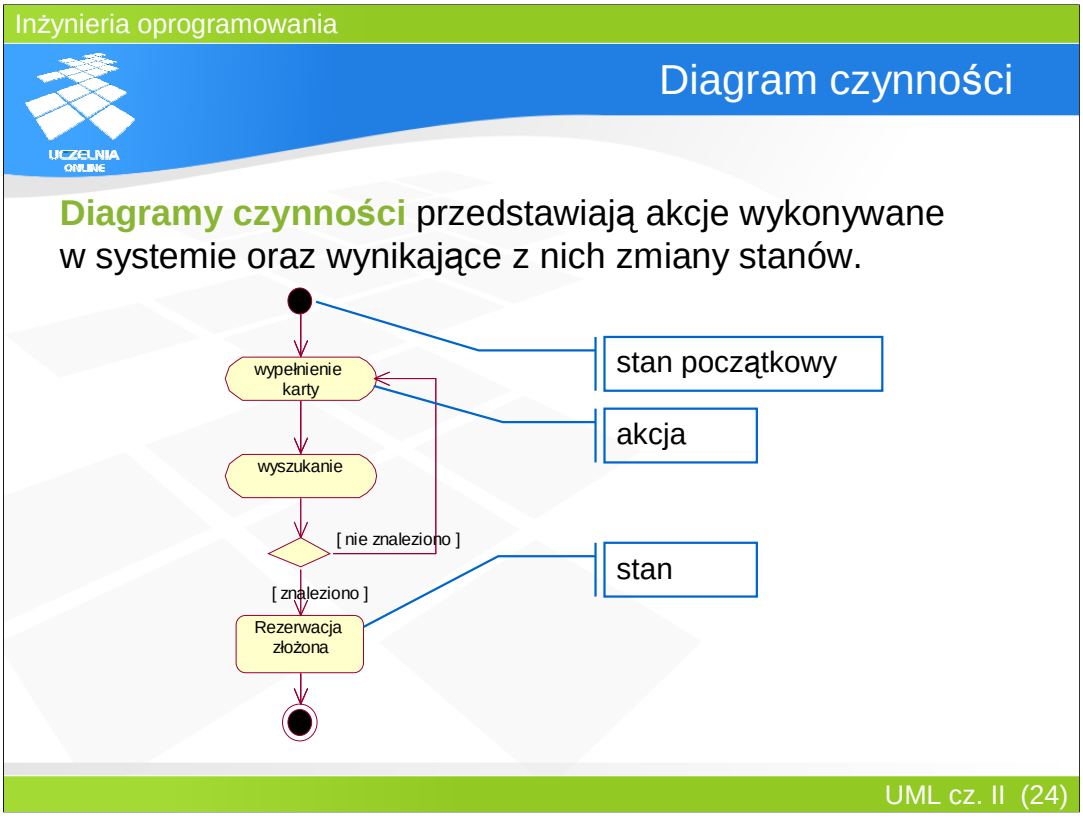


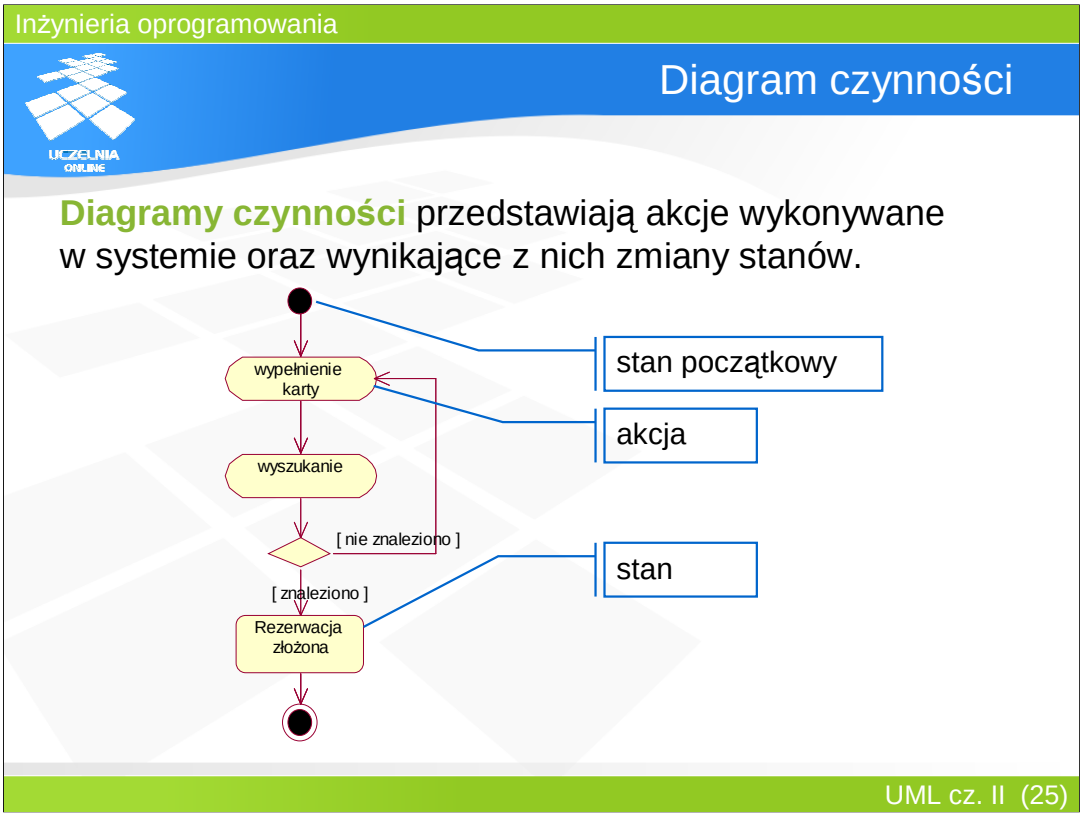
Diagram ten obejmuje przykładowy cykl życia obiektu Książka w bibliotece. Książka zostaje Nabyta, a następnie (po rejestracji) staje się Dostępna. Jej Wypożyczenie (niezależnie od tego, czy była rezerwowana, czy nie), przeprowadza ją do stanu Wypożyczonej. Książka może pozostać w tym stanie np. wskutek przedłużenia (warunkiem jest, że liczba przedłużeń nie przekroczyła 3) – wówczas data oddania jest przesuwana o 3 tygodnie. Zarówno ze stanu Dostępności, jak i Wypożyczenia książka może przejść do stanu Zniszczenia, o ile wskutek zdarzenia Przegląd jej stan zostanie oceniony na mniej niż 10%. Jeżeli tak się stanie, atrybut Książki dostępna otrzymuje wartość false oraz wysyłane jest zdarzenie zablokowania dostępności Książki w katalogu. Wypożyczona książka, której nie oddano w terminie 3 miesięcy od terminu zwrotu, jest uważana za Zagubioną.



Diagramy czynności (ang. *activity diagrams*) prezentują przepływ sterowania w systemie związany z wykonaniem pewnej funkcji. Przepływy łączą czynności wykonywane przez poszczególne obiekty i stany obiektów, w których znajdują się po wykonaniu czynności.

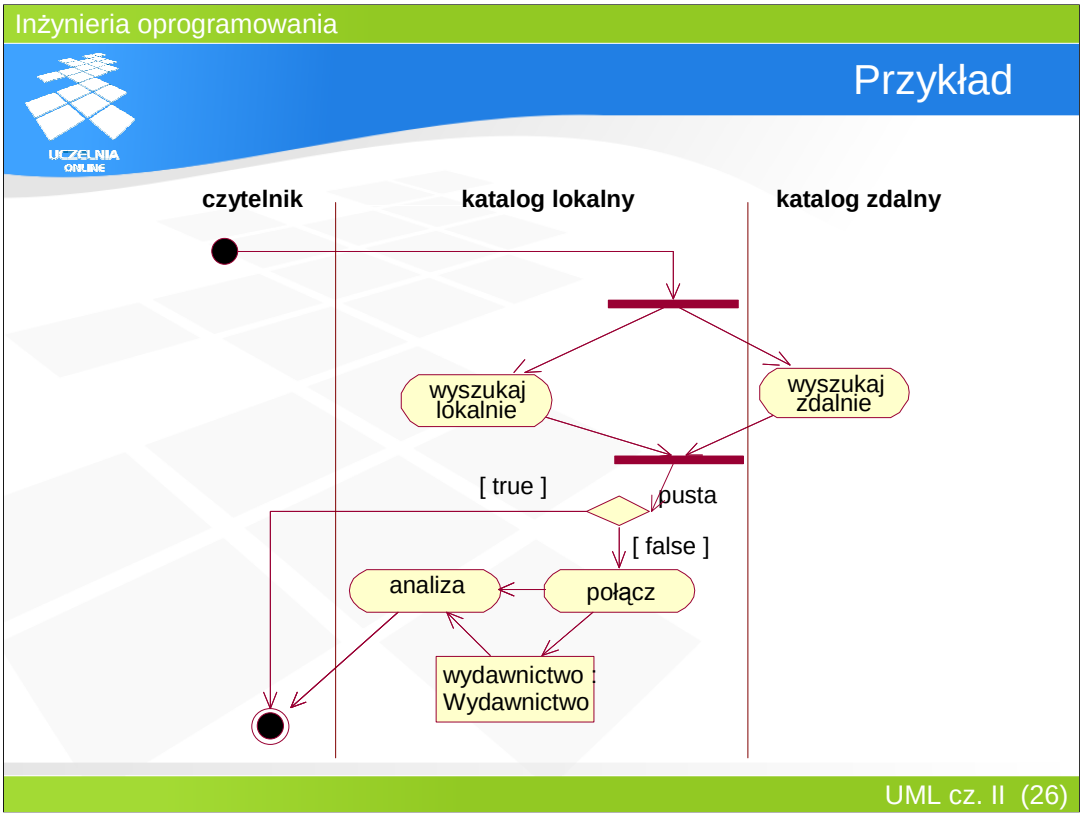
Wygląd tych diagramów przypomina diagramy stanu (w UML 1.x były to diagramy pokrewne, blisko związane ze sobą), jednak ich przeznaczenie jest inne. Diagramy stanu skupiają się – jak nazwa wskazuje – na stanach, a akcje związane z ich zmianą są elementem dodatkowym. W diagramach czynności jest odwrotnie: akcje są na pierwszym planie, a zmiany stanów są efektem ich wykonania. Dlatego diagramy czynności dobrze nadają się do opisu przepływu sterowania pomiędzy obiektami (szczególnie w przypadku przetwarzania współbieżnego) oraz przepływu danych pomiędzy nimi.

Diagram, podobnie jak diagram stanu, może posiadać punkt startowy i i dowolną liczbę stanów końcowych. Najważniejszym jego elementem są akcje, reprezentowane przez prostokąty z zaokrąglonymi narożnikami oraz przejścia (łuki) przedstawiające przepływ sterowania. Łuki mogą być opatrzone warunkami dozoru, które decydują o wykonaniu przejścia oraz zdarzeniami, które są generowane w momencie gdy przejście jest wykonywane. Diagramy czynności zawierają także stany, w jakich może znaleźć się określony obiekt po wykonaniu akcji oraz elementy decyzyjne czy synchronizujące.



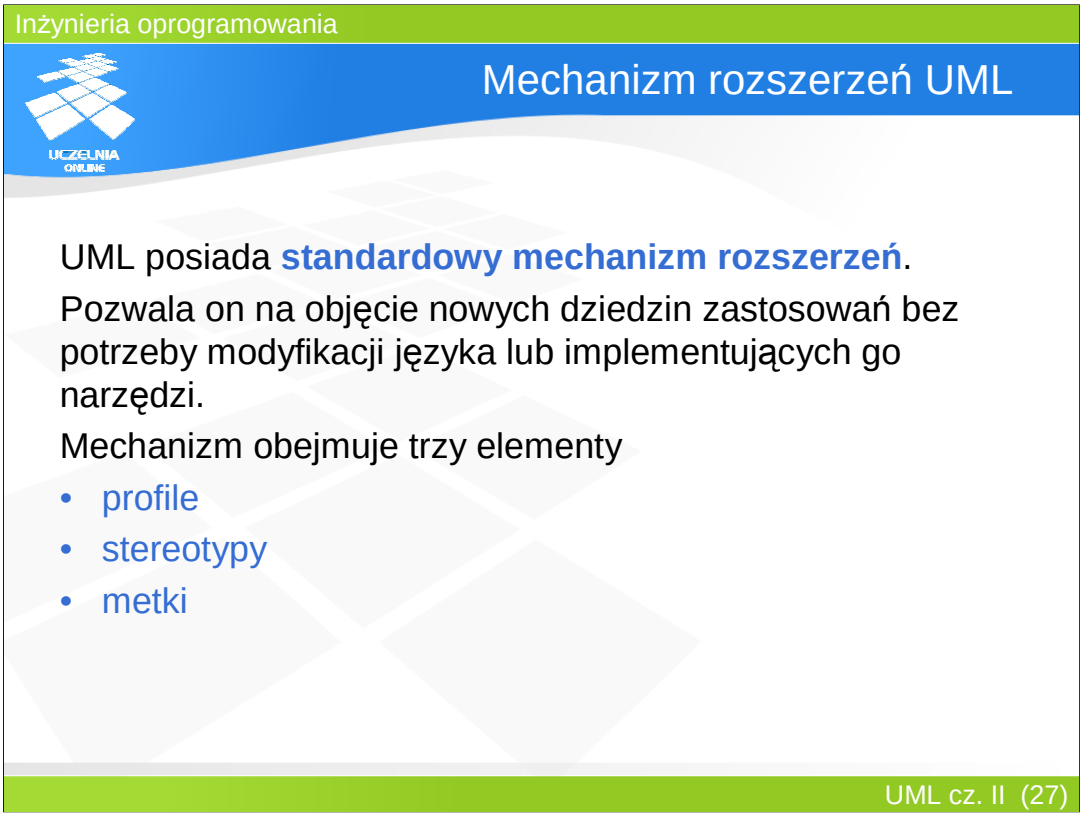
Umieszczanie akcji w torach (ang. *swimlanes*) pozwala na pogrupowanie ich według pewnego kryterium. Może nim być np. obiekt, który wykonuje daną akcję lub inna wspólna cecha akcji.

Obiekty umieszczone na diagramach czynności są podporządkowane przepływowi sterowania i reprezentują parametry wejściowe lub wyniki działania czynności.



Na powyższym slajdzie przedstawiono przykład czynności związanych z wyszukaniem informacji w katalogu przez czytelnika.

Czytelnik jednocześnie uruchamia proces wykonywania lokalnego (w katalogu lokalnym) i zdalnego (w katalogu zdalnym), które są realizowane współbieżnie. Po zakończeniu obu operacji następuje sprawdzenie, czy lista wyników jest pusta. Jeżeli tak, wówczas proces wyszukiwania jest zakończony, w przeciwnym przypadku lista jest łączona z dodatkowymi informacjami prezentowanymi czytelnikowi, analizowana i zwracana inicjatorowi przeszukiwania



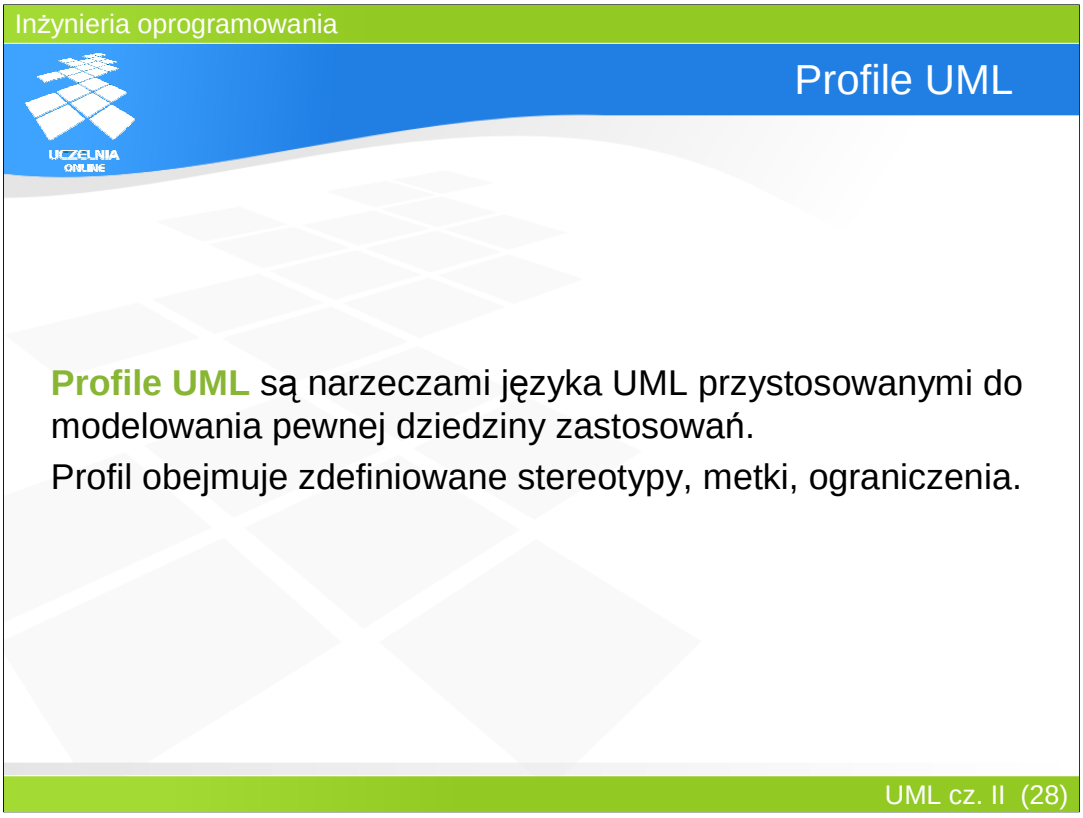
UML służy obecnie do opisu modeli w wielu dziedzinach zastosowań. Aby umożliwić obejmowanie kolejnych dziedzin bez potrzeby modyfikowania lub rozszerzania języka, wprowadzono do niego mechanizm rozszerzeń. Pozwala on redefiniować pojęcia i elementy oraz doprecyzowywać ich znaczenie tak, aby odpowiadały potrzebom nowego obszaru zastosowań.

W skład mechanizmu rozszerzeń wchodzą trzy elementy:

•profile, zawierające rozszerzenia (stereotypy i metki) do modelowania określonych dziedzin

•stereotypy, zmieniające znaczenie poszczególnych elementów

•metki, opisujące dodatkowe właściwości elementów



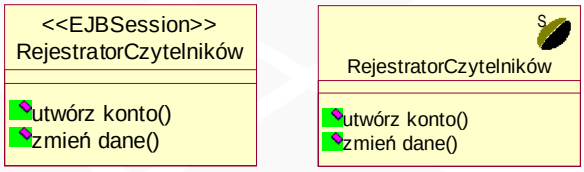
Wśród tych mechanizmów najważniejszym są profile, zawierające kompletny i spójny zestaw elementów dedykowanych do modelowania określonej dziedziny, np. systemów czasu rzeczywistego, baz danych, logiki biznesowej etc. Zdefiniowane i zaakceptowane profile pozwalają uniknąć kaskady różnorodnych rozszerzeń dokonywanych przez użytkowników na własną rękę, co znacznie zmniejszało czytelność i komunikatywność modeli.

Profile zawierają zdefiniowany zestaw stereotypów i metek, z których powinni korzystać analitycy działający w dziedzinie zastosowań profilu. Na przykład, profil do modelowania ziaren EJB definiuje stereotypy EJBSessionBean, EJBPrimaryKey, EJBHomeInterface czy EJBCreateMethod, oznaczające odpowiednio klasę ziarna sesyjnego, atrybut będący kluczem podstawowym ziarna encyjnego, interfejs domowy ziarna oraz metodę tworzącą instancję interfejsu ziarna. Stereotyp EJBSessionBean definiuje m.in. metkę EJBTransType, natomiast każda operacja może posiadać metkę EJBRoleNames, określającą role, które musi odgrywać wywołujący tę operację element.

Stereotypy

Stereotypy służą do zmieniania lub doprecyzowania semantyki elementów modelu. Dzięki nim można dokładnie określić ich rolę w systemie.

Stereotypy są reprezentowane przez ikonę lub słowo kluczowe ujęte w "łapki": **«stereotyp»**



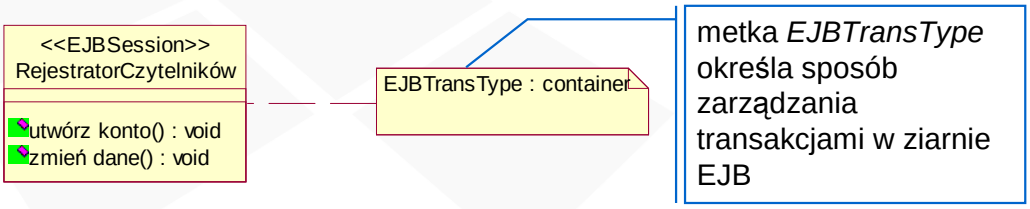
Różne sposoby przedstawienia stereotypu «*EJBSession*» w klasie *RejestratorCzytelników*

Kolejnym elementem jest stereotyp, który historycznie był podstawowym narzędziem rozszerzania i modyfikowania UMLa. Stereotypy dodają do znanych już elementów: klas, atrybutów, asocjacji, nową semantykę. W UML 1.x stereotypami były wszelkie dekoracje zmieniające znaczenie wybranego elementu. Wersja ta zawierała także sporą grupę zdefiniowanych stereotypów standardowych. W UML 2.0 nazwę tę zarezerwowano wyłącznie dla dekoracji wchodzących w skład profili, natomiast pozostałe użycia tego słowa zostały przemianowane na słowa kluczowe. Stereotypy posiadają specjalną notację, polegającą na umieszczeniu nazwy stereotypu w specjalnych znakach *guillemets* («stereotyp»). Stereotypy, szczególnie te najczęściej używane, posiadają także własną ikonę, która jest umieszczana wewnątrz stereotypowanego elementu lub całkowicie go zastępuje.

Metki

Metki (ang. *tagged* values) pozwalają dołączyć do elementu dodatkowe właściwości

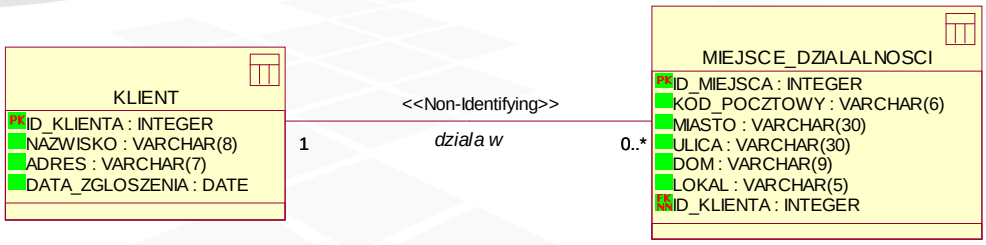
- Metki posiadają postać par *klucz : wartość*
- Stereotypy i profile definiują własne zestawy metek
- Metki są zapisywane w postaci notatki dołączonej do elementu, którego dotyczą



Wreszcie, na najniższym poziomie znajdują się tzw. metki (ang. *tagged values*), pozwalające opisywać dodatkowe właściwości elementu, które nie zostały przewidziane w UMLu. Metki są zapisywane w postaci par klucz-wartość w nawiasach klamrowych i dołączane do opisywanych elementów w postaci notatek. W większości narzędzi są one jednak zapisywane w postaci metadanych, zawartych wewnątrz elementu, ponieważ tak łatwiej je przetwarzać. Metki, podobnie jak stereotypy, są zasadniczo definiowane wewnątrz profili (i znajdujących się w nich stereotypów), jednak istnieje także możliwość ich definiowania przez użytkowników.

Najprostszym przykładem metki może być informacja o autorze modelu {autor = "Jan Kowalski"}.

Przykład: modelowanie baz danych

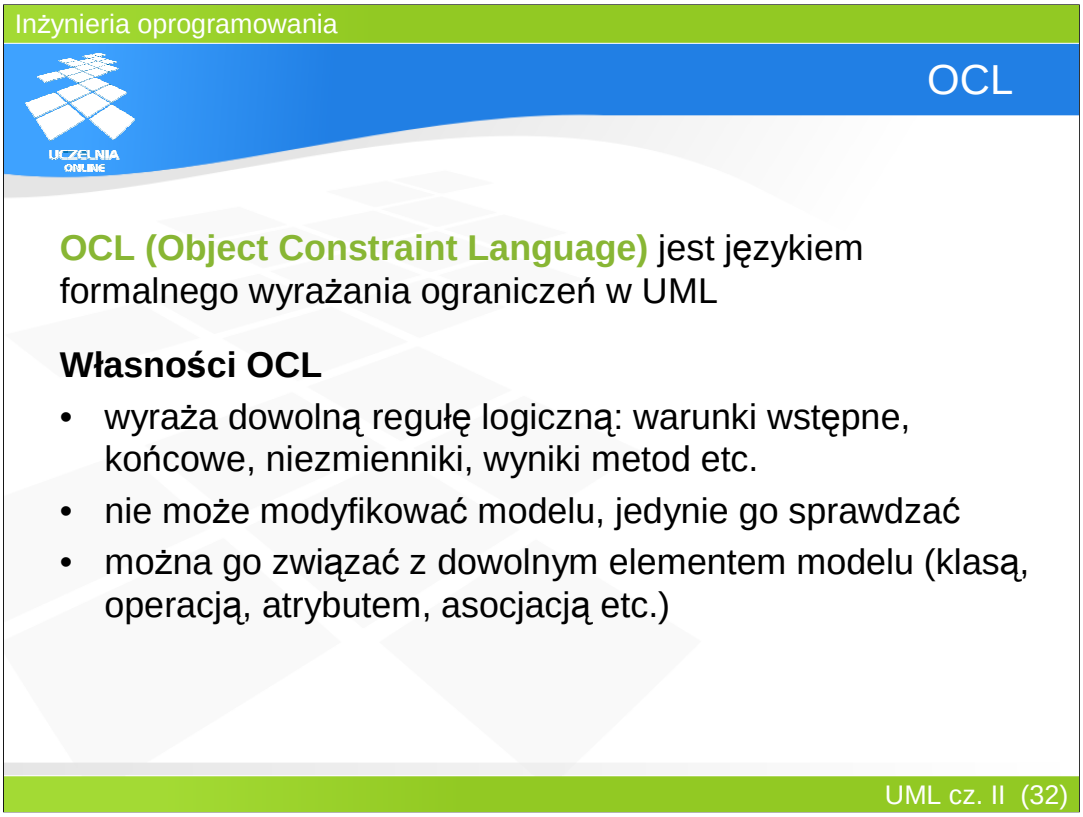


schemat	pakiet **«schema»**
tabela	klasa **«RelationalTable»**
klucz podstawowy	atrybut **«PK»**
klucz obcy	atrybut **«FK»**
ograniczenie	metoda **«constraint»**

Na powyższym slajdzie przedstawiono prosty przykład wykorzystania jednego z profili UML, służącego do modelowania danych. Wprawdzie relacyjne bazy danych posiadają własną notację, opartą na diagramach ERD (*Entity Relationship Diagrams*), jednak możliwość ich tworzenia w UMLu jest ważnym uzupełnieniem jego możliwości.

Profil ten definiuje stereotypy, które można umieszczać na istniejących elementach UML w celu nadania im nowego znaczenia w dziedzinie projektowania baz danych. Na przykład, schemat bazy danych jest reprezentowany przez pakiet ze stereotypem schema, tabela jest modelowana jako klasa ze stereotypem RelationalTable, a jej klucze podstawowe i obce – jako atrybuty odpowiednio ze stereotypami PK i FK. Ograniczenia integralnościowe wewnątrz relacji są metodami ze stereotypem constraint.

Tak opisany schemat danych może być użyty do wygenerowania kodu w języku definicji baz danych (np. SQL DDL), który następnie posłuży do utworzenia schematów i tabel zgodnych z nim.



OCL jest formalnym językiem wyrażania wszelkiego rodzaju ograniczeń obecnych w UMLu. Choć użycie jego nie jest obowiązkowe (ograniczenia można równie dobrze specyfikować w języku naturalnym), jednak jego rola w dobie narzędzi generujących kod z diagramów, będzie stale rosła.

Warto pamiętać, że OCL jest językiem potrafiącym jedynie weryfikować elementy modelu, ale nie mogącym na ten model w żaden sposób wpływać. Ewaluacja wyrażeń OCL następuje w sposób atomiczny (niepodzielny), nie powodując nigdy zmiany stanu jakiegokolwiek obiektu.

OCL posiada zestaw wbudowanych operatorów, predykatów, ma możliwość definiowania własnych funkcji, warunków i niezmienników. Dzięki nim możliwe jest użycie go przy niemal wszystkich elementach występujących w UML

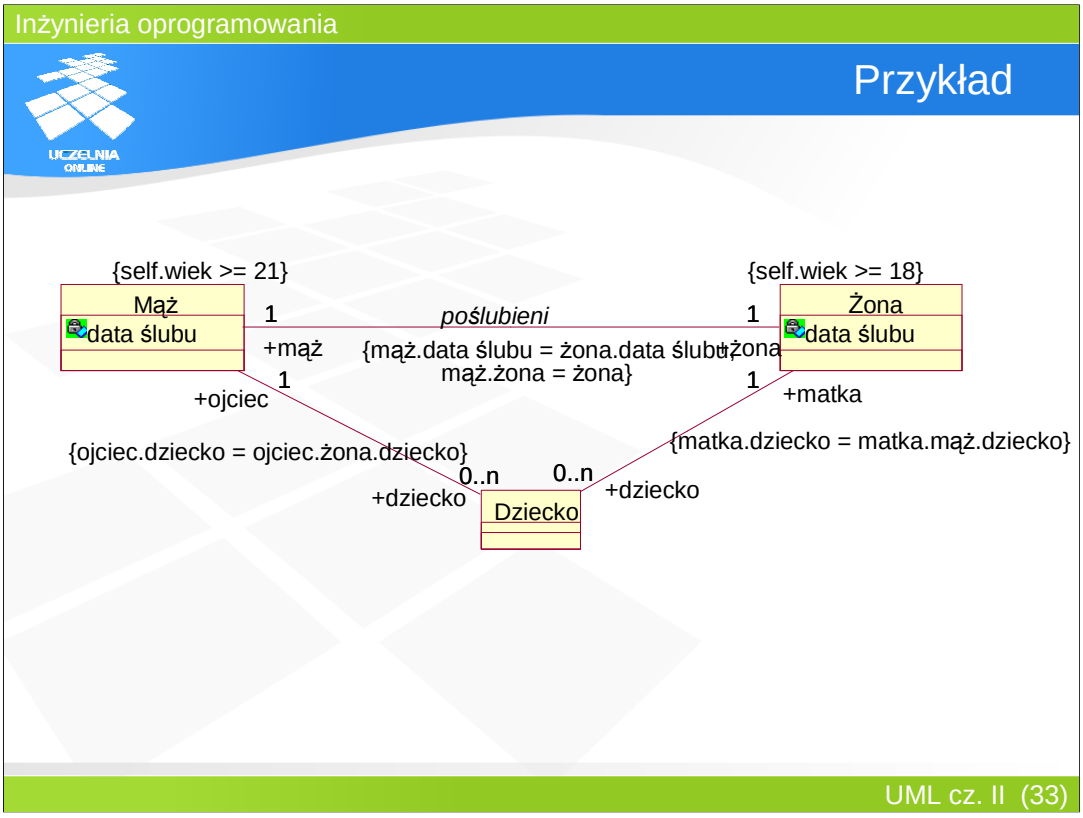


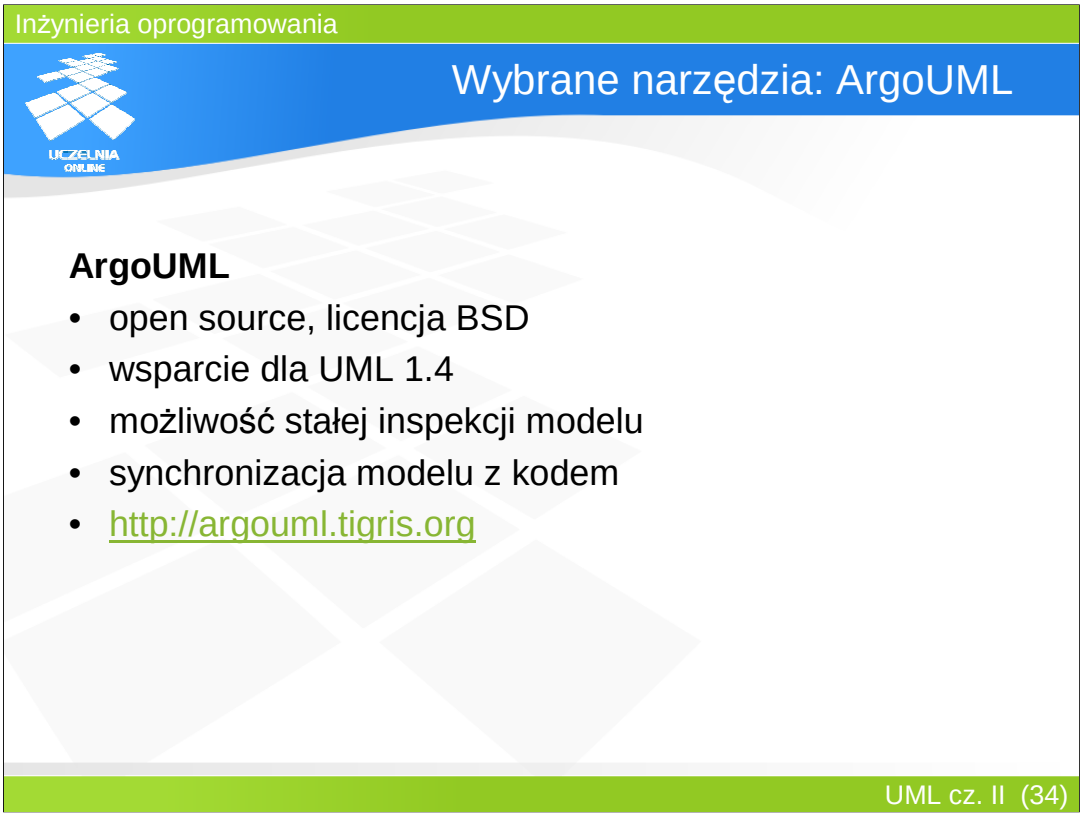
Diagram przedstawia rodzinę. Obiekt klasy Mąż jest związany z dokładnie jednym obiektem klasy Żona. Każde z nich jest związane z obiektami klasy Dziecko.

Sam rysunek bez ograniczeń mógłby prowadzić do rozmaitych interpretacji, także nieprawdziwych. Dlatego wprowadzenie ograniczeń w OCL pozwala uściślić model.

Relacja pomiędzy Mężem i Żoną ma nałożone ograniczenie, że data ślubu obu obiektów musi być identyczna, a także nawigując od Męża poprzez związany z nim relacją poślubieni obiekt Żona, otrzymujemy uczestniczący w tej relacji obiekt Żona (zatem Mąż i Żona są ze sobą związani relacją wzajemności).

Ponadto Żona musi mieć wiek powyżej 18 lat, a Mąż – 21.

Aby zapewnić, że dzieci posiadane przez Żonę były także dziećmi Męża, nałożono odpowiednie ograniczenia na relacje między Mężem i Dzieckiem oraz Żoną i Dzieckiem.



Przykładem darmowego i otwartego narzędzia do modelowania w UML jest ArgoUML. Jest to program zaimplementowany w języku Java, który można uruchomić na dowolnej platformie programowej wyposażonej w interpreter tego języka.

Posiada on wsparcie dla wersji 1.4 UML, natomiast nie ma zaimplementowanej obsługi żadnego z nowych diagramów, jakie pojawiły się w wersji 2.0 języka. Posiada także moduł inspekcji modelu, znajdujący najpopularniejsze błędy popełniane przez analityków, zaimplementowane w postaci reguł. Umożliwia także synchronizację kodu z modelem dla wybranych języków programowania.

Wybrane narzędzia: Rational

Rational Rose, Rational Software Modeler

- narzędzia komercyjne
- integracja z innymi narzędziami Rational
- wsparcie dla UML 1.x (Rose) i UML 2.0 (Modeler)
- wsparcie dla modelowania wybranych dziedzin i technologii
- modelowanie biznesowe
- synchronizacja modelu z kodem
- http://www-306.ibm.com/software/rational/

Na drugim biegunie znajdują się narzędzia firmy Rational (obecnie Rational Division wewnątrz IBMa). Klasycznym narzędziem do modelowania jest Rational Rose, natomiast nowszą linię produktową reprezentuje Rational Software Modeler. Ten ostatni produkt jest oparty na środowisku Eclipse i posiada wsparcie dla wersji 2.0 UML.

Wśród możliwości obu środowisk warto wymienić możliwość wykorzystania profili języka UML w postaci wtyczek dedykowanych do rozmaitych technologii (modelowania danych, modelowania biznesowego etc.). Narzędzia te łatwo integrują się z innymi produktami Rational i IBM, posiadają jednak także możliwość współpracy z wybranymi innymi narzędziami.

Podsumowanie

- Diagramy komponentów i wdrożenia przedstawiają logiczną i fizyczną strukturę podsystemów
- Diagramy interakcji służą do opisu komunikacji pomiędzy obiektami
- Diagramy czynności definiują algorytmy realizacji funkcji, a diagramy stanu – zmianę zachowania obiektów
- Mechanizm rozszerzeń UML pozwala na obejmowanie nowych obszarów zastosowań
- OCL jest językiem formalnego opisu ograniczeń

Wykład zakończył krótkie wprowadzenie do modelowania z wykorzystaniem języka UML. Przedstawiono najważniejsze rodzaje diagramów i ich możliwości wyrazu, a także rozszerzone możliwości języka.