3. Problemy transformacji i spójność modeli

## Modelowanie

* Proces budowy modeli

## Model

* Abstrakcyjna (uproszczona) wizja pewnego rzeczywistego lub wyobrażanego bytu
* Zależy od przyjętej perspektywy modelowania (wybrane istotne własności, zależy od celu modelowania)
* Abstrakcja systemu lub jego części
* Role modelu:
  + Reprezentuje to co istnieje (dziedzinowe, biznesowe)
  + Reprezentuje to co ma powstać (projektowy)

## Cechy dobrego modelu

* Poprawność semantyczna modelu - model odpowiada dziedzinie problemu którą opisuje
* Czytelność - model musi operować pojęciami związanymi z dziedziną zastosowania
* Kompletność – model musi być kompletny względem modelu odniesienia
* Adekwatność – model musi być adekwatny względem celu modelowania
* Elastyczność – model musi być podatny na zmiany
* Spójność – zgodność, brak sprzeczności pomiędzy elementami modelu.

## Kluczowe aspekty związane z modelowaniem

* Reprezentacja modeli (np. wykorzystanie języków dziedzinowych)
* Transformacja modeli
* Walidacja i weryfikacja modeli

## Transformacja modeli

Źródło: (<http://www.pst.informatik.uni-muenchen.de/personen/kochn/iet0307-koch.pdf>)

1. Transformacja modeli to proces konwersji jednego bądź wielu modeli wejściowych (źródłowych) na **dokładnie jeden** model wyjściowy (docelowy) tego samego systemu.
2. Do definiowania transformacji modeli wykorzystuje się języki transformacji, które na poziomie meta-modelu specyfikują jak określone typy elementów meta-modelu modelu źródłowego są konwertowane na inny typ elementów meta-modelu modelu docelowego. :-) (chodzi generalnie o to że zarówno model źródłowy jak i docelowy są oparte na jakiś meta-modelach i te języki definiują nam jak elementy meta-modelu modelu źródłowego zamienia się na elementy meta-modelu modelu docelowego)

Aplikuje je się na poziomie modelu, tak aby zamienić elementy modelu źródłowego na elementy modelu docelowego.

* Wśród transformacji modeli możemy wyróżnić transformacje **jednokierunkowe** (tzw. mapowania) lub **dwukierunkowe** (tzw. relacje)

## Aspekty związane z transformacją modeli

* **Poziom automatyzacji**

Ze względu na ilość „manualnej interwencji” można wyróżnić:

* + **Transformacje automatyczne** – transformacja jest automatyczna, jeżeli nie wymaga żadnych decyzji od osoby dokonującej transformacji (projektanta).
  + **Transformacje pół-automatyczne** – Transformacja jest pół-automatyczna jeżeli osoba jej dokonująca musi podjąć decyzje, które elementy źródłowego modelu mają ulec transformacji
  + **Transformacje ręczne** (tradycyjne) – Transformacja jest manualna (ręczna) jeżeli jest w pełni dokonywana przez projektanta (on podejmuje wszystkie decyzje)

Celem procesu wytwarzania z wykorzystaniem modeli jest zdefiniowanie reguł transformacji, które pozwolą naautomatyczną transformacje modeli.

* **Transformacje w MDA** *(patrz dalej)*
* **Ilość modeli źródłowych**

Przy wykonywaniu transformacji do budowy modelu docelowego może być wykorzystanych wiele modeli źródłowych. Ze względu na ilość wykorzystywanych modeli źródłowych można wyróżnić transformacje **proste** (Simple) lub **scalające** (Merge).

* **Różnorodność języków modelowania**

Modele wyraża się przy pomocy różnych języków modelowania takich jak UML dla modeli projektowych (design models) czy język programowania dla modeli implementacyjnych (Source-code models). Elementy modelu oraz częściowo ich semantyka są reprezentowane przez powiązany z nim meta-model. Można wyróżnić:

* **Transformacje wewnętrzne (**endogeneus) – pomiędzy modelami wyrażonymi w tym samym języku modelowania np. z UML do UML.
* **Transformacje zewnętrzne** (exogeneus) – pomiędzy modelami wyrażonymi w różnych językach np. z UML do kod źródłowego. Jeżeli transformacja jest **scaleniem** (merge) wielu modeli źródłowych, to mogą być przy tym zaangażowane więcej niż dwa różne języki.
* **Wykorzystanie znaczników i dodatkowych informacji**

Aby wybrać elementy modelu źródłowego, do których zostaną zastosowane odpowiednie reguły transformacji, można wykorzystać pewne znaczniki. Mogą one być wbudowane w model źródłowy np.

* Typy – np. klasy lub asocjacje,
* Szablony lub stereotypy Profilu UML. Profil UML – zbiór stereotypów, dodatkowych elementów modelowania, które uściślają semantykę istniejących elementów modelowania nie naruszając strukturalnego opisu języka (meta-modelu) np. stereotypy Boundary, Control, Entity
* Wzorce – jako kombinacje elmentów modelu,

Lub oddzielne „modele znakujące”, które łączy się z modelem źródłowym podczas transformacji – np. szablony (podobne do wzorców ale mogą zawierać bardziej szczegółowe informacje niezbędne do transformacji).

Dodatkowe informacje – bazujące na wiedzy projektanta. Np. informacja o użytym stylu architektonicznym

* **Techniki implementacji (transformacje automatyczne)**

Transformacje mogą zostać zaimplementowane w:

1. Językach ogólnego przeznaczenia – np. Java, C#
2. Językach do transformacji grafów – np. AGG, VIATRA (modele traktowane jako grafy)
3. Specjalnych językach do transformacji – np. QVT (*parz niżej*)

Transformacje często też bazują na inwariantach, warunkach wstępnych i końcowych (pre-conditions i post-conditions) zdefiniowanych w OCL. Modele serializowane za pomocą XML mogą skolei być transformowane z wykorzystaniem XSLT

## Problemy związane z transformacją modeli

Wykonywanie transformacji manualnie, bez wsparcia żadnego narzędzia, może stanowić przyczynę wielu problemów związanych z transformacją modeli, np.:

* **Niespójność** między modelami źródłowym a docelowym. Niektóre elementy modelu źródłowego i docelowego są sprzeczne. (*patrz niżej*)
* **Niekompletność** modelu docelowego. Np. pominięcie niektórych elementów (klas), powiązań ( asocjacji, generalizacji) , ograniczeń, nieuwzględnienie liczności.
* **Nieczytelność, niepoprawność** modelu docelowego. Operuje on pojęciami niezwiązanymi z dziedziną (innymi pojęciami niż model źródłowy). Np. osoba dokonująca transformacji pomyliła/przekręciła/pominęła pojęcia
* **Błędnie dokonana transformacja.** Osoba dokonująca transformacji nie zrozumiała/źle zinterpretowała model wejściowy lub nie zastosowała odpowiednich reguł transformacji. Może prowadzić do wszystkich wymienionych wyżej problemów.

Przed powyższymi problemami może uchronić podejście MDA i zastosowanie automatycznej transformacji modeli.

## Spójność modeli

* Można wyróżnić:
  + **Spójność wewnątrz - modelową** – model nie zawiera elementów sprzecznych, wewnętrzną poprawność
  + **Spójność między - modelową** – dwa modele (zarówno na tym samym, jak i na różnych poziomach abstrakcji) tworzone w tym samym procesie wytwarzania oprogramowania nie mogą się zaprzeczać.
* Przy budowie i transformacji modeli istotne jest zapewnienie spójności modeli. Wiąże się to z nakładaniem ograniczeń na tworzone modele. Ogólne podejście w modelowaniu obiektowym, przy wykorzystaniu języka UML, zaleca stosowanie OCL jako języka do formułowania ograniczeń.
* W modelowaniu z wykorzystaniem języka UML, różne aspekty systemu są objęte różnymi diagramami co prowadzi do ryzyka niespójności całej specyfikacji systemu. Dlatego ważne jest zapewnienie sposobów na sprawdzenie spójności i kompletności modelu.

## OCL - Object Constraint Language

* Język, który pozwala uzupełnić model UML o dodatkowe informacje. Jest rozszerzeniem UML które umożliwia:
  + Definiowanie ograniczeń przy budowie modeli (niezmienni klas, warunki wstępne i końcowe operacji) - wymagania semantyczne, warunki nałożone na elementy modelu
  + Definiowanie składni kontekstowej (semantyki statycznej) w definicji języka UML.
* OCL nie jest językiem opisującym zachowania/akcje. **Wyrażenia w OCL nie mają ’efektów ubocznych’.**
  + OCL nie zmienia wartości elementów modelu
  + OCL nie deﬁniuje operacji innych niż zapytania
  + OCL może wykonywać tylko takie zapytania, które nie zmieniają wartości
  + OCL nie może dynamicznie zmieniać reguł działania modelu (at runtime) - reguły ustala się statycznie na etapie modelowania
* **Język specyfikacji a nie programowania** – określa to czego się oczekuje. Użytkownik opisuje efekt, jaki chce uzyskać, a nie sposób wykonania.
* OCL nie jest wykonywalny. To tylko język do tworzenia zapytań i nakładania ograniczeń.
* OCL to zmodyfikowana wersja klasycznego języka predykatów
* Wyrażenia OCL można zapisywać w osobnych plikach lub jako notatki dołączone do modelu UML. Sposób realizacji tego zależy od użytego narzędzia. Specyﬁkacja przewiduje bazujący na XML format zapisu umożliwiający wymianę wyrażeń OCL pomiędzy różnymi narzędziami.
* Zalety:
  + Pozwala sprawdzać spójność modelu.
  + Umożliwia odpowiednim narzędziom generowanie kodu bazującego na wyrażeniach OCL
  + Doprecyzowuje model, chroni przed błędną interpretacją
  + Semantyka OCL jest niezależna od użycia konkretnej składni.
  + Pozwala na tworzenie nowych alternatywnych reguł (np. ’SQL-like’).
* Wady:
  + Jest to mały język, ale ma stosunkowo trudną składnię, która ciągle ewoluuje. Zawiera mylące wyjątki i skróty
  + OCL ciężko się czyta. Składnia jest nieregularna.
  + Niewielu ’modelarzy’ zna OCL, i jeszcze mniej programistów.
  + Poziom precyzji oferowany przez OCL nie zawsze jest konieczny. Stosowanie OCL może być przesadą (np. w modelach nieformalnych).

## Transformacje w MDA

* Transformacja jest kluczową techniką używaną w MDA.
* Transformacja w MDA to automatyczna generacja modelu wyjściowego z modelu wejściowego wykonana w oparciu o tzw. **definicję transformacji**
* **Definicja transformacji** to zbiór tzw. **reguł transformacji**
* **Reguła transformacji** to opis przekształcenia jednego lub więcej elementów języka modelu wejściowego w jeden lub więcej elementów modelu wyjściowego

## MDA (Model Driven Architecture = Architektura sterowana modelami)

* Zaproponowana przez OMG (Object Management Group) jako podejście do projektowania i rozwoju systemów informatycznych z wykorzystaniem modelowania
* Projektowanie z wykorzystaniem modeli:
  + ułatwia zrozumienie modelowanej dziedziny przedmiotowej przez projektantów
  + umożliwia weryfikacji modeli już w początkowej fazie procesu projektowania – akceptacja modelu przez zespół/klienta/ekspertów dziedzinowych
* Stosowanie modeli na każdym etapie wytwarzania oprogramowania (różne poziomy abstrakcji)
* **Przejście pomiędzy poszczególnymi poziomami abstrakcji realizowane zgodnie z określonym zbiorem reguł transformacji**
* Kod systemu generowany automatycznie z modelu 🡪 język modelowania nowym językiem programowania (wzrost abstrakcji tworzenia systemów względem tradycyjnych języków programowania )

## Typy modeli wg. MDA

Poziom abstrakcji

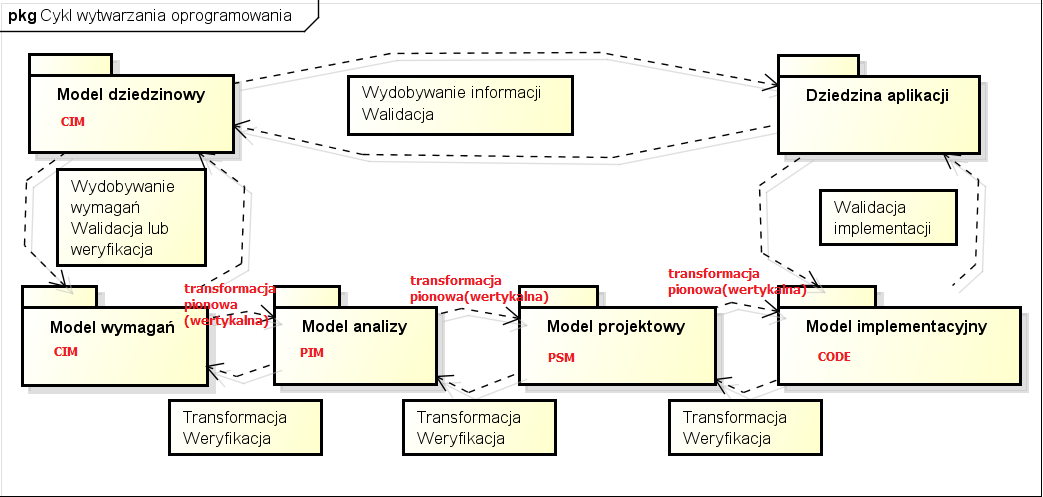
1. Model CIM (Computation Independent Model) :
   * Model niezależny obliczeniowo, nie precyzujący zakresu odpowiedzialności oprogramowania,
   * Odpowiednik modeli biznesowego i PU, reprezentuje dziedzinę przedmiotową (model dziedzinowy + model wymagań).
   * Musi być **wewnętrznie spójny i kompletny**.
2. Model PIM (Platform Independent Model) :
   * Model niezależny od platformy (opisuje działanie system niezależnie od platformy)
   * Odpowiednik modelu analizy, zawiera szczegóły zachowań opisane niezależnie od systemu, na którym ma działać
   * Medium komunikacji zespołu
3. Model PSM (Platform Specific Model):

* Model specyficzny dla platform – te same dane co PIM + szczegóły dotyczące platformy
* Odpowiednik modelu projektowego
* Na jego podstawie automatyczne generowanie kodu

1. Implementation Model = Code – kod źródłowy generowany automatycznie z modelu PSM

## Klasyfikacja transformacji modeli (MDA)

1. Ze względu na to czy zmieniają poziom abstrakcji. Poziom abstrakcji modelu to miara ilości szczegółów jakich zawiera model.
   1. **Poziome = Horyzontalne**:
      * Modele wejściowy i wyjściowy znajdują się **na tym samym** poziomie abstrakcji
      * Może być związana np. z uściśleniem modelu pozostając na tym samym poziomie abstrakcji, poprawą jakości modelu
      * Transformacje horyzontalne zwykle zachodzą z dwóch powodów:
        + Poprawa jakości atrybutów modelu (refaktoryzacja)
        + Wsparcie analizy modelu
      * Migracja (zmiana języka bez zmiany poziomu abstrakcji) i restrukturyzacja (refaktoryzacja) - poprawa
   2. **Pionowe = Wertykalne**:
      * Modele wejściowy i wyjściowy znajdują się **na różnych** poziomach abstrakcji
      * Można wyróżnić transformacje wertykalne:
        + Doskonalące, uściślające (dodają szczegóły do modelu) – ang. refinement transformations
        + Czyniące model bardziej abstrakcyjnym (usuwają szczegóły z modelu) – abstraction transformations
      * Inżynieria wstecz i do przodu
        + **Inżynieria do przodu** to proces transformacji modelu na kod za pomocą przekształcenia do danego języka implementacji.
        + **Inżynieria wstecz** to proces transformacji kodu na model za pomocą przekształcenia z danego języka implementacji.

****

1. Ze względu na obiekt docelowy transformacji:
2. Transformacje **model-to-model** – element docelowy transformacji: model (CIM do CIM; CIM do PIM, PIM do PIM, PIM do PSM)

Metody transformacji pomiędzy modelami:

* 1. **Metody relacyjne/deklaratywne** – zamiast skupiać się jak przeprowadzić transformacje, skupiamy się na tym co powinno ulec transformacji. Deklaratywne modele transformacji opisują relacje pomiędzy źródłowymi i docelowymi meta modelami (mogą być interpretowane dwukierunkowo)
  2. **Metody imperatywne/operacyjne** – transformacja opisywana jest jako sekwencja akcji. Użycie języka imperatywnego do opisu jak transformacja powinna być przeprowadzona
  3. **Metody hybrydowe** – hybryda dwóch pierwszych. Języki hybrydowe oferują konstrukcje zarówno imperatywne, jak i deklaratywne.
  4. **Manipulacja bezpośrednia** - użycie języka programowania ogólnego przeznaczeniado implementacji modelu transformacji. Minus: duże i trudne w pielęgnacji implementacji. Plus: brak konieczności nauki nowego języka
  5. **Transformacje grafów** – modele traktowane jako grafy, a transformacje operują na podgrafach. Triple Graph Grammars (TGG) – do opisu transformacji grafowych.
  6. **Transformacje wykorzystujące XSLT** (język przekształceń - Zawiera elementy służące do definiowania reguł opisujących sposób przekształcania jednego dokumentu XML na inny dokument; Transformacja XSLT to automatyczna konwersja dokumentu XML do formatu HTML, WML, itp. lub innego dokumentu XML)

1. Transformacje **model-to-text** – elementy modelu źródłowego są mapowane na tekst (np. transformacja do kodu źródłowego- PSM do CODE)

Metody transformacji pomiędzy modelem i kodem:

* 1. Metody wykorzystujące wzorzec Wizytator
  2. Metody oparte o szablony (np. Visual Studio Visualization and Modeling SDK). Szablony zawierają fragmenty tekstu docelowego. Zakłada się, ze meta program użyty do transformacji ma dostęp do modelu źródłowego. Meta Języki oparte na szablonach często są łączone z użyciem wzorca Wizytator.

## Transformacja modeli w procesie wytwarzania oprogramowania

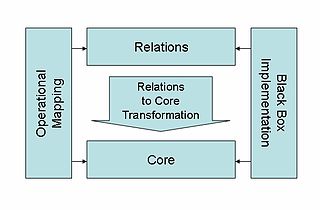
* Transformacja modeli wiąże się pośrednio z językami dziedzinowymi (Domain Specific Language – DSL - języki programowania przeznaczone do specyficznych zadań, specyficzne dla domeny, ograniczona ekspresywność).
* W procesie wytwarzania oprogramowania **modele są kolejno przekształcane z wykorzystaniem transformacji** – model reprezentowany przez jeden język dziedzinowy (na jednym poziomie abstrakcji ) może być przetransformowany na model reprezentowany w innym lub tym samym języku dziedzinowym (znajdujący się na tym samym lub innym poziomie abstrakcji )
  + - **Idealny model wytwarzania** zakładałby tylko transformacje wertykalne (pionowe): z CIM bezpośrednio do PIM, następnie do PSM i w kod 🡪 Taki model jest jednak praktycznie niemożliwy



* + - **W rzeczywistości** między transformacjami wertykalnymi mamy zwykle dodatkowo transformacje poziome (horyzontalne) – transformacje modeli na tym samym poziomie abstrakcji:



## Standard QVT (Query-View-Transformation)

* Jest standardem dla modelu transformacji, a ściślej mówiąc QVT jest językiem pozwalającym zaprojektować automatyczne transformacje między modelami UML. Transformacje raz stworzone mogą zostać np. zapisane w języku QVT i wykorzystane później w innych projektach.
* Standardowy zbiór języków specjalnie przeznaczonych do definiowania transformacji, zdefiniowany przez OMG.
* Definiuje on 3 języki transformacji modelowej. Wszystkie z nich operują na meta modelach zdefiniowanych w MOF 2.0 (MOF = Meta-Object Facility - określa sposób, standardy modelowania oraz zarządzania modelami). Transformacja wykonana w dowolnym z tych 3 języków może być uważana za model zgodny z metamodelami zdefiniowanymi w standardzie MOF. Standard QVT integruje OCL 2.0 i rozszerza go o elementy języka imperatywnego (program jako sekwencja instrukcji zmieniających stan programu).
* **QVT-Operational** – język imperatywny zaprojektowany do pisania transformacji jednokierunkowaych.
* **QVT-Relations** – język deklaratywny (programista opisuje warunki, jakie musi spełniać końcowe rozwiązanie -*co chcemy osiągnąć*- a nie szczegółową sekwencję kroków, które do niego prowadzą -*jak to zrobić*) zaprojektowany aby zezwolić na pisanie zarówno jednokierunkowych, jak i dwukierunkowych transformacji. Transformacja w tym języku uwzględnia spójność zbioru modeli. Spójność ta może być sprawdzana poprzez wywołanie transformacji w trybie (check-only) – zwraca ona true gdy zbiór modeli jest spójny i false w.p.p. Ta sama transformacja może być też użyta do modyfikacji jednego z modeli – tak, że zbiór modeli będzie spójny. Jezyk QVT-Relations – składna tesktowa i graficzna.
* **QVT-Core** – prosty język deklaratywny, wynik transformacji z języka QVT-Relations.
* Mechanizm QVT-BlackBox służy do wywoływania transformacji na modelach w innych językach (np. XSLT, XQuery)
* Języki QVT nie zezwalają na transformację z lub do modeli tekstowych. Transformacje model-to-text są standaryzowane osobno przez OMG (MOFM2T)