КВАЛІФІКАЦІЙНА РОБОТА

Група МІПЗс-22

Рущак О.Я.

2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Рущак Олексій Ярославович

УДК 004.4

Покращення ефективності методів та засобів моделювання контенту в web-рішеннях та сервісах

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації магістра

Нормоконтроль

_____Стисло О.В.

(підпис, дата, розшифрування підпису)

Студент

_____ Рущак О.Я.

(підпис, дата, розшифрування підпису)

Допускається до захисту

Завідувач кафедри

_____к.т.н., доц. Ващишак С.П.

(підпис, дата, розшифрування підпису)

Керівник роботи

_____ к.ф-м.н, доц. Бойчук А.

(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «магістр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« 19 » лютого 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рущаку Олексію Ярославовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Покращення ефективності методів та засобів моделювання контенту в web-рішеннях та сервісах

керівник роботи:
Бойчук Андрій Михайлович, кандидат фізико-математичних наук, доцент

затверджена наказом вищого навчального закладу від « 26 » червня 2023 року № 32/1 с

2. Термін подання студентом роботи 16.02.2024

3. Вихідні дані роботи: Формальні моделі, методи та алгоритми.

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)
   1. Аналіз та опис підходів web-розробки на основі моделей
   2. Дослідження засобів та підходів моделювання процесів розробки web-додатків та контенту
   3. Імплементація моделі CMS-рішення для організації web-контенту
   4. Моделювання шаблону web-сервісу на основі CMS

5. Дата видачі завдання 29.06.2023

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз та опис підходів web-розробки на основі моделей	26.09.2023	Виконано
2.	Дослідження засобів та підходів моделювання процесів розробки web-додатків та контенту	20.10.2023	Виконано
3.	Імплементація моделі CMS-рішення для організації web-контенту	15.11.2023	Виконано
4.	Моделювання шаблону web-сервісу на основі CMS	30.11.2023	Виконано
5.	Формування висновків	09.12.2023	Виконано
6.	Оформлення пояснювальної записки	22.12.2023	Виконано
7.	Оформлення графічного матеріалу та підготовка до захисту роботи	11.01.2024	Виконано

Студент _____ Рущак О.Я.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бойчук А.М.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
21	Зв'язок між моделлю та мета моделлю	57	Огляд аспектів і проблем щодо використання CMS
23	Типи діаграм UML	60	Огляд запропонованого підходу до розробки, орієнтованого на концепцію MDE
25	Залежності між UML і MOF	66	Метамодель ReMMM
26	Модель EMOF як підмножина MOF 2.0	68	Взаємозв'язок між різними моделями CMS-ML

27	Модель CMOF як комбінація EMOF з додатковими засобами	69	Ролі та артефакти моделювання, розглянуті CMS-ML
28	Чотирирівнева архітектура метамоделі OMG	71	Метарівні, що використовуються CMS-ML
30	Огляд підходу MDA	73	Представлення, що застосовуються у визначенні шаблону веб-сайту
31	Різниця між інструментами CASE та DSM щодо підтримуваних метарівнів	74	Абстрактний синтаксис для перегляду структури шаблону веб-сервісу
32	Типовий процес DSM	75	Конкретний синтаксис для перегляду структури шаблону сайту
34	Метамоделі QVT та їх зв’язок із MOF та OCL	76	Абстрактний синтаксис перегляду ролей шаблону веб-сайту
35	Огляд архітектури метамоделі QVT	77	Абстрактний синтаксис перегляду дозволів шаблону
36	Приклад конкретного синтаксису для відношення QVTRelation	79	Абстрактний синтаксис для моделі анотацій веб-сервісу
41	Шаблони QVT Core і залежності між ними	81	Конкретний синтаксис для анотацій
50	Огляд аспектів і проблем щодо мов моделювання веб-додатків		

АНОТАЦІЯ

Кваліфікаційна робота присвячена дослідженню процесів покращення ефективності методів та засобів моделювання контенту в web-рішеннях та сервісах шляхом імплементації моделі CMS-рішення для організації web-контенту та використання підходів розробки web-додатків з використанням CMS та концепції Model-Driven Engineering.

В першому розділі здійснено аналіз підходів web-розробки та засобів моделювання web-контенту на основі моделей, описано концепції та підходи розробки web-додатків, наведена концепція моделювання та розробки web-додатків основі моделей. Виконано дослідження та опис стандартів моделювання згідно концепції OMG, приведена методологія проектування і розробки на основі методів Domain-Specific Modeling (DSM).

В другому розділі виконано моделювання процесів ефективної розробки web- контенту та додатків, досліджено процеси розробки з використанням моделі Query-View-Transformation та підходи розробки web-додатків на основі моделей. Здійснено порівняльний аналіз систем управління контентом (CMS) для розробки web-додатків з використанням моделі.

В третьому розділі виконана імплементація моделі cms-рішення для моделювання контенту в web-рішеннях та сервісах, запропоновано підхід розробки web-додатків з використанням CMS та концепції Model-Driven Engineering. Представло робочий процес розробки та метамоделі Revised Metamodel for Multiple Metalevels, описано сутність використання мови CMS-ML для розробки веб-додатків. Проведена реалізація метарівнів та архітектури концепції CMS-ML та моделювання шаблону web-сервісу на основі CMS.

КЛЮЧОВІ СЛОВА: WEB-КОНТЕНТ, РОЗРОБКА НА ОСНОВІ МОДЕЛІ, СИСТЕМА КЕРУВАННЯ КОНТЕНОМ, МОДЕЛЮВАННЯ ШАБЛОНУ, МЕТАМОДЕЛЬ, ГЕНЕРАЦІЯ КОДУ, WEB-ДОДАТОК.

SUMMARY

The qualification work is dedicated to researching the processes of improving the effectiveness of content modeling methods and tools in web solutions and services through the implementation of the CMS solution model for the organization of web content and the use of web application development approaches using CMS and the concept of Model-Driven Engineering.

In the first section, the analysis of web development approaches and means of modeling web content based on models is carried out, the concepts and approaches of web application development are described, the concept of modeling and development of web applications based on models is presented. The study and description of modeling standards according to the OMG concept was carried out, the design and development methodology based on Domain-Specific Modeling (DSM) methods was presented.

In the second section, modeling of the processes of effective development of web content and applications was carried out, development processes using the Query-View-Transformation model and approaches to developing web applications based on models were investigated. A comparative analysis of content management systems for the development of web applications using the model was carried out.

In the third section, the implementation of the cms-solution model for modeling content in web solutions and services is carried out, the approach of developing web applications using CMS and the concept of Model-Driven Engineering is proposed. It presented the workflow of Revised Metamodel for Multiple Metalevels development and metamodels, described the essence of using the CMS-ML language for the development of web applications.

KEY WORDS: WEB CONTENT, MODEL BASED DEVELOPMENT, CONTENT MANAGEMENT SYSTEM, TEMPLATE MODELING, METAMODEL, CODE GENERATION, WEB APPLICATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CASE - Computer-Aided Software Engineering

CMS - Content Management System

CMS-IL - CMS Intermediate Language

CMS-ML - CMS Modeling Language

CRUD - Create, Read, Update, and Delete

CSS - Cascading Style Sheet

DBMS - Database Management System

DSL - Domain-Specific Language

DSM - Domain-Specific Modeling

HTML - HyperText Markup Language

IEEE - Institute of Electrical and Electronics Engineers

MDE - Model-Driven Engineering

MOF - Meta Object Facility

MOFM2T - MOF Model To Text Transformation Language

MoMM - Metamodel for Multiple Metalevels

MYNK - Model sYNchronization frameworK

OCL - Object Constraint Language

OMG - Object Management Group

OOP - Object-Oriented Programming

UI - User Interface

UML - Unifed Modeling Language

QVT - Query/View/Transformation

RCS - Revision Control System

ReMMM - Revised Metamodel for Multiple Metalevels

RSS - Really Simple Syndication

SQL - Standard Query Language

URL - Universal Resource Locator

WYSIWYG - What You See Is What You Get

XMI - XML Metadata Interchange

XML - eXtensible Markup Language

XSL - eXtensible Stylesheet Language

XSLT - XSL Transformations

ВСТУП

Актуальність теми дослідження. З поширенням Інтернету та Всесвітньої павутини в останні роки ми стали свідками бурхливого зростання популярності веб-додатків. Хоча в деякій літературі веб-додаток визначається як виконувана програма, яка широко використовує веб-концепції та технології, у цій роботі ми вибираємо подібне визначення, яке набуває популярності: веб програма складається з програми, яка використовує веб-технології та доступ до якої використовується через веб-браузер. Окремий сегмент технологій для веб-додатків, системи керування вмістом (CMS), нещодавно набув особливої актуальності, оскільки вони полегшують розповсюдження широкого спектру вмісту нетехнічними користувачами. Хоча деякі з цих систем CMS також забезпечують підтримку розробки більш складних веб-додатків, які базуються на них, ця розробка все ще виконується за допомогою традиційних методів, орієнтованих на вихідний код.

З іншого боку, подібно до того, як парадигми розробки програмного забезпечення змінювалися протягом останніх десятиліть, поточна парадигма розробки змінюється від поточних мов програмування третього покоління до інженерії, керованої моделлю (MDE). Ця все більш популярна парадигма, яка спрямована на досягнення вищого рівня абстракції, ніж те, що є загальноприйнятим сьогодні, виступає за використання моделей як найважливіших артефактів у процесі розробки програмного забезпечення. Інші артефакти, такі як вихідний код і документація, можуть бути автоматично створені з цих моделей за допомогою механізмів перетворення моделі.

Типовий контент в Інтернеті змінився за останні роки. Мережа починалася з малих особистих домашніх сторінок, а на сьогодні ми маємо величезні пошукові і торгові портали. В свій час, блоги та вікіси принесли в

Інтернет новий тип вмісту. Цю зміну дехто сприйняв, як таку різку зміну, що вважав її новою версію Інтернету.

Проте, вміст у Web часто є більш дрібним, ніж повні веб-сторінки, отже термін micro-content з'явився для цього нового набору адресного вмісту, що складається з тегів (окремих термінів), коментарів (часто не більше одного абзацу), дописів в блозі (часто близько половини сторінки), зображення (включаючи метадані та назву) або фрагменти відео.

З точки зору загальної концепції CMS, структура опису ресурсів (RDF) є основним форматом представлення знань у семантичній мережі. Це було визначено від початку як формат для опису метаданих про ресурси в Інтернеті. Модель RDF, та бібліотеки програмування, які його реалізують, пропонують хороші способи маніпулювання для метаданих ресурсів, але їм бракує способів опису, доступу або зміни вмісту ресурсів. Насправді модель RDF навіть не має поняття вмісту загалом. Другою проблемою RDF є відсутність інструментів для створення нового контенту. Ці засоби можуть бути розділені на загальні засоби розробки RDF, де користувач також може визначити schema's під час виконання, та інший клас інструментів, які дозволяють створювати лише дані за встановленою схемою, наприклад редактор адресної книги, який виводить її дані у фіксованому форматі RDF.

Відсутність загальних засобів розробки для RDF, ймовірно, пов'язана з відсутністю деяких обмежень, які би покращили зручність використання. Так приклад, засобам розробки зараз доводиться мати справу з моделями RDF, які не містять міток, тому елементи можуть відображатися лише як незручні URI. Популярні інструменти редагування вмісту та структури, як-от інструменти ментальних карт або контури дозволяють кінцевим користувачам досить легко створювати складні моделі. Однак вони не мають гнучкості RDF на рівні дуг довільного типу в графовій моделі контенту . У багатьох інструментах, створення розумних карт, користувач фактично може редагувати лише чіткі дерева.

Інструментам, які виходять за рамки цього й дозволяють повне моделювання прямокутників і стрілок, усе ще бракує розумності, яку може забезпечити онтологічний спосіб висновку поверх RDF. Інші структуровані системи керування контентом (CMS) стикаються з тими ж проблемами: недостатня гнучкість і недостатня інтелектуальність рішення.

Деякі підходи спрямовані на об'єднання RDF і CMS для отримання користі від моделювання кінцевого користувача. Прикладами таких інструментів є iMapping та Conceptual Data Structuring (CDS). Вони стикаються з проблемою низької підтримки інструментів або його відсутності загалом. Проте, наразі немає жодної моделі, яка б дозволяла посилатися на метадані (RDF) і вміст (двійковий), ані інструментів, що реалізує таку модель загалом.

Мета і завдання дослідження. Мета кваліфікаційної роботи полягають в аналізі підходів web-розробки на основі моделей, описі концепцій та підходів розробки web-додатків, концепцій моделювання та розробки web-додатків основі моделей, дослідження та опис стандартів та методологій моделювання контенту web-ресурсів.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- виконати аналіз підходів web-розробки на основі моделей;
- дослідити методологію проектування і розробки на основі методів Domain-Specific Modeling;
- виконати порівняльний аналіз систем управління контентом (CMS) для розробки web-додатків з використанням моделі;
- провести аналіз засобів та підходів моделювання процесів розробки web-додатків та контенту;
- здійснити моделювання шаблону web-сервісу на основі CMS;
- виконати імплементацію моделі CMS-рішення для організації web-контенту.

Об’єктом дослідження є алгоритми та моделі рішень для семантичної організації web-контенту та метамоделі з використанням мови CMS-ML. самі методи та засоби моделювання контенту в web-рішеннях та сервісах.

Предметом дослідження покращення ефективності методів та процесів імплементації моделі CMS-рішення для організації web-контенту та підходи розробки web-додатків з використанням концепції Model-Driven Engineering.

Методи дослідження базуються на використанні методів Model-Driven Engineering, CMS-ML, метамоделі ReMMM, методів стандартів моделювання OMG, методів проектування і розробки на основі DSM.

Наукова новизна одержаних результатів полягає у тому, що на основі аналізу предметної області запропоновано удосконалення DE-орієнтованого підходу до розробки веб-додатків на основі моделей CMS, який відрізняються від інших використанням кількох мов моделювання, а також використанням механізму синхронізації моделі для забезпечення узгодження коду та моделі. Запропонований підхід базується на двох CMS-орієнтованих мовах, CMS-ML і CMS-IL, які розміщені на різних рівнях абстракції.

Практичне значення одержаних результатів полягає в розробці архітектурних шаблонів веб-ресурсів які призначені для створення абстракцій (моделей) конкретних веб-додатків за допомогою елементів, орієнтованих на CMS, а набори інструментів використовують загальні елементи моделювання для створення нових елементів моделювання, орієнтованих на CMS.

Структура. Кількість розділів – 3. Загальний обсяг основної частини – 90 сторінка. Список використаних джерел містить – 50 позицій.

РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ WEB-РОЗРОБКИ ТА ЗАСОБІВ МОДЕЛЮВАННЯ WEB-КОНТЕНТУ НА ОСНОВІ МОДЕЛЕЙ

1.1 Опис концепцій та підходів розробки web-додатків

Інтернет став потужною платформою для розгортання різноманітних артефактів і систем, а також змінив спосіб розробки програм. Цей перехід у розробці – від багатофункціональних програм для робочого столу до веб-додатків, доступних за допомогою звичайних веб-переглядачів – призвів до появи безлічі веб-орієнтованих фреймворків і бібліотек, які дозволяють розробникам використовувати потужність Інтернет-технологій для досягнення різного роду цілей. Основна відмінність між фреймворком (також відомим як платформа в контексті веб-додатків) і бібліотекою полягає в тому, що остання є просто кодом, який розробник може (повторно) використовувати в програмі

Фреймворк зазвичай надає ряд властивостей, яким розробник може надати функціональність і загальну функціональність, яку розробник може змінити або налаштувати. Прикладами добре відомих веб-орієнтованих фреймворків є ASP.NET від Microsoft, JavaEE і Java Server Pages (JSP) або Ruby on Rails (RoR) .

Дедалі популярнішим результатом цього переходу є багато веб-орієнтованих CMS (систем керування вмістом) і ECM (ерування корпоративним вмістом), які зараз доступні, які мають основну метою полегшення керування та публікації цифрового вмісту в інтрамережі чи навіть в Інтернеті.

Через різні цілі кожного фреймворку (що, у свою чергу, впливає на характер функціональності, яку він надає), вони часто супроводжуються підходами до розробки простих веб-додатків на основі цього фреймворку. Однак Інтернет змінив місце, де ми розгортаємо програми, але не те, як ми

розробляємо ці програми, оскільки їх розробка все ще виконується вручну. З іншого боку, нова парадигма MDE (Model-Driven Engineering) спрямована на те, щоб усунути значення, яке поточні процеси розробки програмного забезпечення все ще надають вихідному коду, і натомість зосередитися на моделях і концепціях з проблем і цільові домени.

Підходи на основі моделей. Ці підходи стають дедалі популярнішими завдяки появі парадигми інженерії, керованої моделлю (MDE). Відомо, що MDE виступає за використання моделей як основних артефактів у процесі розробки програмного забезпечення, тоді як артефакти, такі як документація та вихідний код, можуть бути створені з цих моделей за допомогою автоматизованих перетворень моделі. Така орієнтація на моделі дозволяє розробникам зосереджуватися на відповідних концепціях (замість того, щоб зосереджуватися на вихідному коді та деталях реалізації, які з точки зору кінцевого користувача можуть вважатися несуттєвими), що, у свою чергу, дозволяє ефективніше керувати постійно зростаючою складністю програмне забезпечення. Окрім того, що більшість повторюваних і схильних до помилок завдань залишаються на перетворення моделі, підходи MDE також мають додаткові переваги, такі як:

- відмежування розробників від таких проблем, як базова складність платформи або нездатність мов програмування виражати концепції домену;
- націлювання на кілька платформ розгортання без потреби в різних базах коду.

Хоча деякі приклади підходів MDE існують – найвідомішим з яких є, мабуть, модельно-керована архітектура (MDA), більшість розробників все ще використовують MDE в контексті доменно-орієнтованого моделювання (DSM), у якому:

- візуальна мова визначена та орієнтована на певну (переважно просту) область,

- розроблені моделі будуть використовуватися для автоматичного створення відповідного вихідного коду.

Таким чином, сам процес розробки програмного забезпечення все ще орієнтований на вихідний код, хоча деякі його частини спрощені DSL.

Підходи на основі CMS. Хоча зазвичай це не є їх основною метою, системи керування вмістом (CMS) можуть використовуватися не лише для створення веб-сайтів, але й як допоміжні платформи для розгортання спеціалізованих веб-додатків. Перш ніж продовжити, важливо встановити різницю між поняттями веб-додаток і веб-сайт.

Веб-додаток — це програма, яка використовує веб-технології (як було зазначено у вступі), тоді як веб-сайт складається з конкретного використання веб-додатку. Іншими словами, веб-сайт розглядається як конкретне розгортання веб-додатку (наприклад, веб-сайт пошуку Google – це розгортання веб-додатку, який використовує програмне забезпечення пошукової системи, розроблене компанією).

Ці системи CMS (які самі по собі є веб-додатками) можуть ефективно підтримувати динамічне керування веб-сайтами та їхнім вмістом і зазвичай мають такі аспекти:

- розширюваність і модульність;
- незалежність між вмістом і презентацією;
- підтримка кількох типів вмісту;
- підтримка керування доступом і керування користувачами;
- підтримка визначення та виконання робочого процесу.

З іншого боку, системи керування корпоративним вмістом (ECM) — це зазвичай веб-додатки, орієнтовані на використання Інтернет-технологій і робочих процесів для захоплення, керування, зберігання, збереження та доставки вмісту й документів у контексті організаційних процесів.

Тим не менш, ці дві області не є роз’єднаними і нерідко можна знайти звичайну систему CMS, яка діє як сховище для документів і вмісту організації, хоча й на дещо примітивному рівні (наприклад, без перевірки

дублікатів інформації, без логічного групування). документів і немає підтримки надання метаданих для кожного документа). Крім того, платформу CMS можна використовувати для виконання тих самих завдань, що й платформу ECM, за умови, що CMS надає розробнику відповідну функціональність і підключення. Фактично, деякі системи CMS розвиваються, щоб самі стати справжніми фреймворками, наділяючи розробників набором гачків і функцій, достатніх для створення веб-додатків з цілями, відмінними від простого керування вмістом. Окрім цих функцій, системи CMS також надають набір концепцій високого рівня, таких як користувач, роль або модуль, які полегшують розробку складних веб-додатків.

1.2 Концепції моделювання та розробки web-додатків основі моделей

Щоб покращити спосіб розробки програмного забезпечення, необхідно, щоб зацікавлені сторони програмного забезпечення мали можливість виражати свої знання, наміри та цілі в галузі, використовуючи відповідні мови, з якими вони можуть спілкуватися. Такі мови можуть бути графічними (UML) або навіть текстовими (наприклад, традиційні мови програмування, такі як Python, Ruby, C # , Java або багато інших) залежно від таких факторів, як потреба зацікавлених сторін.

Ця потреба в мовах, зручних для зацікавлених сторін, у свою чергу, породжує потребу в механізмах для створення цих мов, щоб підтримувати визначення нових способів створення та визначення моделей, що представляють рішення для проблем. Крім того, тема створення мови тісно пов’язана з постійними зусиллями інженерів програмного забезпечення та дослідників, спрямованих на підвищення рівня абстракції, на якому вони виконують завдання аналізу, проектування та розробки програмного забезпечення.

У контексті даного дослідження ми вивчили та проаналізували такі теми та підходи до мовного дизайну:

1. Інженерія, керована моделлю (MDE).
2. Архітектура аа основі моделі (MDA), підхід групи управління об'єктами.
3. Доменне-специфічне моделювання (DSM).
4. Метамоделювання.
5. Подібності між діяльністю проектування мов моделювання та мов програмування.

Попередньо слід зазначити, що представлений аналіз не є вичерпним. Це пояснюється тим, що деякі аспекти такого аналізу виходять за рамки цієї роботи. Більше того, кожна з цих тем уже була предметом численних наукових досліджень, тому представлення такого аналізу значно розширило роботу, але дало б незначну додаткову цінність.

Системи програмного забезпечення досягають такого високого рівня складності, що навіть поточні мови програмування третього покоління, такі як Java або C # , стають нездатними ефективно підтримувати створення таких систем. Однією з проблем таких сучасних мов є те, що вони все ще надто орієнтовані на визначення того, як має працювати рішення, а не на те, яким має бути рішення. Це призводить до потреби в механізмах і техніках, які дозволяють розробнику абстрагуватися від самої мови програмування та зосередитися на створенні простого вирішення певної проблеми.

Останніми роками було докладено значних дослідницьких зусиль, щоб підняти рівень абстракції в інженерію, керовану моделлю (MDE). MDE, яку іноді називають розробкою, керованою моделлю (MDD), є новою парадигмою, заснованою на систематичному використанні моделей як основи специфікації рішення. На відміну від попередніх (на основі вихідного коду) парадигм розробки програмного забезпечення, моделі стають найважливішими об’єктами, і такі артефакти, як вихідний код або документація, можуть бути отримані з цих моделей в автоматизований спосіб,

звільняючи розробників від таких проблем, як складність базової платформи або нездатність мов третього покоління для вираження понять домену. MDE не є новою ідеєю. Фактично, інструменти автоматизованої розробки програмного забезпечення (CASE) вже були зосереджені на забезпеченні розробників методами та інструментами для проектування систем програмного забезпечення з використанням графічних представлень мови загального призначення. Потім розробник зможе виконувати різні завдання над цими представленнями, наприклад перевірку або перетворення коду. Однак ці інструменти CASE не змогли виконати заплановану мету через такі проблеми, як:

- погане відображення мов загального призначення на базових платформах, через що генерований код набагато важче зрозуміти та підтримувати;
- неможливість масштабування, оскільки вони не підтримували паралельний розвиток;
- вихідний код все ще був найважливішою сутністю в процесі розробки, тоді як моделі розглядалися як такі, що підходять лише для документування.

Сьогодні ситуація більш сприятлива для успіху парадигми MDE. Це через згадану вище нездатність мов третього покоління вирішувати зростаючий ступінь складності поточних систем програмного забезпечення в поєднанні з вибором доступних на даний момент платформ розробки (наприклад, Java), на які можна відобразити моделі.

В даний час існує значна кількість підходів, орієнтованих на MDE, з яких ми виділяємо модельно-керовану архітектуру (MDA) і предметно-орієнтоване моделювання (DSM). Важливо зазначити, що MDE не визначає жодного конкретного підходу, натомість сам MDE є парадигмою, яка розглядається за допомогою таких підходів, але не залежить від мови чи технології.

Тим не менш, усі підходи, орієнтовані на MDE, поділяють однакові основні концепції. Модель – це інтерпретація певної області – фрагменту реального світу, на якому зосереджені задачі моделювання та розробки системи – відповідно до визначеної структури понять. Іншими словами, модель можна розглядати як скорочене представлення системи, висвітлюючи властивості, які становлять інтерес відповідно до конкретної точки зору. Його також можна розглядати як систему, яка сама дає відповіді про систему, що аналізується, без необхідності безпосереднього розгляду останньої.

Структура понять, що визначає модель, забезпечується метамоделлю, яка є спробою описати оточення з певною метою через точне визначення конструкцій і правил, необхідних для створення моделей. Це означає, що метамодель надає мову, яку можна використовувати для створення моделі, як показано на рисунку 1.1, так само метамодель, яка визначає мову, якою специфікована інша метамодель, називається метаметамоделлю.

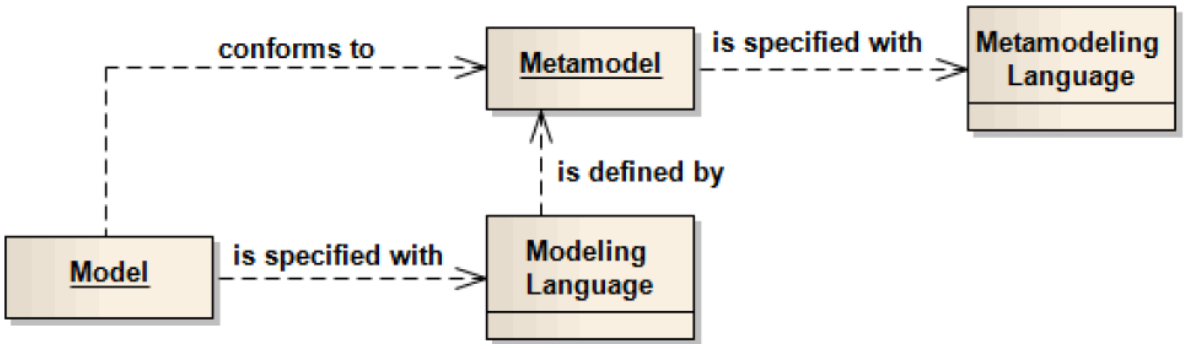


Рисунок 1.1 – Зв’язок між моделлю та мета моделлю

Також розглянемо концепцію metalevel (іноді називається metalayer, а саме в специфікаціях OMG). Часто кажуть, що метамодель знаходиться на метарівні, вищому за метамодель відповідної моделі. Таким чином, визначення стека металевих рівнів забезпечує спосіб ієрархічно структурувати набір моделей по відношенню до їхніх відповідних метамоделей. Концепція метарівня особливо важлива при визначенні архітектур метамоделювання, таких як OMG.

З практичної точки зору та з огляду на рисунок 1.1. варто зазначити, що більшість інструментів моделювання зазвичай мають справу лише з одним логічним рівнем (тобто редагування моделі користувача та жорстко закодована метамодель). Це пояснюється тим, що створення такого інструменту легко зробити за допомогою типової мови об’єктно-орієнтованого програмування (ООП), використовуючи переваги зв’язку клас–екземпляр і створюючи підтримку логічного рівня рівнем екземпляра.

Тепер, коли ці основні концепції моделювання представлено, ми тепер можемо надати аналіз деяких відповідних підходів і мов MDE відповідно до аспектів, які були визначені на початку цього розділу.

1.3 Дослідження та опис стандартів моделювання згідно концепції Object Management Group (OMG)

Група управління об'єктами (OMG) визначила добре відомий підхід до парадигми MDE. Цей підхід, який називається архітектурою, керованою моделлю (MDA), базується на наборі стандартів OMG, які використовують методи для специфікації мови моделювання та трансформації моделі. Особливо важливим для цієї глави є той факт, що деякі з цих стандартів (а саме MOF і UML, представлені далі) забезпечують механізми для визначення нових мов моделювання.

Уніфікована мова моделювання (UML). Надбудова уніфікованої мови моделювання (UML) є мовою моделювання загального призначення. Спочатку розроблений для специфікації, візуалізації та документування систем програмного забезпечення, він також використовується як основа для різноманітних інструментів, методологій та дослідницьких зусиль щодо розробки програмного забезпечення та створення вихідного коду.

UML складається з 14 різних типів діаграм (як показано на рисунку 1.2), розділених на дві категорії:

- структурні діаграми, які стосуються зі статичною структурою об'єктів в системі, що моделюється;
- діаграми поведінки, які визначають динамічні аспекти та поведінку об'єктів системи.

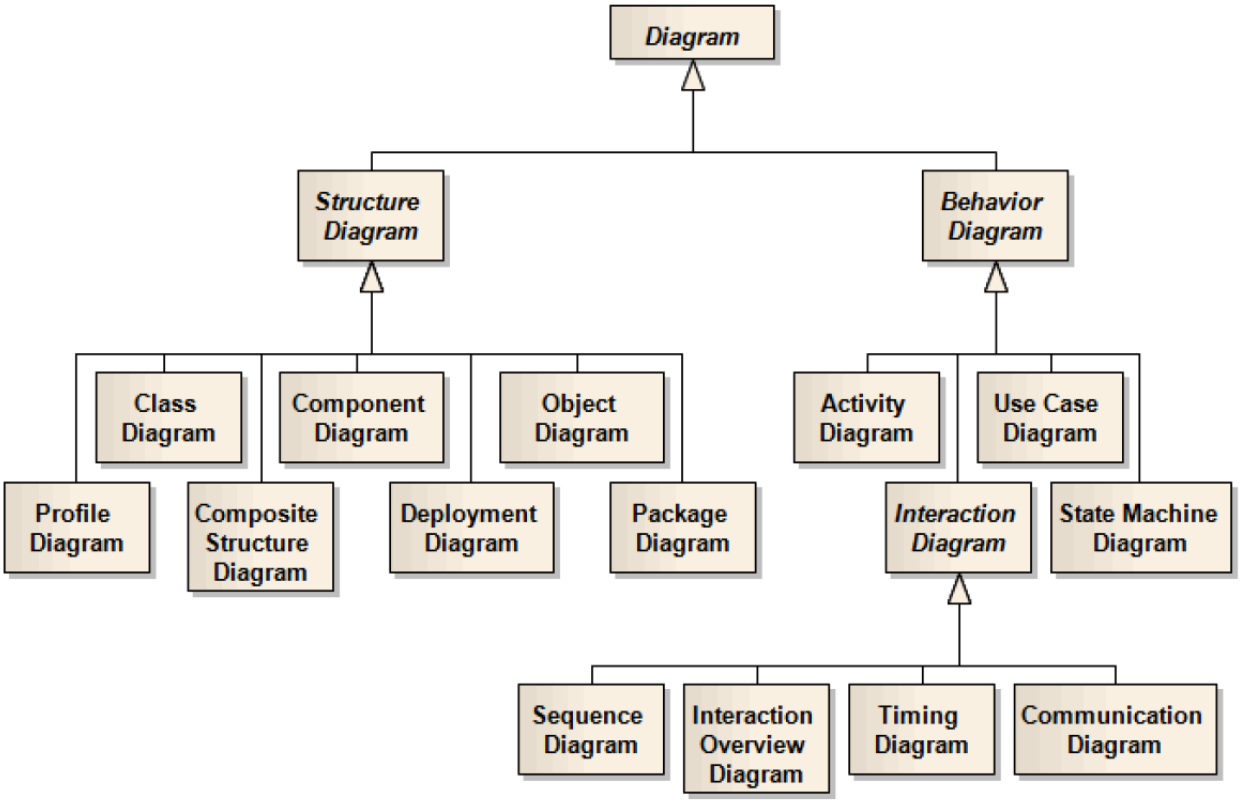


Рисунок 1.2 – Типи діаграм UML

UML визначає такі структурні діаграми:

- діаграма класів;
- діаграма компонентів;
- діаграма складеної структури;
- діаграма розгортання;
- діаграма об’єкта;
- діаграма пакета;
- діаграма профілю.

Він також визначає такі діаграми поведінки:

- діаграма діяльності;

- діаграма зв’язку;
- діаграма огляду взаємодії;
- діаграма послідовності;
- діаграма кінцевого автомата;
- діаграма часу;
- діаграма варіантів використання.

Незважаючи на те, що UML був кроком вперед у встановленні стандарту, зрозумілого всій спільноті розробників програмного забезпечення і узгодженні його з MDE, його все ще критикують з таких причин, як:

1. простота використання в сферах програмного забезпечення, таких як IT або телекомунікації, але не підходить до інших сфер, таких як біологія чи фінанси;

2. не орієнтується на те, як це буде використано на практиці;

3. може бути надто складним в цілому об’єкті.

Тим не менше, UML часто стає об’єктом реклами, яка підвищує очікування користувачів до недосяжного рівня, що зазвичай впливає на наступну критику. Прикладом такої критики є те, що стосується труднощів у використанні UML для моделювання областей, не пов’язаних із програмним забезпеченням: хоча UML є мовою моделювання загального призначення, вона орієнтована на моделювання програмних систем, а не призначена для моделювання кожної та кожен домен.

UML традиційно використовується як метамодель (тобто розробники створюють моделі, використовуючи мову встановлену UML). Однак специфікація UML також надає механізм профілю, який дозволяє визначати нові нотації або номенклатури, забезпечуючи спосіб розширення метакласів UML (таких як Class або Association) і адаптації їх для різних цілей. Профілі можна розглядати як сукупність стереотипів, позначених значень і обмежень. Стереотип визначає додаткові властивості елемента, але ці властивості не повинні суперечити властивостям, які вже пов’язані з оригінальним елементом моделі. Таким чином, профіль не дозволяє користувачеві

редагувати існуючу метамодель UML, а лише розширити її, зробивши більш конкретною.

Meta Object Facility (MOF). Meta Object Facility (MOF) разом із UML є основою підходу OMG до MDE. UML і MOF були розроблені, щоб самі бути екземплярами MOF. Це було досягнуто шляхом визначення бібліотеки інфраструктури UML яка забезпечує структуру моделювання та нотацію для надбудови UML і MOF, а також може використовуватися для інших метамоделей.

Рисунок 1.3 ілюструє залежність між UML і MOF при цьому OF можна описати за допомогою самого себе, що робить його рефлексивним. Окрім UML, OMG також визначив інші стандарти на основі MOF, такі як XML Metadata Interchange (XMI) і Query/View/Transformation (QVT).

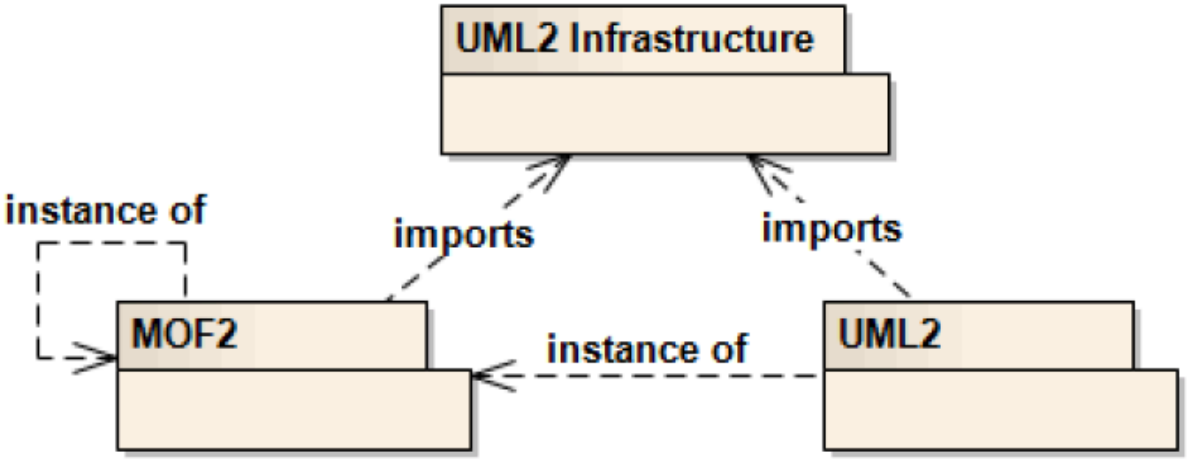


Рисунок 1.3 – Залежності між UML і MOF

Коли користувачі посилаються на MOF, вони насправді вказують на один із двох варіантів метамоделі MOF, EMOF або CMOF.

EMOF (Essential MOF) — це підмножина MOF, яка особливо підходить для використання в області підходів до об’єктно-орієнтованого моделювання і XML. Таким чином, EMOF дозволяє сценарії, в яких моделі MOF відображаються в документах XML (наприклад, з використанням формату XMI, представленого нижче в цьому розділі) або навіть в концепціях

об'єктно-орієнтованого програмування. На рисунку 1.4 представлено огляд об’єктів, визначених у контексті EMOF.

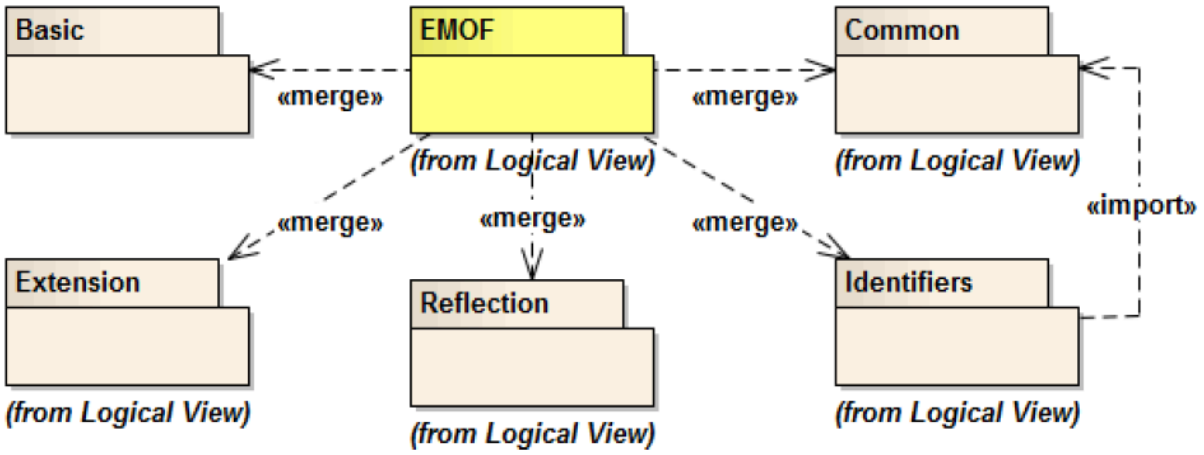


Рисунок 1.4 – Модель EMOF як підмножина MOF 2.0

З іншого боку, коли розробники застосовують MOF, зазвичай мають на увазі CMOF (Complete MOF). CMOF є результатом об’єднання (або об’єднання за допомогою оператора Package Merge) EMOF з його розширеннями, які забезпечують перевизначення елементів, що робить CMOF адекватним для основних потреб метамоделювання.

На рисунку 1.5 показано CMOF як комбінацію EMOF з цими додатковими засобами.

Обмін метаданими XML (XMI) дозволяє обмінюватися – за допомогою XML – будь-які метадані, метамодель яких можна вказати в MOF. Це дозволяє відображати будь-яку метамодель на основі MOF у XML, забезпечуючи портативний спосіб серіалізації та обміну моделями між інструментами. Тим не менш, користувачі часто вважають XMI останнім засобом для обміну моделями, оскільки інструменти часто використовують власні розширення XMI від постачальника, потенційно втрачаючи інформацію під час обміну моделями між інструментами від різних постачальників.

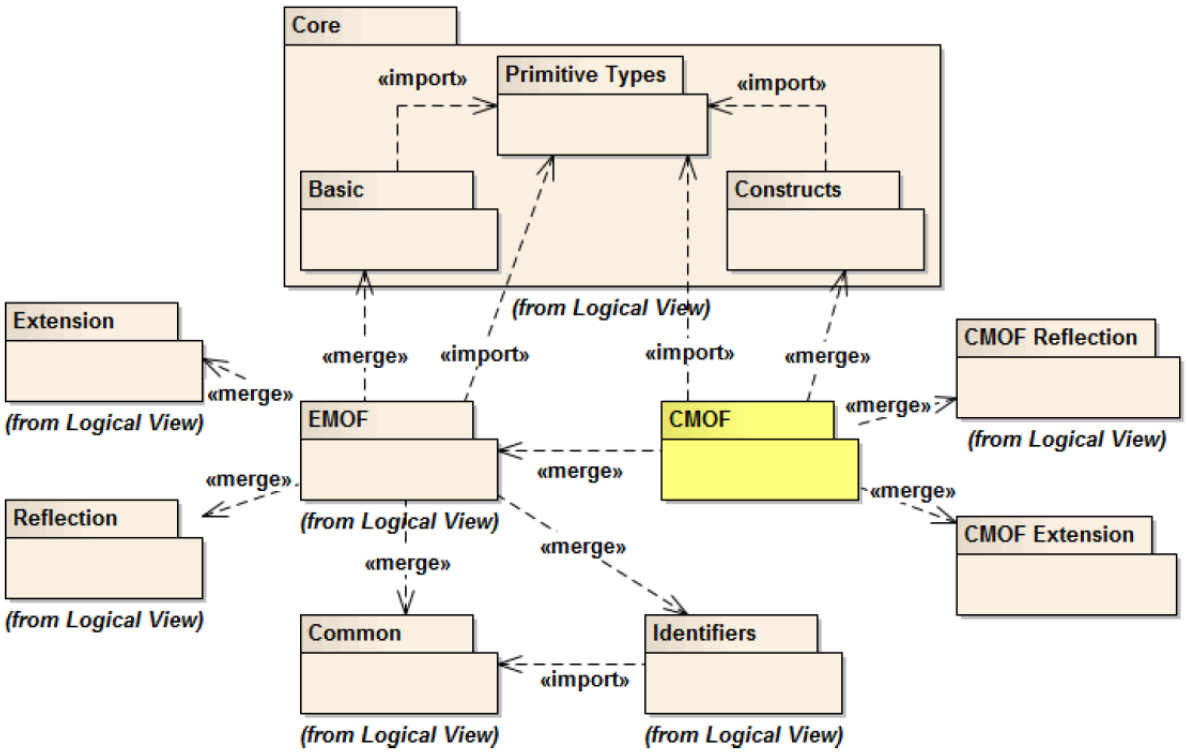


Рисунок 1.5 – Модель CMOF як комбінація EMOF з додатковими засобами

З іншого боку, специфікація QVT визначає спосіб перетворення вихідних моделей у цільові моделі, дозволяючи визначення запитів, переглядів і операцій перетворення над моделями. Однією з найцікавіших ідей щодо QVT є те, що саме перетворення є моделлю на основі MOF, що призводить до сумісності QVT з MOF.

Особливо важливим стандартом для мов на основі QVT і MOF є мова обмеження об’єктів (OCL). Хоча спочатку він був тісно пов’язаний з UML, з тих пір його сфера розширилася і OCL став частиною підходу OMG, заснованого на MOF. OCL — це текстова декларативна мова, яка використовується для визначення правил і запитів для моделей і метамоделей на основі MOF (включаючи UML), коли доступні діаграмні оператори не мають достатньої виразності для визначення таких правил.

MOF було розроблено як основу підходу OMG. OMG визначила чотирирівневу архітектуру метамоделі, засновану головним чином на MOF і UML, яка не тільки дозволяє користувачам визначати власні UML-моделі, але

й дозволяє визначати подальші мови на основі MOF, такі як метамодель Common Warehouse (CWM).

На рисунку 1.6 зображено цю архітектуру:

- MOF є метаметамоделлю на метарівні M3;
- UML – екземпляр MOF – це метамодель на метарівні M2;
- модель користувача містить елементи моделі та знімки екземплярів цих елементів моделі в певний час або стан, визначений за допомогою UML, на метарівні M1;
- метарівень M0 містить екземпляри часу виконання елементів моделі, визначені на метарівні M1.

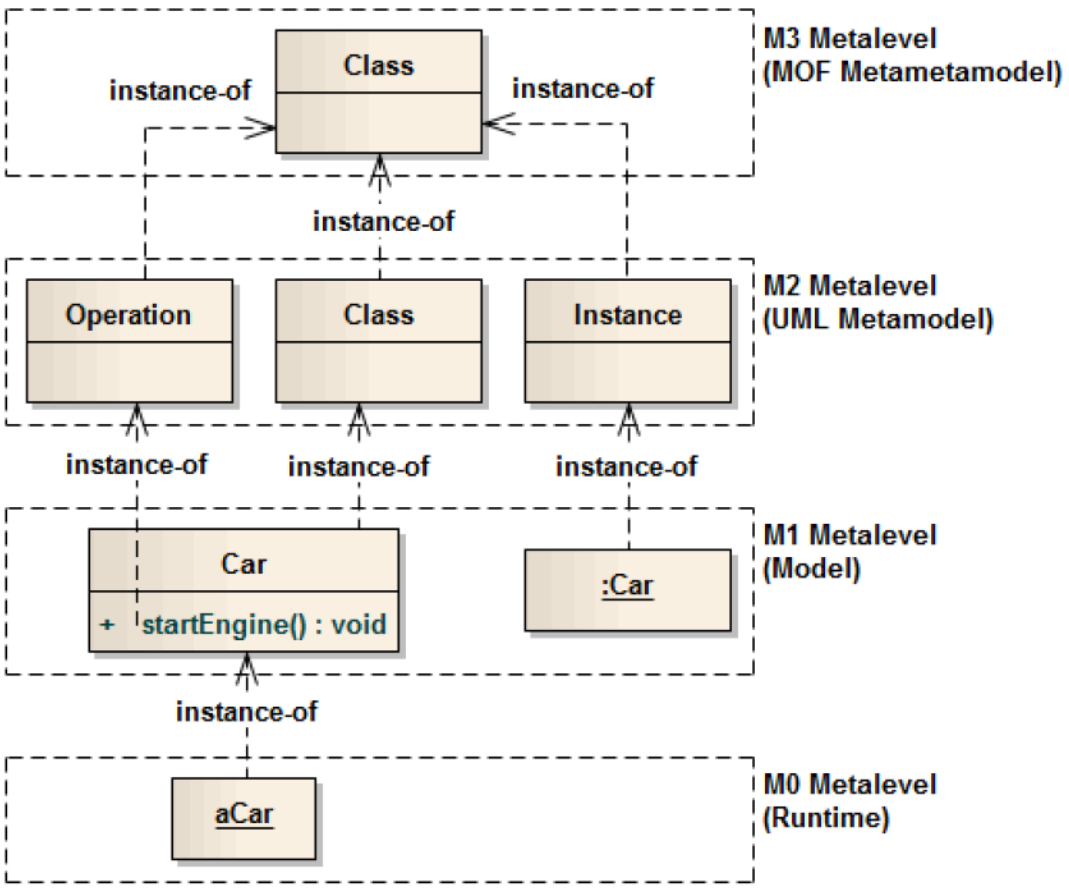


Рисунок 1.6 - Чотирирівнева архітектура метамоделі OMG

Як показано на рисунку 1.6, MOF відіграє роль метаметамоделі (тобто забезпечує мову метамоделювання, згідно з рисунком 1.1). Таким чином,

поняття, визначені MOF (а саме EMOF) не призначені для визначення моделей програмних систем, а скоріше для визначення нових понять у метамоделі (як у випадку на метарівні M2, що містить метамодель UML), створюючи таким чином нову мову моделювання. Ці нові концепції повинні бути адекватними для моделювання передбачуваної області мови.

Крім того, інший варіант MOF був визначений у контексті Eclipse Modeling Framework, який називається ECore, схожий на EMOF, однак механізми обміну між обома мовами були визначені, що дозволяє сценарії, в яких моделі ECore можуть використовуватися генераторами коду (або іншими інструментами обробки моделей), які були розроблені для роботи з моделями MOF.

Архітектура, керована моделлю (MDA). Архітектура, керована моделлю (MDA) – це підхід OMG до розгляду життєвого циклу розробки програмного забезпечення, керованого моделюванням програмної системи. Він в основному базується на стандарті MOF OMG і робить більший акцент на трансформації моделі, ніж на самому метамоделюванні.

MDA визначає три різні види моделі:

1. незалежну від обчислень модель;
2. модель, незалежну від платформи;
3. модель, що залежить від платформи.

Обчислювально-незалежна модель (CIM) — це модель, яка відображає вимоги до системи, а саме шляхом визначення характеристик середовища, в якому система працюватиме, і опису того, що система повинна робити. З іншого боку, платформно-незалежна модель (PIM) — це модель із високим рівнем абстракції та повністю незалежна від будь-якої технології реалізації. Це робить її придатною для опису програмної системи із загальної точки зору, без розгляду реалізації або технологічних деталей, таких як конкретні реляційні бази даних або сервери додатків. Нарешті, платформо-специфічна модель (PSM) також є моделлю програмної системи; однак, на відміну від PIM, PSM специфікується з точки зору технології впровадження.

PIM можна перетворити на PSM за допомогою перетворень від моделі до моделі. Можливість мати декілька PSM, отриманих з одного PIM, пов’язана з тим, що для поточних систем програмного забезпечення дуже часто використовують кілька технологій (наприклад, сервер веб-додатків, що використовує JBoss у поєднанні з системою баз даних MySQL). MDA передбачає існування правил трансформації, але не визначає, що це за правила; у деяких випадках постачальник може надавати правила як частину стандартного набору моделей і профілів.

На рисунку 1.7 представлено огляд MDA (моделі CIM не представлені): суцільні лінії, що з’єднують прямокутники, є перетвореннями, які визначаються правилами перетворення.

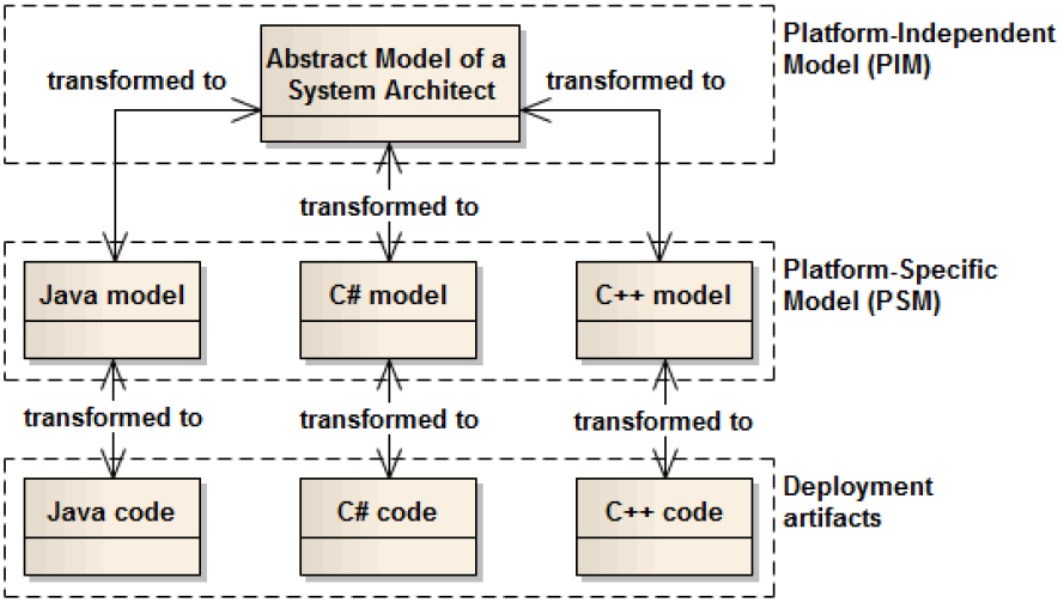


Рисунок 1.7 – Огляд підходу MDA

Однак MDA не є розповсюдженим через такі проблеми:

1. використання UML, мови, яка досі критикується;
2. коли виникають проблеми, розробникам доводиться вручну налагоджувати код, який був автоматично згенерований інструментом;
3. генератори вихідного коду зазвичай здатні створити значну частину програми, але отриманий вихідний код часто вимагає від

розробників значних зусиль для інтеграції цього коду з існуючими специфічними для платформи API (такими як Java або .NET).

1.4 Методологія проектування і розробки на основі методів Domain-Specific Modeling (DSM)

Доменне-специфічне моделювання (DSM) використовує концепції проблемної області як основні блоки моделей, на відміну від традиційних інструментів CASE, які використовують концепції мови програмування (наприклад, Class, Object).

З технологічної точки зору DSM підтримується інструментом DSM, який можна розглядати як додаток для створення предметно-спеціальних CASE-інструментів (іншими словами, середовище для створення CASE-інструментів, які можна використовувати для створення програм).

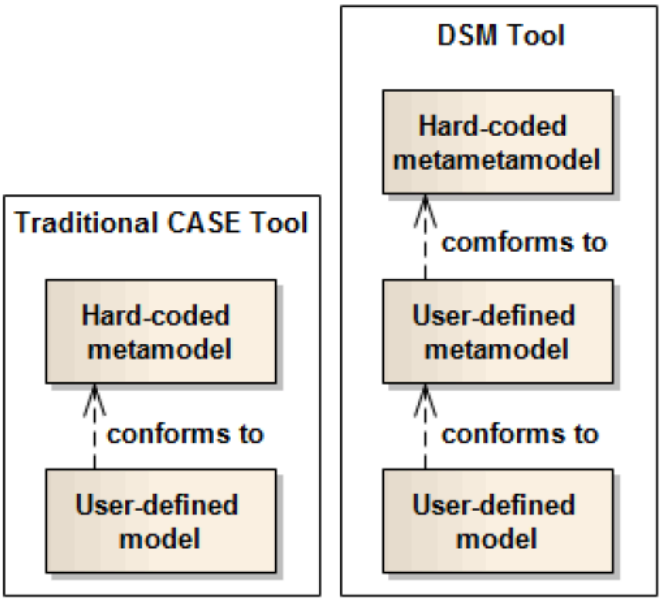


Рисунок 1.8 – Різниця між інструментами CASE та DSM щодо підтримуваних метарівнів

Таким чином, інструменти DSM додають рівень абстракції в порівнянні з традиційним CASE, уможливлюючи предметно-спеціальну конфігурацію

результуючої програми моделювання, як показано на рисунку 1.8. Через цей рівень абстракції інструменти DSM також називають мовними інструментами або інструментами метамоделювання .

DSM тісно пов'язаний з концепцією доменно-специфічної мови (DSL). DSL — це мова, призначена для виконання конкретного завдання (або набору завдань) у певній проблемній області на відміну від мови загального призначення.

Як показано на рисунку 1.9, DSL (тобто метамоделі домену) і відповідні генератори зазвичай визначаються розробниками, які є експертами в проблемній області, через вузькоспеціалізовану природу, яка лежить в основі DSL. Інші розробники, які мають менший досвід зіставлення між концепціями домену та вихідним кодом, тоді виконуватимуть свою роботу, використовуючи DSL.

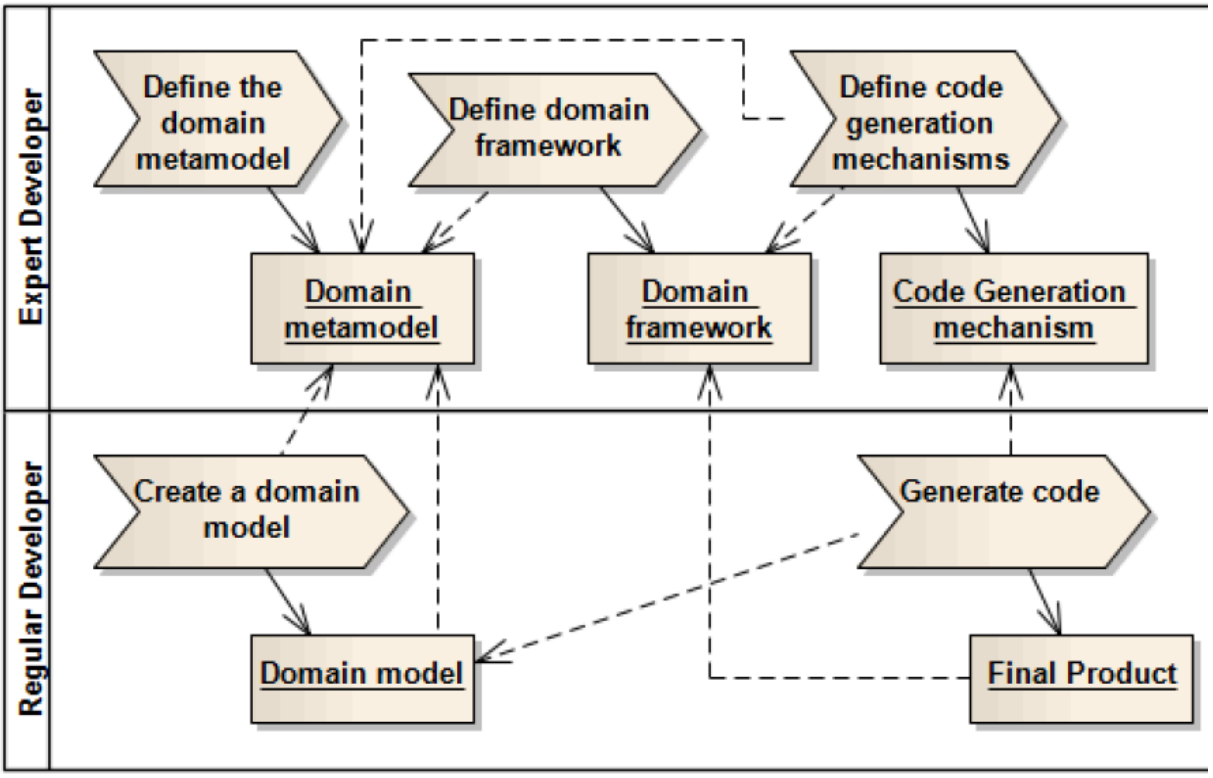


Рисунок 1.9 – Типовий процес DSM: деякі експерти розробляють предметно-орієнтовані інструменти для використання менш досвідченими розробниками

Добре відомим прикладом DSL є стандартна мова запитів (SQL), яка є стандартною текстовою мовою для доступу до баз даних (іншими словами, SQL є текстовою DSL для проблемної області запитів до бази даних і маніпулювання ними). Іншим типовим прикладом DSL є оболонка Unix, орієнтована на область даних і файлів маніпуляції, оскільки він надає низку відповідних понять предметної області і операцій над цими поняттями, наприклад каналів і завдань, таких як підрахунок слів або заміна рядків.

Розробники зазвичай віддають перевагу використанню DSL, а не загальним мовам моделювання, таким як UML через концепції та абстракції, які надає кожна з них. Останній використовує концепції об'єктно-орієнтованого програмування, що розміщує моделі на рівні абстракції трохи вище вихідного коду. З іншого боку, DSL призначений для використання концепцій із проблемної області, що дає змогу розробникам абстрагуватися від того, як ці концепції відображатимуться у вихідному коді, що, у свою чергу, призводить до більшої зосередженості на вирішенні поточної проблеми, замість того, щоб витрачати час на відображення між рішенням проблеми та реалізацією цього рішення. Тим не менш, навіть загальні мови моделювання іноді вважаються DSL, коли вони призначені для використання з певною метою. Саму MOF часто вважають DSL, у якій метою мови є визначення нових мов моделювання.

Висновки до розділу 1

В даному розділі виконано аналіз концепцій пов’язаних із MDE і підходів до розробки програмного забезпечення, що визначені групою керування об’єктами (OMG). Цей розділ також визначає поняття моделі та метамоделі, а також дії метамоделювання та ряд аспектів, які можуть на нього впливати. Також представлені деякі подібності, які можна спостерігати між метамоделюванням і діями, які зазвичай мають місце під час визначення мови програмування.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ПРОЦЕСІВ ЕФЕКТИВНОЇ РОЗРОБКИ WEB- КОНТЕНТУ ТА ДОДАТКІВ

2.1 Дослідження процесів розробки з використанням моделі Query-View-Transformation

Специфікація Query/View/Transformation (QVT) створена як основа підходу MDA OMG і визначає стандартний спосіб перетворення вихідної моделі в іншу цільову модель у якій і вихідна і цільова моделі відповідають довільній метамоделі на основі MOF.

Перетворення QVT визначається як модель, яка визначає запити, перегляди та перетворення над вхідною моделлю. Ця модель QVT визначається відповідно до однієї з трьох метамоделей, наданих QVT:

1. QVTOoperational (або Operational Mapping);
2. QVTRelation (або Relations);
3. QVTCore (або Core).

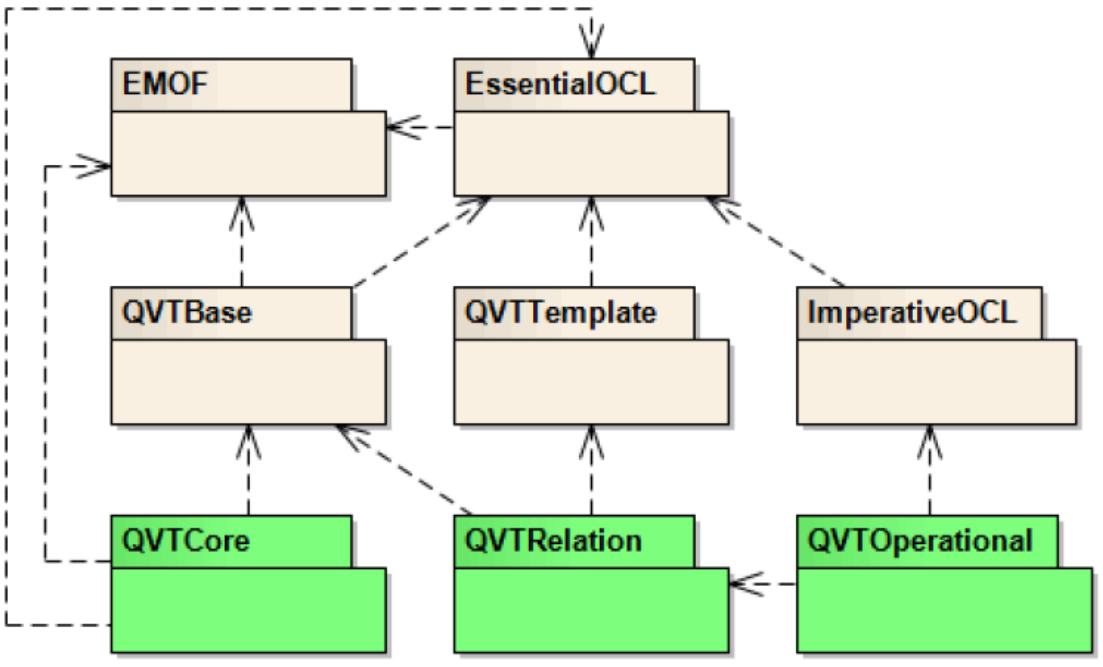


Рисунок 2.1 – Метамоделі QVT та їх зв’язок із MOF та OCL

Перша метамодель, QVTOperational, забезпечує імперативну мову тоді як інші метамоделі визначають декларативні мови. Кожна з них також є MOF-сумісною, тому трансформації моделей на основі QVT також є MOF-сумісними, що дозволяє сценарії, наприклад, зберігати такі трансформації моделі в репозиторії моделей. Рисунок 2.1 ілюструє зв'язок між метамоделями QVT і MOF.

Ці три метамоделі є основою архітектури QVT, зображеної на рисунку 2.2.

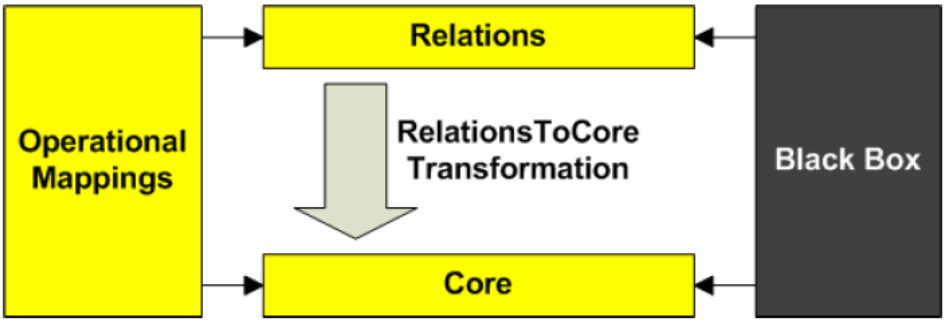


Рисунок 2.2 – Огляд архітектури метамоделі QVT

Перетворення визначаються за допомогою мови зв’язків, а потім перетворюються на мову ядра за допомогою механізму перетворення RelationsToCore . Крім того, QVT також дозволяє використовувати зовнішні утиліти обробки (наприклад, перетворення XML за допомогою XSL) через механізм Black Box .

Як показано на рисунку 2.1, QVT залежить не тільки від MOF, але й від OCL, оскільки останній використовується у всіх метамоделях QVT для визначення предикатів і обмежень. Тим не менш, QVT також розширює OCL за допомогою імперативних функцій, які потім використовуються для надання QVTOperational процедурного стилю програмування.

Метамодель QVTRelation (Relations) дозволяє специфікувати декларативним способом перетворення як набір зв’язків між моделями. Кожне відношення включає два або більше доменів, а також положення where і when (кожне з яких необов’язкове). Домен можна розглядати як типізований

параметр для перетворення (тобто вихідної або цільової моделей). Відношення визначає, для кожного з його доменів, шаблон, який повинен зберігатися (тобто відповідати) якщо перетворення має вважатися успішним. Необхідність, щоб певне відношення зберігалося, визначається його типом: якщо це реляція, то воно має зберігатися інакше воно має виконуватися, лише якщо воно використовується (прямо чи опосередковано) у реченні where іншого відношення, яке має виконуватися. Речення when визначає обставини, за яких має виконуватися відношення – окрім обставин, пов'язаних із контекстом виклику відношення – тоді як речення where визначає умови, які мають бути перевірені елементами моделі, задіяними у відношенні. Якщо порівнювати ці речення з концепціями комп'ютерного програмування, речення when і where схожі на guard.

Більше того, кожен домен є або enforced або checkonly, що вказує на те чи можуть його елементи бути змінені перетворенням чи ні, відповідно (щоб підтримувати відповідний шаблон). Це, у свою чергу, призводить до того факту, що відношення також є або однонаправленим, або двонаправленим, залежно від того, чи всі його домени можуть бути виконані .

Ще одна характеристика цієї мови полягає в тому, що вона неявно встановлює трасування зв'язків між елементами вихідної та цільової моделей.

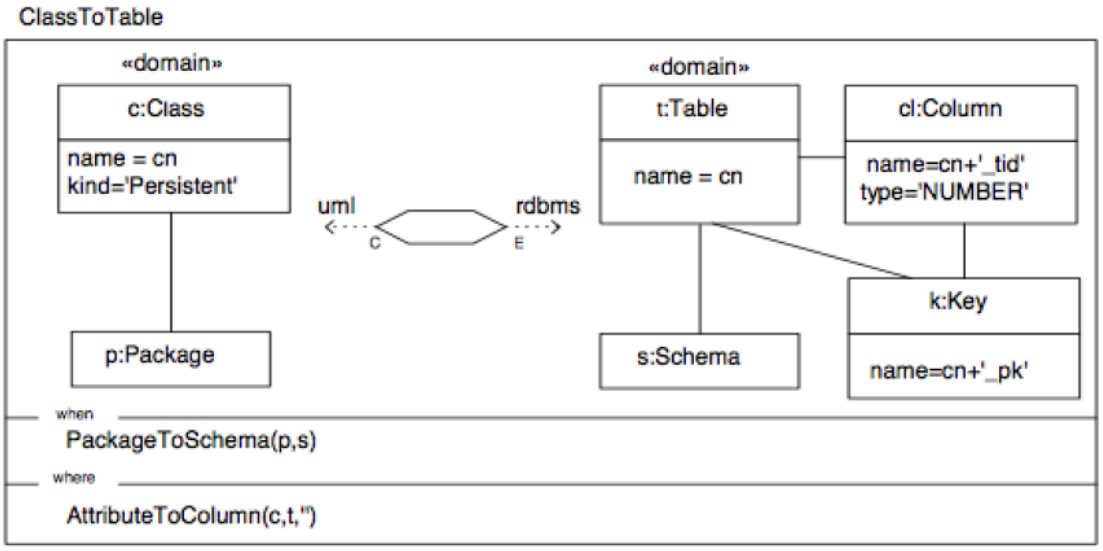


Рисунок 2.3 – Приклад конкретного синтаксису для відношення QVTRelation

QVTRelation надає як графічний, так і текстовий конкретний синтаксис, як запропоновано в лістингу 2.1 і рисунку 2.3: перший містить уривок текстового перетворення QVTRelation, який перетворює модель UML на реляційну модель, тоді як другий надає графічну діаграму відношення еквівалентно відношенню ClassToTable у лістингу 2.1.

Лістинг 2.1. Приклад перетворення QVTRelation:

```
transformation umlToRdbms(uml:SimpleUML, rdbms:SimpleRDBMS) {
  top relation PackageToSchema { // map each package to a schema
    pn: String;
    checkonly domain uml p:Package {name=pn}; enforce domain rdbms s:Schema {name=pn};
  }
  top relation ClassToTable { // map each persistent class to a table
    cn, prefix: String;
    checkonly domain uml c:Class {namespace=p:Package {}, kind='Persistent', name=cn};
    enforce domain rdbms t:Table {schema=s:Schema {}, name=cn, column=cl:Column
        {name=cn+'_tid', type='NUMBER'}, key=k:Key {name=cn+'_pk', column=cl}};
    when { PackageToSchema(p, s); }
    where { prefix = ''; AttributeToColumn(c, t, prefix); }
  }
  relation AttributeToColumn {
    checkonly domain uml c:Class {}; enforce domain rdbms t:Table {};
    primitive domain prefix:String;
    where {
      PrimitiveAttributeToColumn(c, t, prefix); ComplexAttributeToColumn(c, t, prefix);
      SuperAttributeToColumn(c, t, prefix);
    }
  }
  function PrimitiveTypeToSqlType(primitiveTpe:String):String {
    if (primitiveType='INTEGER') then 'NUMBER'
    else if (primitiveType='BOOLEAN') then 'BOOLEAN' else 'VARCHAR' endif
    endif;
  }
  (...)
}
```

З іншого боку, метамодель QVTOperational (Operational Mapping) дозволяє специфікувати перетворення моделі імперативним способом, або шляхом визначення операційних перетворень (тобто перетворень, які виражаються імперативно), або шляхом доповнення реляційних перетворень імперативними операціями.

Операційне перетворення може бути лише односпрямованим і, як і реляційне перетворення, визначає сигнатуру, яка складається з типів її

моделей введення та виведення (наприклад UML, RDBMS). Окрім цього, між класами операційного трансформації та об’єктно-орієнтованого програмування (ООП) можна встановити різні подібності, а саме що кожне перетворення може бути створено як об’єкт. Крім того, завдяки своїй природі операційне перетворення надає виконувану точку входу під назвою main.

Перетворення складається, серед інших елементів, із набору операцій відображення. Операція відображення встановлює відображення між елементами вихідної та цільової моделей. Кожна операція відображення визначається:

- своїм підписом (вихідним і цільовим елементами);
- мовою where і умовою when (обидва необов’язкові);
- тілом операції.

Тіло, у свою чергу, складається з набору інструкцій (OCL у поєднанні з імперативними розширеннями) розділеного на початкові та кінцеві розділи. Розділ ініціалізації містить інструкції для запуску перед створенням екземплярів елементів цільової моделі (результатів операції), тоді як розділ популяції містить інструкції для налаштування та заповнення цих результатів, а розділ кінця містить інструкції для запуску перед завершенням операції. QVTOperational надає інструкції для таких цілей, як:

1. створення та заповнення об’єкта;
2. виклик помічників (операції, які отримують набір вихідних елементів і повертають результат);
3. використання проміжних класів і властивостей (перехідні дані, приєднані до цільових елементів);
4. виклик допоміжних перетворень.

Також можна використовувати деякі засоби, які зазвичай присутні в імперативних мовах, таких як C і C ++ . Одним із них є диз’юнкція, яка подібна до оператора switch і дозволяє вибирати, яку операцію відображення використовувати, залежно від їх захисників (типи елементів і речення when). Ще одна функція — це розширення типу , яке визначає нові типи, які діють

як шаблони для інших існуючих типів, у спосіб, дуже схожий на оператор typedef у C.

Нарешті, метамодель також надає набір змінних із коробки, а саме:

1. this, посилаючись на сам екземпляр перетворення;
2. self, який посилається на елемент вихідної моделі, який встановлює контекст для операції відображення;
3. result, який посилається або на результуючий елемент моделі, якщо відображення оголошує один результат або на кортеж результуючих елементів. Нульове значення також підтримується, як і в типових мовах ООП, щоб вказати відсутність значення.

Лістинг 2.2. Приклад операційного перетворення QVTOperational:

```
transformation Uml2Rdb(in srcModel:UML,out dest:RDBMS) {
  -- entry point: 1. tables created from classes, 2. tables updated with foreign keys implied by associations
  main() {
    srcModel.objects()[Class]->map class2table(); -- first pass
    srcModel.objects()[Association]->map asso2table(); -- second pass
  }
  -- maps a class to a table, with a column per flattened leaf attribute
  mapping Class::class2table () : Table when {self.kind='persistent';} {
    init { -- performs any needed initialization
      self.leafAttribs := self.attribute->map attr2LeafAttrs("","");
    }
    -- populate the table
    name := 't_' + self.name;
    column := self.leafAttributes->map leafAttr2OrdinaryColumn("");
    key_ := object Key { name := 'k_'+ self.name; column := result.column[kind='primary']; };
  }
  -- mapping to update a Table with new columns of foreign keys
  mapping Association::asso2table() : Table {
    init {result := self.destination.resolveone(Table);}
    foreignKey := self.map asso2ForeignKey(); column := result.foreignKey->column;
  }
  -- mapping to build the foreign keys
  mapping Association::asso2ForeignKey() : ForeignKey {
    name := 'f_' + self.name; refersTo := self.source.resolveone(Table).key_;
    column := self.source.leafAttribs[kind='primary']->map leafAttr2ForeignColumn(self.source.name+'_');
  }
  (...)
}
```

На відміну від Relations, метамодель QVTOperational надає лише текстовий конкретний синтаксис, уривок якого проілюстровано в лістингу 2.2 (будь-які символи між -- і кінцем рядка представляють коментарі). У цьому

прикладі показано інший спосіб представлення перетворення UML-реляційної моделі, який раніше було представлено в лістингу 2.1.

QVTCore (Core) створює невелику мову, орієнтовану на зіставлення шаблонів, наприклад Relations. Також декларативний за своєю природою, Core є таким же виразним, як і Relations, хоча він простіший (тобто визначає меншу кількість елементів моделювання). Незважаючи на те, що розробники QVT можуть вказати трансформацію моделі повністю за допомогою Core замість Relations, Core призначений для більш традиційного сценарію, у якому дизайнери вказують свої перетворення за допомогою Relations, а потім використовують перетворення RelationsToCore, показане на рисунку 2.2, щоб перетворити його на Core.

Оскільки Core визначає меншу кількість елементів, він є більш докладним, ніж Relations, оскільки вимагає більшої кількості екземплярів елемента моделювання для вираження того самого наміру, ніж останній. Однак, оскільки Core має бути таким же виразним, як Relations, можна зробити висновок, що Relations складається з синтаксичного цукру для Core.

Крім того, деякі з елементів, які неявно встановлюються у Relations, повинні бути явно визначені в Core, наприклад елементи відстеження (які встановлюють відповідність між елементами у вихідній і цільовій моделях).

Як і у зв’язках, базове перетворення складається з набору відображень між різними доменами (що відповідають моделям різних типів), які визначають шаблони, які повинні зберігатися у вихідній і цільовій моделях. Кожне перетворення можна запустити в режимі примусового виконання або в режимі перевірки. Під час роботи в режимі примусового виконання трансформація виконується в певному напрямку, від вихідної до цільової моделі. З іншого боку, у режимі перевірки перетворення лише перевіряє, чи виконуються обмеження в обох моделях.

Ядро розділяє шаблони кожного домену на захисні та нижні . Він також визначає додатковий набір шаблонів, які називаються середніми , які використовуються для встановлення мікроелементів між вихідною та

цільовою моделями (серед інших цілей) і можуть залежати від шаблонів з інших доменів, як показано на рисунку 2.4. Захисні шаблони служать тій самій меті, що й речення when Relations (тобто, щоб забезпечити подальші обмеження на те, чи може відбутися перетворення), у той час як нижні шаблони визначають структуру елементів моделі, які перетворення фактично перевірятиме або запроваджуватиме. Таким чином, перетворення моделі, визначене в нижніх шаблонах, відбудеться лише в тому випадку, якщо відповідні захисні шаблони успішно зіставлені.

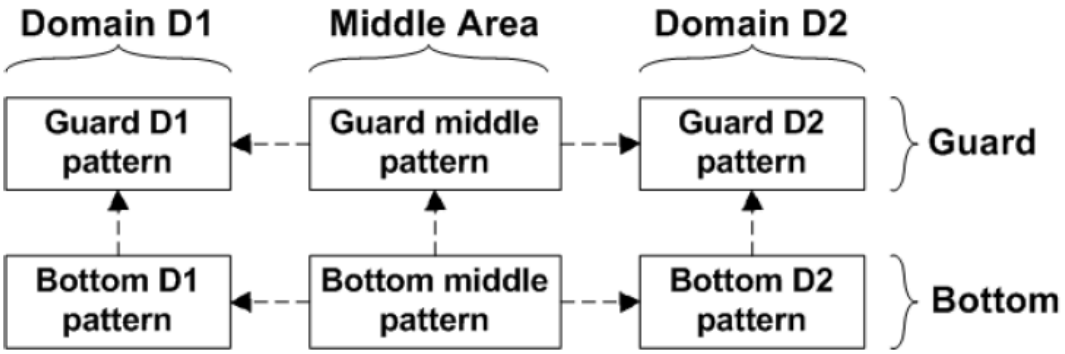


Рисунок 2.4 – Шаблони QVT Core і залежності між ними

Успішний збіг шаблону може встановити набір прив'язок. Прив’язка складається з набору унікальних значень – елементів моделі – які призначаються змінним в узгодженому шаблоні.

Шаблон складається з:

1. змінних, що ідентифікують елементи моделі, що представляють інтерес для успішних збігів шаблонів;
2. предикатів, що визначають перевірки, які необхідно зробити;
3. реалізовані змінні, які дозволяють модифікувати свої значення;
4. призначення, які приписують значення об'єктам;
5. операції чорної скриньки, які вносять зміни до примусових моделей і використовуються лише в нижніх шаблонах.

Останні три можуть мати побічні ефекти, тому вони дійсно ефективні лише тоді, коли перетворення виконується в режимі примусового виконання. У випадку реалізованих змінних побічні ефекти викликає не саме зв'язування, а скоріше можливість їх модифікації, а саме шляхом створення або видалення їхніх значень.

Лістинг 2.3. Приклад перетворення QVTCore:

```
-- A transformation definition from SimpleUML to SimpleRDBMS
module UmlRdbmsTransformation imports SimpleUML, SimpleRDBMS {
  transformation umlRdbms { uml imports SimpleUML; rdbms imports SimpleRDBMS; }
  -- Package and Schema mapping
  class PackageToSchema {
    composite classesToTables : Set(ClassToTable) opposites owner;
    composite primitivesToNames : Set(PrimitiveToName) opposites owner;
    name : String;
    umlPackage : Package; -- uml
    schema : Schema; -- rdbms
  }
  map packageToSchema in umlRdbms {
    uml () { p:Package } -- domain
    rdbms () { s:Schema } -- domain
    where () { -- empty middle guard pattern
      p2s:PackageToSchema| p2s.umlPackage = p; p2s.schema = s; -- middle bottom pattern
    } -- p2s is a realized variable
    map { -- composed mapping example
      where () { p2s.name := p.name; p2s.name := s.name; p.name := p2s.name; s.name := p2s.name; }
    }
  }
  map primitiveToName in umlRdbms {
    uml (p:Package) { prim:PrimitiveDataType| prim.owner = p; } -- prim is a variable
    check enforce rdbms () { sqlType:String } -- domain is checkable and enforceable
    where (p2s:PackageToSchema| p2s.umlPackage=p) {
      realize p2n:PrimitiveToName| p2n.owner := p2s; p2n.primitive := prim; p2n.typeName := sqlType;
    }
    map { where () { p2n.name := prim.name + '2' + sqlType; } }
  }
  (...)
} -- end of module
```

Призначення можуть бути за замовчуванням або не за замовчуванням. Призначення за замовчуванням відбувається (тобто оцінюється) лише під час застосування шаблонів і використовується для встановлення значення об'єкта під час застосування нижнього шаблону або цільового домену, або середнього. З іншого боку, призначення не за замовчуванням мають місце навіть тоді, коли виконуються лише перевірки шаблонів (і, отже, також

виконують роль предикатів), і їх можна використовувати для перевірки, чи забезпечує модель відповідність шаблону, який включає певний алгоритм або обчислення.

На відміну від Relations, Core надає лише текстовий конкретний синтаксис, який зображено в лістингу 2.3. Цей перелік містить фрагмент із базового перетворення, до якого ми додали деякі коментарі, щоб точно визначити використання деяких концепцій, які щойно були представлені. Як і приклади в лістингах 2.1 і 2.2 це перетворення отримує модель UML і перетворює її на реляційну модель.

У специфікації QVT також згадується використання Core як проміжної мови для віртуальної машини QVT у спосіб, дуже схожий на байт-код у віртуальній машині Java.

2.2 Підходи розробки web-додатків на основі моделей

Розробка web-додатків у поєднанні з новими перспективами щодо масштабування та розповсюдження додатків (таких як програмне забезпечення як послуга) серед якомога більшої кількості користувачів стала типовим способом створення нових додатків або оновлення старих. На даний момент існує кілька підходів до розробки веб-додатків, керованих моделлю. Однак підхід до розробки зазвичай керується виразністю мови моделювання, яка використовується в цьому підході. В даному розділі проаналізуємо наступні підходи MDE для розробки веб-додатків і мови їх підтримки:

- WebML;
- UWE;
- XIS2;
- Microsoft Sketchflow;
- Гнучка платформа OutSystems.

Підходи WebML, UWE та OutSystems Agile Platform були обрані для аналізу (замість інших можливих підходів MDE), оскільки вони добре відомі

у сфері розробки веб-додатків. XIS2 також розглядається в цьому аналізі, оскільки він був розроблений в ProjectIT-Studio, яка займалася розробкою додатків на основі моделі та генерацією коду. Хоча Microsoft Sketchflow не є підходом або мовою MDE як такою і він не орієнтований спеціально на веб-програми, ми включили його в цей аналіз, оскільки він може використовуватися для створення міжплатформних інтерактивних веб-програм і оскільки він адресує дуже важливий аспект, який не враховується в інших підходах: забезпечення зручного для зацікавлених сторін бачення дизайну програми.

У цьому розділі ми представляємо аналіз цих підходів до розробки веб-додатків, спочатку вводяться критерії аналізу, а саме шляхом опису аспектів, що розглядаються, а потім подаються результати цього аналізу.

Критерії аналізу. Цей аналіз здебільшого зосереджений на мові моделювання, яка використовується в підході, і на тому, чи він вирішує набір проблем моделювання, таких як моделювання домену, моделювання бізнес-логіки та моделювання інтерфейсу користувача. Тим не менш, ми також аналізуємо деякі аспекти, що стосуються генерації частин веб-додатку, наприклад використання перетворень від моделі до моделі або розглянуте середовище розгортання. Ці характеристики були адаптовані з [12], який визначає еталонну модель для MDE-орієнтованих мов веб-додатків, що є результатом даного аналізу.

А – Моделювання предметної області. Моделювання предметної області стосується ідентифікації концепцій проблемної області та їх представлення за допомогою мови моделювання (наприклад, діаграми UML). Цей аспект аналізується щодо наступних критеріїв:

1. чи підтримки мовою;
2. чи існує прихована залежність від постійності та деталей інтерфейсу користувача (тобто моделі домену не потрібно «налаштовувати» для підтримки цих рівнів).

B – Моделювання бізнес-логіки. Хоча визначення моделювання бізнес-логіки можна вважати дещо суб’єктивним, але ми розглядаємо його як специфікацію поведінки веб-додатку. Цей аспект аналізується з огляду на такі особливості:

1. чи підтримує він запити та маніпулювання концепціями домену (а саме використання шаблонів);
2. чи ця підтримка запитів і маніпуляцій є певним чином низькорівневою подібною до традиційного вихідного коду;
3. підтримка специфікації процесу.

Слід зазначити, що тема низькорівневої підтримки вважається актуальною, оскільки вона часто відображає експресивність мови: типові мови, орієнтовані на вихідний код (такі як C або Java), хоча й складні, проте дуже експресивні.

C – Моделювання навігації. Підтримка цього підходу для визначення потоку навігації (в контексті змодельованої веб-програми) між різними HTML-сторінками або навіть усередині HTML-сторінок також є аспектом, який аналізується.

D – Моделювання інтерфейсу користувача. Іншим важливим аспектом є підтримка підходу для моделювання інтерфейсу користувача (UI). Проаналізовані наступні теми:

1. чи є мова моделювання інтерфейсу користувача незалежною від платформи (тобто не вимагає спеціального програмного забезпечення для представлення інтерфейсу користувача);
2. чи підтримує специфікацію контролю доступу (тобто певні елементи керування відображаються або приховуються відповідно до автентифікованого користувача);
3. чи дозволяє визначати спеціальні елементи інтерфейсу;
4. можливість використовувати шаблони взаємодії (наприклад, створювати, редагувати або асоціювати та роз'єднувати);

5. підтримка процесів зв’язування між елементами інтерфейсу користувача та елементами моделі домену.

E – Перетворення від моделі до моделі. Цей аспект аналізує, чи використовує підхід перетворення від моделі до моделі. Цей вид перетворень зазвичай використовується для прискорення завдання проектування веб-додатку за допомогою аналізу моделі та механізмів висновку для автоматичного визначення деяких частин моделі веб-додатку, таким чином звільняючи розробника моделі від деяких повторюваних (і схильних до помилок) завдань.

F – Автоматична генерація коду. Цей аспект визначає, чи здатна інструментальна підтримка підходу повністю генерувати програму (тобто вона не потребує ручної реалізації певних функцій програмістами).

G – Незалежність від середовища розгортання. Цей аспект вивчає цільову платформу, яка розглядається підходом, а саме чи існує тісний зв’язок між підходом і цільовою платформою.

Результати аналізу. Аналіз цих підходів до розробки веб-додатків відповідно до критеріїв аналізу, наведених у попередньому підрозділі, дав результати, які представлені в таблиці 2.1.

A – Моделювання предметної області. Моделювання предметної області є важливим аспектом будь-якого підходу до розробки додатків. Проаналізовані підходи забезпечують незалежність між моделлю домену та іншими частинами дизайну програми, але лише WebML та XIS2 моделювання у спосіб, який повністю не залежить від решти деталей програми, не вимагаючи коригування моделі предметної області за допомогою деталей, орієнтованих на інтерфейс користувача. У випадку UWE модель вмісту має бути певною мірою орієнтована на модель презентації. У Agile Platform модель предметної області моделюється як схема бази даних, хоча з іншими термінами; хоча ця відсутність абстракції може розглядатися як недолік, на практиці вона надає дизайнеру більший рівень контролю над реальною схемою бази даних веб-додатку. WebML також дає можливість

налаштувати модель домену відповідно до конкретних потреб інших рівнів за допомогою моделі похідних даних, а XIS2 надає представлення BusinessEntities View для вирішення проблеми маніпулювання сутностями.

Таблиця 2.1

Характеристики проаналізованих підходів і мов MDE

	Web-ML	UWE	XIS2	Out-Systems	Sketch-flow
A. *Domain modeling*	✓	✓	✓	✓	✗
Independent from persistence	✓	✓	✓	✗	—
Independent from UI	✓	✗	✓	✓	—
B. *Business Logic modeling*	✓	✓	✓	✓	✓
Domain manipulation using patterns	✓	✗	✓	✓	✗
Custom patterns	✓	—	✓	✗	—
Low-level specifications	✓	✓	✗	✓	✓
Domain query	✓	✓	—	✓	✓
Domain manipulation	✓	✗	—	✓	✗
Process specification	✗	✓	—	✓	✗
C. *Navigation Flow modeling*	✓	✓	✓	✓	✓
D. *User Interface modeling*	✓	✓	✓	✓	✓
Access control specification	✓	✗	✓	✓	✗
Custom interface elements	✓	✗	✗	✓	✓
Interaction patterns	✓	✗	✓	✓	✓
Custom interaction patterns	✗	—	✗	✗	✓
UI elements bound to domain elements	✓	✓	✓	✓	✓
Bindings are customizable	✗	✗	✓	✓	✗
E. *Model-to-model transformations*	✗	✓	✓	✗	✗
F. *Generated application is complete*	✗	✗	✗	✓	✗
G. *Independent from deployment environment*	✓	✓	✓	✗	✓

B – Моделювання бізнес-логіки. Моделювання бізнес-логіки розглядається всіма проаналізованими підходами, хоча й дуже різними способами. XIS2 враховує цей аспект за допомогою Business-Entities дозволяючи іншим рівням маніпулювати елементами, які більш грубі, ніж елементи домену та шаблони (типові операції створення/читання/оновлення/ видалення підтримуються і дизайнер може додати більше операцій, але вони

повинні бути реалізовані вручну). WebML використовує схожі механізми, але розробник може оркеструвати їх у спосіб, подібний до діаграми, що дозволяє специфікувати робочі процеси маніпулювання даними. Однак UWE розглядає цей аспект лише за допомогою моделі процесу і діаграм класів, які представляють зв'язки між різними процесами і діаграми діяльності, які визначають етапи кожного процесу та обмеження OCL; сама реалізація повинна виконуватися вручну. Agile Platform дозволяє дизайнерам вказувати складну бізнес-логіку в графічній формі: шляхом визначення дій, таких як шаблони запитів до домену та маніпулювання, дизайнери можуть вказувати більшість функціональних можливостей, які зазвичай доводиться кодувати вручну.

C – Моделювання навігації. Моделювання стосується всіх проаналізованих підходів, оскільки воно є фундаментальним для будь-якого типу веб-додатків. Через природу моделювання веб-додатків і все, що з цього випливає (наприклад, існування HTML-сторінок, гіперпосилань для переходу між сторінками, використання параметрів запиту тощо), усі вони дотримуються однакових вказівок (хоча використані терміни дещо відрізняються): створюється напрямлений граф, де вузли відповідають сторінкам HTML, а ребра відповідають можливим потокам навігації, доступним для користувача. в контексті певної сторінки. Тим не менш, проаналізовані підходи відрізняються тим, чи може дизайнер вказати набори ребер (тобто дій або посилань), доступних у кожному вузлі. У WebML кожен вузол DataUnit має чітко визначений набір посилань для введення та вихід, і дизайнер не може вказати додаткові посилання. З іншого боку, XIS2 дозволяє дизайнеру вказувати дії, пов’язані з елементами інтерфейсу користувача на сторінці і потоки навігації згодом будуть пов’язані з цими діями на сторінці. Підхід XIS2 згідно цього аспекту схоже працює як в Agile Platform, Sketchflow і UWE.

D – Моделювання інтерфейсу користувача. За винятком WebML, усі проаналізовані підходи стосуються моделювання інтерфейсу користувача

графічним способом WYSIWYG. WebML дозволяє лише вказувати, які елементи будуть присутні на сторінці, але не дозволяє вказувати, де вони будуть розташовані. Також підтримується Agile Platform, WebML і Sketch flow створення нової сторінки або елементів інтерфейсу для повторного використання на екранах інших програм. Це дозволяє розробникам вказувати певні розділи екрана лише один раз та імпортувати ці розділи на всі екрани програми, з очевидною додатковою перевагою, що зміни потрібно робити тільки в одній точці моделі. Хорошим прикладом такого компонента може бути банер веб-сайту або дерево навігації веб-сайту.

F – Автоматична генерація коду. З проаналізованих підходів лише Agile Platform вважає, що вихідний код не слід редагувати вручну: редагувати можна лише саму модель, а згенерований код і бази даних завжди зберігаються поза досяжністю розробника. Усі інші підходи розглядають традиційне програмування (тобто ручне редагування згенерованих артефактів вихідного коду) як діяльність, яка має відбуватися протягом життєвого циклу розробки, щоб врахувати особливі вимоги, які можуть бути невиражені мовою моделювання підходу.

G – Незалежність від середовища розгортання. За винятком Agile Platform, усі підходи фактично не залежать від середовища розгортання. Завдяки використанню елементів високого рівня WebML, UWE та XIS2 можуть генерувати код для будь-якої веб-орієнтованої платформи (наприклад, ASP.NET, Apache Tomcat); XIS2 також може генерувати код для настільних платформ. Веб-орієнтована версія Sketchflow генерує програму Silverlight, яка, хоч і потребує плагіна Silverlight, встановленого у веб-браузері, може розглядатися як кросплатформна, оскільки вона доступна для платформ Microsoft Windows, Linux і Mac OS X.

Моделі, створені на платформі OutSystems Agile, можна розгортати лише в стеках розгортання OutSystems, які використовують або сервер прикладних програм JBoss і базу даних Oracle або сервер прикладних програм IIS і базу даних SQL Server Express від Microsoft. Хоча в цьому

аспекті Agile Platform, очевидно, більш обмежена, ніж інші підходи, саме це дозволяє їй автоматизувати більшу частину життєвого циклу розробки веб-додатків, а саме сценарії розгортання та оновлення або відкату додатків.

Хоча в даний час більшість веб-додатків все ще розробляються вручну (тобто за допомогою типових завдань програмування), концепція розробки веб-додатків на основі моделі швидко набуває популярності. Приклади цієї тенденції зростання можна знайти в ряді мов моделювання веб-додатків, таких як ті, що проаналізовані в цьому розділі.

Ми представили аналіз вибраного набору підходів і мов моделювання, орієнтованих на веб-додатки. Цей аналіз, у свою чергу, був зосереджений насамперед на аспектах, які мають відношення до такого роду мов моделювання. На рисунку 2.5 наведено простий огляд аналізованих аспектів у формі розумової карти.

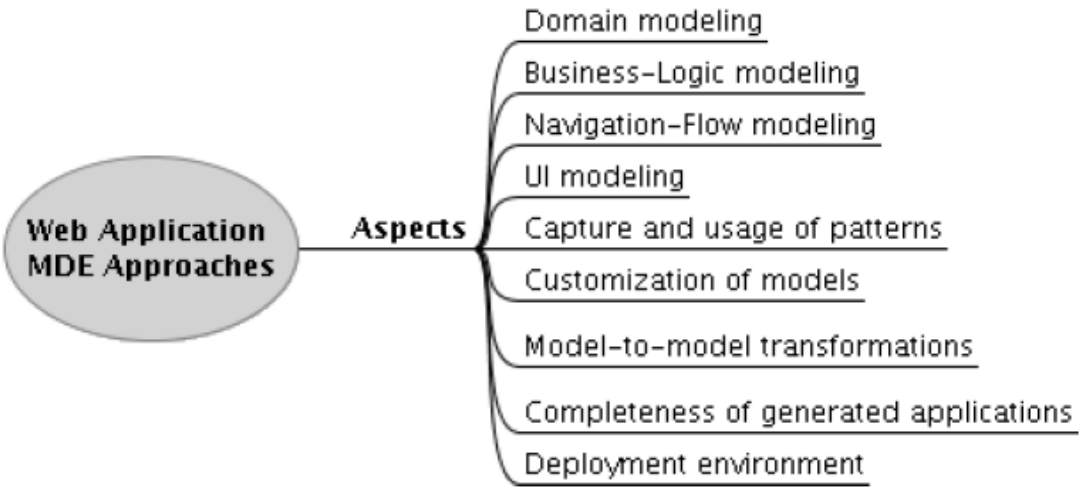


Рисунок 2.5 – Огляд аспектів і проблем щодо мов моделювання веб-додатків

У наступному підрозділі надамо аналіз систем управління контентом (CMS), ще однієї важливої теми для даної роботи. Окрім того, що ці системи надають своїм користувачам (не лише адміністраторам вмісту, а й споживачам) корисний набір функцій, таких як модульність або керування доступом, також мають деякі характеристики, які роблять їх придатними для

розробки веб-додатків, а саме розширювану архітектуру, за допомогою якої вони зазвичай створюються.

2.3 Порівняльний аналіз систем управління контентом (CMS) для розробки web-додатків з використанням моделі

Незважаючи на те, що ідея керування вмістом існує з самого початку Інтернету, лише в останні роки CMS стали досить популярними. Система керування вмістом (CMS) — це особливий вид веб-додатку, який орієнтований на керування та публікацію вмісту (який може бути майже чим різноманітним, наприклад допис у блозі, запис на форумі, текст HTML або відео). Ці системи зазвичай наділяють адміністраторів вмісту та споживачів (тобто звичайних користувачів, які просто переглядають веб-сайт) набором відповідних аспектів, таких як:

1. модульність;
2. незалежність між вмістом і його поданням;
3. керування доступом;
4. керування користувачами;
5. налаштовуваний зовнішній вигляд і макет вмісту.

Крім того, розробка веб-додатків на основі CMS – це тема, яка була запропонована досить недавно, оскільки раніше більшість систем CMS не надавали розробникам достатнього набору функцій.

У цьому розділі ми аналізуємо такі системи CMS, які ми вважаємо доречними для нашої дослідницької роботи: 1. DotNetNuke; 2. Drupal; 3. Joomla; 4. Vignette Content Management; 5. WebComfort.

Незважаючи на те, що кількість доступних систем CMS постійно зростає, вони були обрані для аналізу через значні відмінності між собою. Інші системи CMS (наприклад, WordPress дуже популярна система для ведення блогів, яка, тим не менш, вважається традиційною CMS) хоча так само підходять для аналізу, як і згадані раніше, дадуть дуже схожі результати

– наприклад, аналіз WordPress дасть майже ті самі значення, що й Joomla, що зробить їхнє включення в цей аналіз зайвим.

У цьому підрозділі представимо аналіз цих CMS. Як і в попередньому підрозділі 2.2, почнемо з представлення критеріїв аналізу, а потім переходимо до результатів, які були отримані в результаті цього аналізу.

Критерії аналізу. Цей аналіз здебільшого зосереджений на критеріях, що охоплюють різноманітні аспекти, починаючи від адміністративних можливостей (таких як конфігурація структури веб-сайту або підтримка кількох орендарів) до функцій, орієнтованих на розробника, наприклад розширюваність або надання спеціального підходу для розробки веб-програм на основі CMS.

A – Управління. Системи CMS зазвичай використовують один із двох підходів до керування: орієнтований на сторінку та орієнтований на контент. Перший підхід передбачає, що спочатку має бути визначена структура веб-сайту (тобто набір сторінок і компонентів), а потім визначається вміст у контексті цієї структури. З іншого боку, підхід, орієнтований на вміст, вимагає визначення самого вмісту, а потім адміністратор може вказати структуру сторінок, які відображатимуть частину цього вмісту. Цей аспект аналізує, який із цих підходів до управління використовується CMS.

B – Налаштовувана структура сайту. Цей аспект визначає, чи дозволяє CMS користувачам-адміністраторам таким чином налаштовувати структуру веб-сайту, що ефективно дозволяє відвідувачам сприймати організацію веб-сайту як структурований ієрархічний набір сторінок. Хоча цей аспект може здатися несуттєвим (оскільки він надає перевагу CMS із підходом до управління, орієнтованим на сторінку), насправді він використовується для вказівки чи підтримує CMS специфікацію структури веб-сайту, а ця структура, у свою чергу, часто є фундаментальною для допомоги відвідувачам під час навігації веб-сайтом і доступу до опублікованого вмісту.

C – Налаштовуваний візуальний макет сторінки. Окрім можливості налаштування структури веб-сайту, адміністратори також повинні мати

можливість налаштувати візуальний макет веб-сайту (тобто зовнішній вигляд веб-сайту, як-от використовувані кольори або відносне розташування кожного контейнера, який сторінки використовуватимуть для відображення зміст). Цей аспект визначає, чи підтримує CMS такий візуальний механізм.

D – Підтримка мультиорендування. Суть даної функції полягає в тому чи можливо для когось створити логічний набір веб-сайтів в межах однієї інсталяції CMS. Іншими словами, цей аспект визначає, чи може одна фізична інсталяція CMS підтримувати визначення кількох логічних веб-сайтів (наприклад, особистого веб-сайту та веб-сайту електронної комерції), кожен із яких зазвичай доступний через іншу URL-адресу.

E – Кілька видів зберігання контенту. CMS через свою динамічну природу повинна десь зберігати свою інформацію (наприклад, базу даних або навіть файлову систему). Хоча на перший погляд цей аспект може здатися лише деталлю, пов’язаною з технологією, важливо зазначити, що важливість цього аспекту випливає з роз’єднання логіки веб-програми та механізму збереження, що, у свою чергу, вводить уявлення про те, що моделювання предметної області мовою, орієнтованою на CMS, не повинно залежати від специфічних для технології деталей (наприклад, переглядів бази даних) або навіть припускати їх.

F – Може бути розширено третіми сторонами. Цей аспект ретельно перевіряє механізми, які надає CMS для її розширення третіми сторонами (наприклад, чи надає CMS API для розробки нових функцій). Цей аналіз особливо зосереджений на таких пунктах:

1. які мови програмування можна використовувати;
2. чи можна використовувати функції безпеки, надані CMS, щоб обмежити можливі дії;
3. чи дозволено змінювати типову поведінку системи CMS;
4. чи можливо додати до системи нову поведінку (наприклад, нові компоненти CMS або додатковий код для запуску, коли відбуваються певні події).

G – Підхід для розробки. Цей аспект аналізує тип підходу (якщо такий є), який CMS підтримує для розробки веб-програм на її основі (навіть якщо веб-програма складається лише з налаштувань, розширень або будь-чого, що змінює типову поведінку системи). Зокрема, ми визначаємо:

1) чи CMS розглядає будь-який конкретний підхід до розробки таких веб-додатків;

2) чи керується він моделлю, чи виконується більш традиційним способом, керованим вихідним кодом.

Результати аналізу. Аналіз цих систем CMS, згідно з попередньо перерахованими критеріями, привів до наступних результатів, представлених у таблиці 2.2. A – Управління. Що стосується підходу до управління, який використовується проаналізованими CMS, ми помітили, що використовуються обидва підходи. Зокрема, системи DotNetNuke і WebComfort використовують підхід, орієнтований на сторінку, за якого спочатку визначається структура веб-сайту (тобто набір вкладок і модулів), а потім у модулях визначається вміст.

Таблиця 2.2

Характеристики проаналізованих CMS

	DotNet-Nuke	Drupal	Joomla	Vignette	Web-Comfort
A. *Management approach*	Page--centric	Content--centric	Content--centric	Content--centric	Page--centric
B. *Customizable website structure*	✓	✓	✓	✓	✓
C. *Customizable visual layout of page*	✓	✓	✓	✓	✓
D. *Supports multi-tenancy*	✓	✓	✗	✓	✗
E. *Multiple kinds of persistence*	✗	✓	✗	✗	✓
F. *Can be extended by third-parties*	✓	✓	✓	✓	✓
Programming language(s)	C#/ VB.NET	PHP	PHP	Java	C#/ VB.NET
Provides security features	✓	✓	✓	✓	✓
Can change default behavior	✓	✗	✗	✗	✓
Can add new behavior	✓	✓	✓	✓	✓
G. *Development approach*	Tradi-tional	Tradi-tional	Tradi-tional	Tradi-tional	Tradi-tional

Тим не менше ці дві CMS підтримують спільне використання вмісту між модулями за допомогою функції копіювання модулів DotNetNuke і функцій копіювання модулів і посилань на модулі WebComfort. З іншого боку, системи Drupal і Joomla використовують підхід, орієнтований на вміст, за якого адміністратор спочатку визначає вміст, який відображатиметься користувачеві, а потім визначає структуру сторінок, які відображатимуть певні частини доступного вмісту. Саму систему Vignette можна розглядати як поєднання цих двох підходів, оскільки вміст визначається незалежно та представляється користувачеві за допомогою механізму шаблонів Vignette, який використовує попередньо визначену структуру веб-сайту для представлення існуючого вмісту. Тим не менш, ми вважаємо, що Vignette здебільшого орієнтована на вміст, оскільки вона робить більший акцент на визначенні вмісту, а не на визначенні структури веб-сайту.

B – Налаштовувана структура сайту. Усі проаналізовані системи CMS дозволяють адміністраторам налаштовувати структуру веб-сайту як ієрархічний набір сторінок або вузлів (у поєднанні з механізмами зв'язування, такими як меню Joomla) залежно від підходу до керування, який використовує CMS.

C – Налаштовуваний візуальний макет сторінки. Можливість налаштувати візуальний макет веб-сайту (тобто зовнішній вигляд веб-сайту, як-от використовувані кольори або відносне розташування кожного контейнера, який сторінки використовуватимуть для відображення вмісту) підтримується всіма аналізованими системами CMS.

D – Підтримка мультиорендування. У деяких із проаналізованих версій підтримується мультиоренда CMS системи. Якщо говорити точніше, лише Joomla та WebComfort не підтримують цю функцію, хоча в обох випадках це аспект, над яким зараз працюють розробники. Тим не менш CMS, які підтримують цю функцію, зазвичай роблять це шляхом визначення окремого префікса таблиці бази даних для кожного веб-сайту, що означає, що різні логічні веб-сайти часто повністю незалежні один від одного.

E – Кілька видів зберігання контенту. Щодо використовуваних механізмів збереження, лише деякі системи CMS (Drupal і WebComfort) підтримують кілька видів механізмів збереження. Зокрема, ця функція має форму підтримки різних типів СУБД таких як MySQL, PostgreSQL або Microsoft SQL Server.

F – Може бути розширено третіми сторонами. Усі ці системи CMS допускають розширення сторонніми розробниками. Аспекти, які можуть використовуватися або розширюватися розробниками, різняться між системами, хоча деякі з них є загальними, наприклад функції безпеки. Однак, як і у випадку з підтримуваними механізмами збереження, специфічні для технології деталі (наприклад, використовувані мови програмування) різняться в системах CMS, що ще більше свідчить про те, що мови, орієнтовані на CMS, повинні бути максимально незалежними від технологій. Крім того, усі ці системи підтримують додавання функцій і поведінки, але лише деякі підтримують зміну існуючої поведінки за замовчуванням. В останньому випадку це зазвичай робиться за допомогою шаблону розробки стратегії і використання вбудованої поведінки за замовчуванням, лише якщо іншої стратегії немає.

G – Підхід до розробки. Жодна з проаналізованих систем CMS не розглядає підхід до налаштування або розробки розширень. Хоча, як згадувалося вище, кожна з цих систем надає певну підтримку розробнику – у різних формах, але зазвичай складається з API для розробки вихідного коду, а сам підхід до розробки залишається визначати окремо.

Використання систем CMS як базової платформи для визначення нових веб-сайтів швидко зростає у популярності. Приклади цього буму можна знайти в кількості доступних систем CMS і зростаючій кількості підтримуваних ними функцій.

У цьому підрозділі ми проаналізували деякі системи CMS, відповідно до аспектів, які зазвичай мають значення під час розробки розширень CMS, налаштувань або навіть веб-додатків на основі CMS із значним ступенем

складності. На рисунку 2.6 наведено огляд проаналізованих аспектів у формі розумової карти.

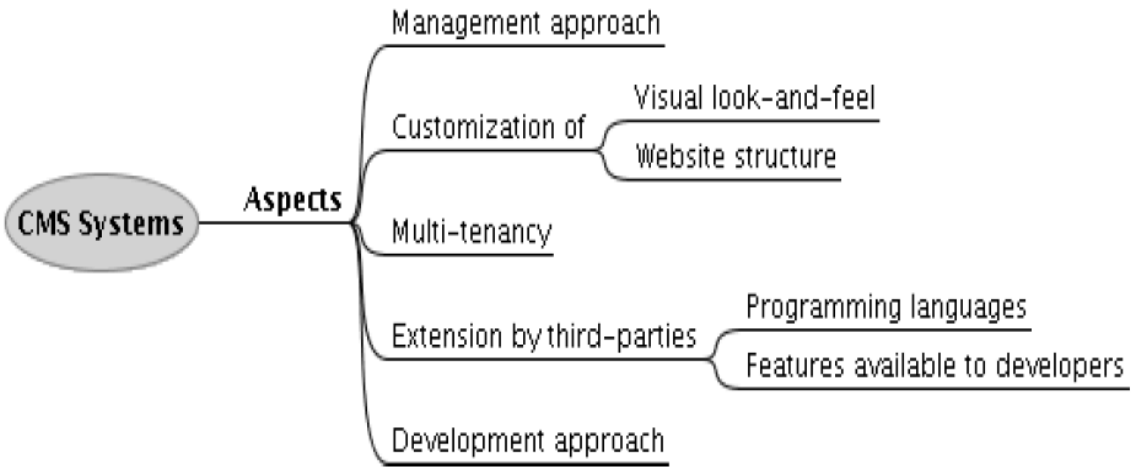


Рисунок 2.6 – Огляд аспектів і проблем щодо використання CMS

Висновки до розділу 2

В даному розділі представлено процеси моделювання для ефективної розробки web-контенту та додатків. Подано контекст для використання перетворень моделі в MDE та деякі мови перетворення моделі, які мають особливе значення, зокрема було представлено QVT і OMG. Також описані підходи до розробки веб-додатків, орієнтовані на моделі і виконано порівняльний аналіз підходів і мов моделювання, орієнтованих на веб-додатки. Проведено порівняння CMS і визначено деякі аспекти, які мови, орієнтовані на CMS, як базису для створення веб-додатків.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МОДЕЛІ CMS-РІШЕННЯ ДЛЯ МОДЕЛЮВАННЯ КОНТЕНТУ В WEB-РІШЕННЯХ ТА СЕРВІСАХ

3.1 Підхід розробки web-додатків з використанням CMS та концепції Model-Driven Engineering

Як було зазначено в попередніх розділах, розробка веб-додатків, які підтримуються платформами CMS, зазвичай виконується з використанням традиційних процесів розробки програмного забезпечення, у яких вихідний код є основним артефактом, а моделі дизайну та документація вважаються артефактами підтримки. Такі процеси займають багато часу та можуть викликати помилки, оскільки вони значною мірою залежать від програмістів та виконання ними типово повторюваних завдань. З іншого боку, підходи до розробки MDE мають на меті залишити більшість цих повторюваних завдань автоматизованим перетворенням моделі.

Хоча наразі існують деякі підходи MDE для розробки веб-додатків, такого підходу немає для веб-додатків на основі CMS. Це зрозуміло, оскільки ідея використання систем CMS як платформ для більш складних веб-додатків є відносно новою, але факт залишається фактом: ці системи забезпечують певний ступінь розширюваності, який варто було б використати.

Крім того, більшість підходів MDE до розробки веб-додатків не забезпечують належної підтримки для різних точок зору, які зацікавлені сторони програми мають щодо цієї програми. Ці перспективи часто фіксуються вручну (за допомогою таких засобів, як документи з вимогами та зустрічі із зацікавленими сторонами), вручну інтерпретується групою розробників програми (включно з системними архітекторами та експертами домену), а потім надається програмістам програми для впровадження в конкретну систему. Однак більша частина цієї обробки виконується вручну через залучення різних зацікавлених сторін. Зазвичай вважається, що мати

справу з усіма цими різними точками зору легше, використовуючи природну мову (одну з найбільш експресивних мов, які ми маємо, хоча іноді й неоднозначну), яка, у свою чергу, є мовою, до якої люди все ще набагато більші. вправніше, ніж комп'ютери.

Інша проблема в сучасних підходах MDE полягає в тому, що вони включають занадто багато деталей низького рівня в мову моделювання, намагаючись зробити її більш виразною або намагаються включити в мову якомога менше деталей низького рівня, щоб полегшити її вивчення та використання. Хоча це здається очевидним (оскільки було б дуже важко, якщо не неможливо, визначити мову моделювання, яка була б достатньо виразною для потреб кожної зацікавленої сторони), насправді це розкриває наслідки компромісу, який в кінцевому підсумку має бути прийнятий через проблема, згадана в попередньому абзаці.

Нарешті, кінцевою метою більшості сучасних підходів MDE все ще є отримання вихідного коду, а не моделей. Це часто призводить до того, що розробники змінюючи згенерований вихідний код саму модель залишають незмінною, а отже, несинхронізованою з вихідним кодом. Хоча існують методи, які дозволяють вручну змінювати вихідний код, зберігаючи його синхронізацію з моделлю: розробникам зазвичай потрібно вручну редагувати вихідний код, коли мова моделювання недостатньо виразна.

Для вирішення виявлених проблем ми пропонуємо підхід, орієнтований на MDE, для розробки веб-додатків на основі CMS. Цей підхід демонструє деякі відмінності від інших підходів, орієнтованих на MDE, для розробки веб-додатків, а саме:

1. він базується на використанні кількох мов моделювання;
2. він використовує механізм синхронізації моделі щоб забезпечити узгодженість між моделями різних мов і дозволити зацікавленим сторонам одночасно змінювати різні типи моделей (які відповідають різним точкам зору бажаної веб-програми) без можливої втрати інформації. На рисунку 3.1 наведено спрощений огляд запропонованого підходу.

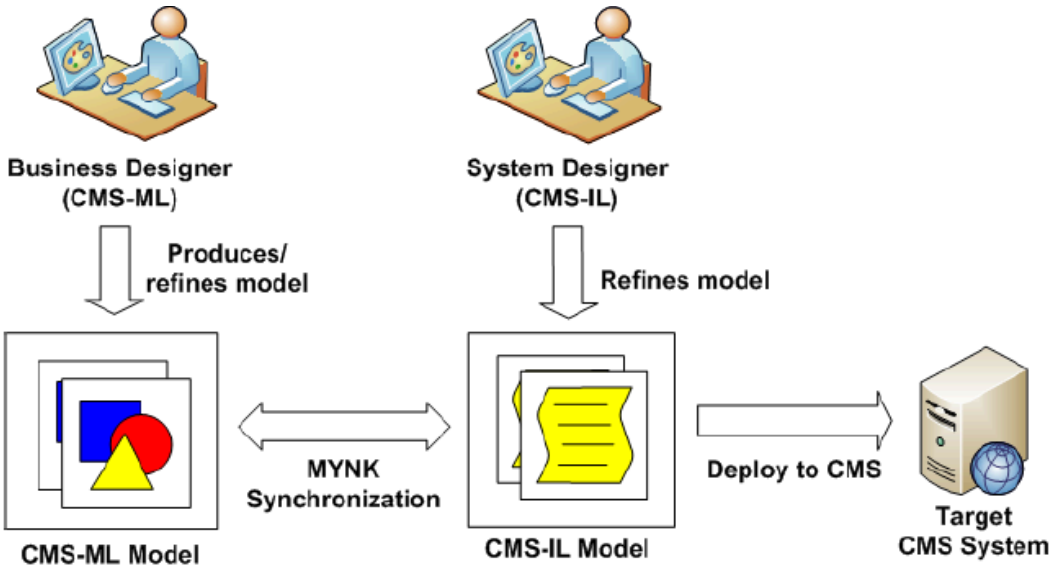


Рисунок 3.1 – Огляд запропонованого підходу до розробки, орієнтованого на концепцію MDE

Замість визначення єдиної мови моделювання, орієнтованої на CMS, цей підхід визначає дві такі мови:

1. CMS-IL (проміжна мова CMS), мова низького рівня, яка забезпечує спільну основу для специфікації веб-додатків на основі CMS;
2. CMS-ML (мова моделювання CMS), яка надає набір елементів, які можна використовувати для визначення моделі високого рівня веб-додатку.

Окрім мов CMS-ML та CMS-IL, підхід також враховує мову MYNK (Model SYNchronization frameworkK), яка підтримує механізм синхронізації моделі.

Заздалегідь слід зазначити, що мови CMS-IL і CMS-ML недостатньо для підтримки всіх зацікавлених сторін, оскільки така мета не була б практичною. В даній роботі ми зосереджуємось на дуже обмеженому наборі типів зацікавлених сторін, які, є достатньо репрезентативними для зацікавлених сторін, які зазвичай беруть участь у такого роду проектах розробки програмного забезпечення. Через це можливе обмеження щодо типів зацікавлених сторін, які розглядаються, ми також пропонуємо короткий набір інструкцій для створення нових мов моделювання, які можна

використовувати з цим механізмом синхронізації моделі. Ці рекомендації, описані в наступному підрозділі, були використані у визначенні двох мов і можуть бути використані у визначенні нових мов моделювання для задоволення потреб додаткових типів зацікавлених сторін.

Мова CMS-ML. CMS-ML — це мова графічного моделювання, яка розробляє специфікації веб-додатків на основі CMS на високому рівні та незалежно від платформи.

Вона розроблена, щоб дати можливість нетехнічним зацікавленим сторонам швидко змоделювати веб-програму, що підтримується CMS, а саме, дозволяючи таким зацікавленим сторонам легко читати, розуміти та змінювати модель, щоб вона точніше відображала їхні наміри. Крім того, CMS-ML також має механізм розширення, спрямований на підтримку конкретних вимог зацікавлених сторін і моделювання веб-додатків з вищим ступенем складності.

Цільовою аудиторією CMS-ML є, як уже згадувалося, нетехнічні зацікавлені сторони, які будуть моделювати, якою має бути веб-програма. Однак механізм розширюваності мови містить деякі елементи моделювання, які більше орієнтовані на те, як веб-додаток має виконувати свою роль. Ці елементи можуть вимагати від зацікавлених сторін певного технічного розуміння того, як має функціонувати веб-додаток; тим не менш, використання таких елементів є абсолютно необов'язковим.

Мова CMS-IL. CMS-IL — це текстова мова низького рівня з вищим рівнем виразності, ніж CMS-ML, яка, незважаючи на це не залежить від будь-якої конкретної платформи CMS. Ця мова надає елементи структурного моделювання, але, на відміну від CMS-ML, вона також приділяє більше уваги тому, як має працювати веб-програма. Таким чином, він визначає набір елементів моделювання, схожих на мову програмування, які дозволяють специфікувати веб-додаток на основі CMS на рівні абстракції, близькому до реалізації.

Кінцева мета CMS-IL полягає в тому, щоб забезпечити спільну основу для специфікації веб-додатків на основі CMS, на відміну від CMS-ML, яка має на меті забезпечити спосіб створення моделей веб-додатків простим способом. Цільовою аудиторією CMS-IL є технічні зацікавлені сторони, а саме розробники.

Мова MYNK. MYNK — це текстова мова, яка підтримує механізм синхронізації моделей між моделями CMS-ML і CMS-IL. Цей механізм, у свою чергу, є основою запропонованого підходу. Це пояснюється тим, що бажане збільшення продуктивності та менша кількість помилок моделювання залежать від використання автоматизованої синхронізації моделей або перетворень моделі; альтернативою було б ручне перетворення моделей CMS-ML на CMS-IL (і навпаки), що знизило б продуктивність і збільшило кількість помилок моделювання, зроблених дизайнерами, через схильність до помилок характеру цього завдання.

Цю мову можна розглядати як модельно-орієнтовану мову запитів (хоча такий опис не зовсім коректний). Він частково базується на проаналізованих механізмах трансформації моделі та мові SQL.

З першого він успадковує концепцію встановлення відповідностей між розглянутими мовами моделювання, тоді як другий є джерелом для створення запитів моделі, які можуть просто отримати елементи моделі або фактично змінити їх.

На відміну від CMS-ML і CMS-IL, мова MYNK не призначена для використання зацікавленими сторонами, які беруть участь у розробці веб-додатків. Це тому, що цим зацікавленим сторонам зазвичай не потрібно знати, як виконується синхронізація моделі між CMS-ML і CMS-IL. Натомість ця мова призначена для розробників мов, які потім можуть використовувати MYNK для розширення цього підходу за допомогою додаткових мов моделювання (орієнтованих на CMS чи інших). Можливою причиною для визначення нових мов моделювання може бути підтримка додаткових типів зацікавлених сторін.

3.2 Представлення робочого процесу розробки та метамоделі Revised Metamodel for Multiple Metalevels (ReMMM)

Запропонований підхід також враховує робочий процес, який ми вважаємо найбільш часто використовуваним. Цей робочий процес, який уже було показано на рисунку 3.1, розглядає два основних типи зацікавлених сторін, які ми позначаємо як Business Designer і System Designer.

Business Designer (дизайнер) – загальний термін для ідентифікації нетехнічних зацікавлених сторін, що можуть розпочати робочий процес, створивши модель CMS-ML, яка представляє заплановану веб-програму відповідно до деяких заздалегідь визначених бізнес-вимог.

Після використання MYNK для отримання моделі CMS-IL (що відповідає вищезазначеній моделі CMS-ML), System Designer (системний дизайнер) який, на відміну від бізнес-дизайнера, є терміном для ідентифікації технічних зацікавлених сторін – повинен визначити чи ця модель CMS-IL є задовільним, а саме шляхом визначення будь-яких конкретних вимог які не можуть бути вирішені лише за допомогою CMS-ML. Якщо такі вимоги існують то розробник системи має додатково модифікувати отриману модель CMS-IL для їх вирішення. Поки розробник системи змінює модель CMS-IL, механізм синхронізації MYNK повинен підтримувати узгодженість моделей CMS-IL і CMS-ML (які відповідають тій же цільовій веб-програмі, що розглядається з точки зору різних зацікавлених сторін).

Після процесу вдосконалення, модель CMS-IL має бути точним відображенням того, якою має бути запланована веб-програма. Однак, якщо бізнес-дизайнер не погоджується з моделлю CMS-ML (яку тим часом було автоматично оновлено через MYNK, щоб відобразити зміни, внесені системним дизайнером у відповідну модель CMS-IL), тоді бізнес-дизайнеру буде потрібно змінити модель CMS-ML, знову запускаючи інший процес механізму синхронізації MYNK.

Коли модель CMS-IL і відповідна модель CMS-ML вважається задовільною, її розгортають у цільовій системі CMS одним із двох способів, залежно від самої CMS:

- розгортання до компонент CMS Model Interpreter (або просто Interpreter), який уже доступний у системі CMS;
- створення низькорівневих артефактів і подальше встановлення.

Компонент інтерпретатора моделі CMS – це компонент, який інстальовано в системі CMS і відповідає за отримання вхідної моделі CMS-IL і розгортання отриманої моделі шляхом налаштування системи CMS для відображення фактів виражених в моделі. Цей компонент Інтерпретатора який, хоча і не зосереджений на системах CMS або розробці веб-додатків, визначає середовище виконання, кероване моделлю

MDR середовище, яке отримує модель (зазначену за допомогою діаграм класу та активності UML, анотованих обмеженнями OCL) та інтерпретує її під час виконання, фактично дозволяючи дизайнерам запускати свої моделі.

Ми вважаємо, що перша альтернатива розгортання – використання компонента інтерпретатора моделі CMS зазвичай краща, оскільки для цього потрібно лише, щоб адміністратор CMS завантажив модель CMS-IL в інтерпретатор. Тим не менш, ця перша альтернатива не буде здійсненною на платформах CMS, які не мають такого компонента; у таких випадках другий варіант (не показаний на рисунку 3.1) вимагає втручання розробника програмного забезпечення щоб скомпілювати згенеровані артефакти у форму, яка може бути виконана CMS і адміністратором CMS, щоб обидва розгорнути скомпільовані артефакти та внести необхідні зміни в конфігурацію.

Хоча цей підхід не усуває потреби в ітераційному процесі розробки, ми вважаємо, що він потенційно зменшує додаткову роботу з обробки невідповідностей між точками зору зацікавлених сторін і їх розумінням розробником.

Нарешті, слід зазначити, що ми не виключаємо можливого сценарію, за яким бізнес-дизайнер визначає модель CMS-ML, а потім розгортає її на

компоненті CMS Model Interpreter цільової системи CMS, оминаючи уточнення моделі системним дизайнером (наприклад, для швидкого отримання прототипу бажаної веб-програми). Однак це можна легко вирішити, дозволивши вищезгаданому компоненту Інтерпретатора отримувати вхідну модель CMS-ML, внутрішньо отримувати відповідну модель CMS-IL і розгортати отриману модель CMS-ML.

Метамодель ReMMM. Метамодель MoMM представляє набір характеристик, які вирішують проблему визначення мов моделювання, враховуючи кілька метарівнів (окрім класичного поєднання метарівнів клас-примірник). Однак ця метамодель також має деякі недоліки, з яких ми виділяємо наступні:

- MoMM не має зв'язків між елементами різних метарівнів як ModelElements, хоча існує онтологічний механізм інстанціювання, який передбачається асоціацією класу ModelElement, для дизайнера моделі неможливо посилатися на певний екземпляр посилання без посилання на відповідний екземпляр ModelElement;
- Дуже просто визначити корисні метамоделі (наприклад, на відміну від UML, MoMM не визначає концепцію типу даних які, у свою чергу, необхідні при визначенні можливих обмежень значення для атрибутів). Тим не менше, слід зазначити, що розглядати це як недолік не можна, оскільки мета MoMM полягає в тому, щоб бути найпростішою метамоделлю, необхідною для визначення нових метамоделей з кількома метарівнями.

Ці недоліки, у свою чергу, були мотивацією для визначення метамоделі ReMMM (метамодель для кількох метарівнів), варіанту MoMM, спрямованого на вирішення цих проблем.

Незважаючи на те, що ReMMM є варіантом MoMM, ReMMM також можна вважати визначеним за допомогою MoMM, оскільки останній є рефлексивним, а тому може використовуватися для визначення не лише самого себе, але й розширень до себе. Абстрактний синтаксис ReMMM представлено на рисунку 3.2.

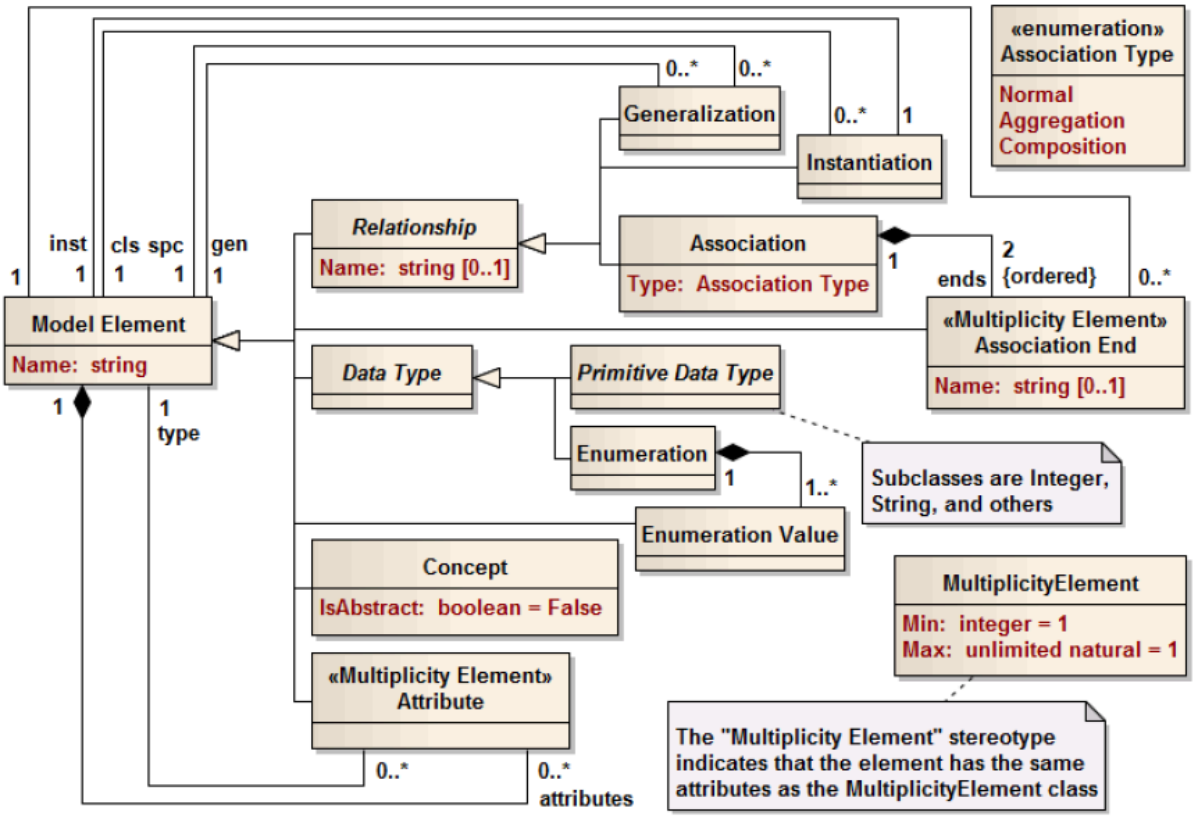


Рисунок 3.2 – Метамодель ReMMM

ReMMM є схожим до MoMM стосовно того, що модель складається лише з набору елементів моделі, які пов’язані між собою. Насправді ReMMM і MoMM схожі один на одного, оскільки перший здебільшого складається з розширень другого. Найбільш помітними відмінностями між ними є наступні:

- відношення екземплярів представлено екземпляром, зв’язком, який встановлює онтологічні зв’язки екземплярів між двома елементами моделі;
- явне визначення типу даних, яке представляє своєрідне значення, яке може приймати атрибут. Елемент Data Type є абстрактним і спеціалізується на простому типу даних;
- одна спеціалізація типу даних, Enumeration, містить значення Enumeration і представляє набори можливих значень, які можуть приймати

атрибути (подібно до перерахувань в UML або типових мовах програмування);

- примітивні типи даних використовуються для представлення «простих» типів значень (таких як рядок або ціле число). Можна припустити, що вони присутні на всіх метарівнях (через рефлексивність ReMMM), щоб атрибут був пов’язаний із цим типом даних без того, щоб цей зв’язок перетинав будь-які межі мета рівня, що порушило б вимоги метамоделювання;
- зв'язок асоціації, який використовувався для того, щоб дозволити дизайнерам визначати власні типи зв'язків між елементами моделі, тепер може призначати імена своїм кінцям асоціації, щоб ідентифікувати роль кожного елемента моделі в цій асоціації.

Що стосується зв’язку атрибута з елементом моделі (його типом), ми вирішили залишити цей зв’язок без змін (на відміну від зв’язування атрибута з типом даних, що має місце в UML), щоб не зменшувати виразність цієї метамоделі (наприклад, , запобігаючи сценаріям, у яких атрибут посилається на конкретний екземпляр елемента моделі). Тим не менш, ми не використовуємо цю функцію в метамоделях на основі ReMMM, тому можна використовувати конкретний синтаксис UML, що полегшує інтерпретацію цих метамоделей. ReMMM також має властивість MoMM бути рефлексивним (тобто ReMMM може використовуватися для опису самого себе). Для цього метамоделі для мов CMS-ML і CMS-IL були визначені з використанням ReMMM як метаметамоделі. Точніше, ми використовуємо переваги рефлексивності ReMMM і явного визначення концепції Instantiation щоб визначити ці мови за допомогою кількох метарівнів. Мова синхронізації моделей MYNK також припускає, що метамоделі моделей, які синхронізуються, можуть бути самі визначені за допомогою ReMMM тому, що MYNK потребує «спільної основи», яка дозволяє йому отримувати та маніпулювати багатьма видами моделей.

3.3 Використання мови CMS-ML для розробки веб-додатків

Моделювання за допомогою CMS-ML в основному зосереджено на трьох різних і взаємодоповнюючих типах моделей:

1. Шаблони веб-сайтів;
2. Анотації веб-сайтів;
3. Наборів інструментів.

На рисунку 3.3 показано, як ці моделі пов’язані одна з одною: і шаблони веб-сайтів і набори інструментів можуть посилатися на інші набори інструментів, але анотації веб-сайтів можуть лише прикрашати шаблони веб-сайтів.

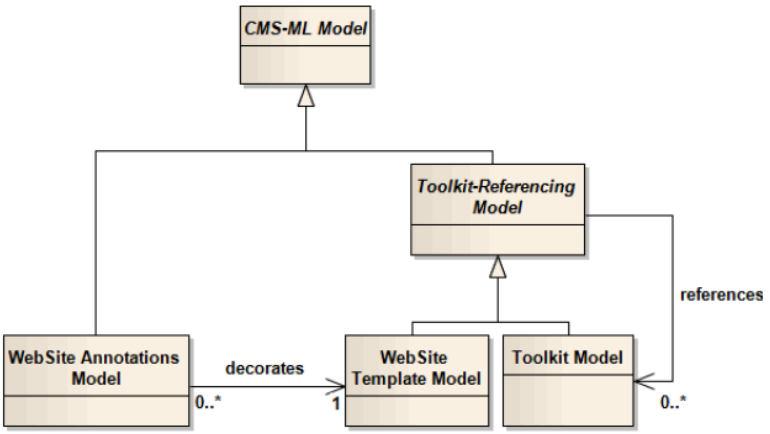


Рисунок 3.3 – Взаємозв’язок між різними моделями CMS-ML

Шаблон веб-сайту (шаблоном) – це модель, яка відображає передбачувану структуру веб-додатку. Цей шаблон моделюється за допомогою елементи CMS, такі як роль, динамічна веб-сторінка та веб-компонент , надані CMS-ML.

З іншого боку, набір інструментів дозволяє визначати нові орієнтовані на CMS елементи моделювання, а саме шляхом визначення моделі домену, інтерфейсу користувача та відповідної поведінки. Як згадувалося раніше, шаблон веб-сайту може посилатися на інструментарій (або набір Toolkits), що, у свою чергу, робить елементи Toolkit доступними для використання в

шаблоні. Крім того, набір інструментів також може посилатися на інші набори інструментів, створюючи можливість сценаріїв, у яких набір інструментів А вдосконалює та розширює функціональні можливості, які раніше були визначені в іншому наборі інструментів Б.

Елементи шаблону веб-сайту можна анотувати за допомогою моделі анотацій веб-сайту (анотацій). Ця модель доповнює шаблон веб-сайту, дозволяючи розробникам шаблону вказувати специфічні властивості CMS (наприклад, налаштування конфігурації) не загромаджуючи сам шаблон деталями, що стосуються конкретної платформи. Таким чином, з практичної точки зору, розробники шаблонів веб-сайтів CMS-ML розглядають не дві різні моделі – шаблон і анотації, а одну модель, яка є результатом поєднання цих двох моделей (тобто модель, яка фактично є результатом комбінації шаблону з анотаціями).

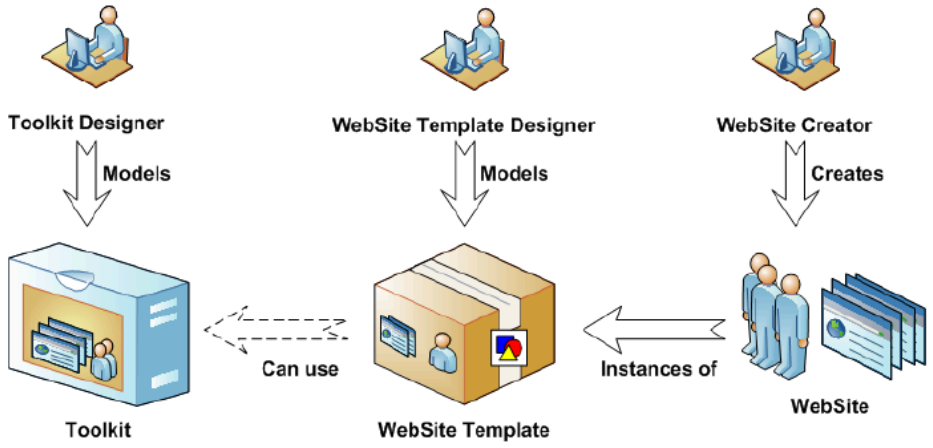


Рисунок 3.4 – Ролі та артефакти моделювання, розглянуті CMS-ML

Не обов’язково, щоб один дизайнер CMS-ML мав навички для створення обох об’єктів: шаблону веб-сайту та моделі інструментарію. Натомість ми вважаємо, що розробка CMS-ML часто виконуватиметься відповідно до наступних ролей моделювання, як показано на рисунку 3.4:

- Toolkit Designer, який моделює Toolkit;

- WebSite Template Designer який моделює шаблон веб-сайту і за необхідності анотує його за допомогою моделі анотацій веб-сайту;
- WebSite Creator який створює різні елементи, визначені в шаблоні веб-сайту.

3.4 Реалізація метарівнів та архітектури концепції CMS-ML

Шаблони веб-сайтів призначені для створення абстракцій (тобто моделей) конкретних веб-додатків за допомогою елементів, орієнтованих на CMS, а набори інструментів використовують загальні елементи моделювання для створення нових елементів моделювання, орієнтованих на CMS. Оскільки деякі концепції набору інструментів також є спеціалізаціями концепцій шаблону веб-сайту і тому екземпляри цих концепцій набору інструментів автоматично вважаються екземплярами відповідних концепцій шаблону веб-сайту, то розробники шаблонів можуть потім використовувати ці концепції набору інструментів для створення шаблонів веб-сайту так само, як і під час використання попередньо визначених елементів моделювання шаблону.

На рисунку 3.5 зображено метарівні, які розглядає CMS-ML:

- Metalevel ML3 містить модель Toolkit Modeling, яка забезпечує визначення загальних елементів моделювання Toolkit, які використовуватимуться для визначення моделей Toolkit. Цей метарівень не можуть змінити будь-які дизайнери;
- `Metalevel ML2 містить моделі моделювання шаблону веб-сайту та моделювання анотацій веб-сайту, які надають елементи моделювання, що використовуватимуться для визначення моделей шаблону веб-сайту та анотацій веб-сайту. Крім того, дизайнери інструментарію можуть створювати екземпляри загальних елементів моделювання, розташованих у ML3, щоб визначати нові елементи, які спеціалізуються на елементах моделювання шаблонів веб-сайтів. Однак, як і моделі моделювання набору інструментів у

ML3, моделі моделювання шаблону веб-сайту та моделювання анотацій веб-сайту є фіксованими та не можуть бути змінені;

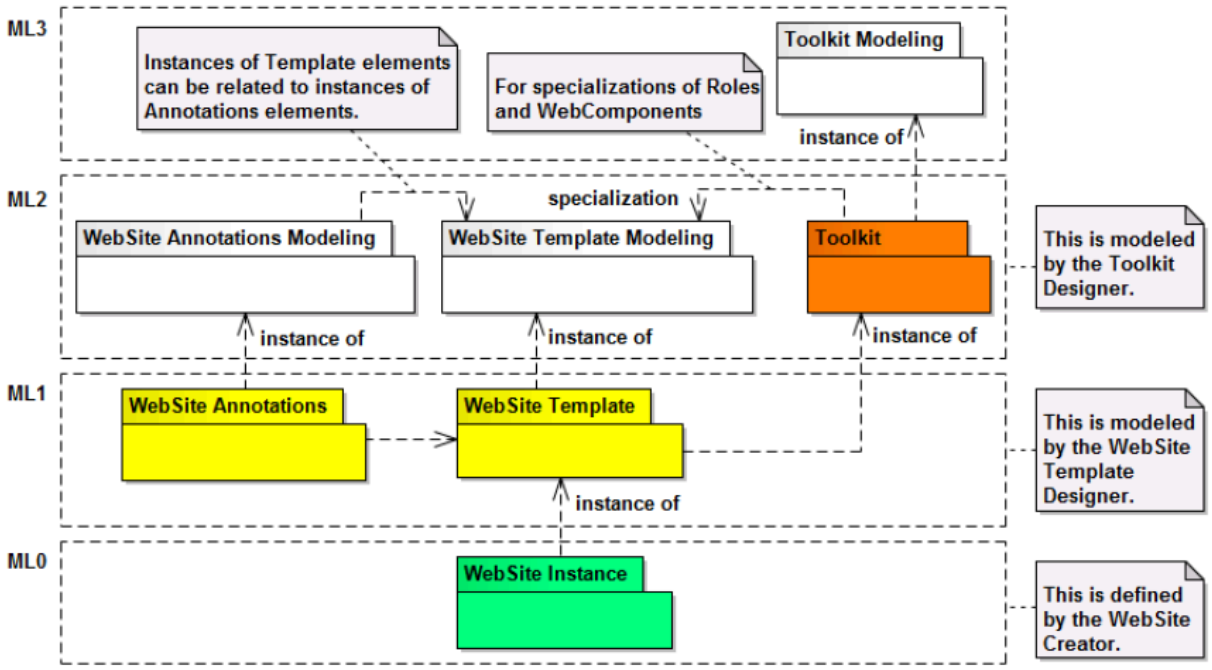


Рисунок 3.5 – Метарівні, що використовуються CMS-ML

- На Metalevel ML1 розробники шаблонів веб-сайтів можуть створювати моделі шаблонів веб-сайтів і анотацій за допомогою елементів моделювання, визначених у ML2. Ці елементи включають не лише ті, що надаються моделями моделювання шаблону веб-сайту та моделювання анотацій веб-сайту, а й елементи, визначені будь-якою моделлю інструментарію, які стають доступними для шаблону веб-сайту через концепцію імпорту інструментарію;
- На Metalevel ML0 розробник веб-сайту, а не дизайнер шаблонів використовує елементи, визначені в моделі шаблону веб-сайту разом із його оформленням WebSite Annotations для налаштування конкретної інсталяції CMS. Зазвичай для цього потрібен певний специфічний для CMS механізм, який встановлює відображення між примірником і елементом моделі (наприклад, стовпець у таблиці бази даних «Користувачі» який для кожного рядка/користувача CMS ідентифікує відповідного користувача шаблону).

Треба зауважити, що існує певна схожість між метарівнями ML1 і ML0 і метарівнем M1, які містяться в специфікації OMG для UML, оскільки їхня мета практично однакова. Основна відмінність полягає в тому, що якщо в UML екземпляри класу та об’єкта розташовані на одному метарівні M1, то в CMS-ML вони розташовані на метарівнях ML1 та ML0 відповідно.

Обґрунтування цієї архітектури метарівня полягало в тому, щоб:

1. вирішити проблему розширення мови простим, але зрозумілим способом;
2. зменшити випадкову складність, яка зазвичай виникає внаслідок використання шаблонів моделювання, схожих на тип-екземпляр на тому самому метарівні;
3. дотримуватися доктрини метамоделювання згідно з якою не повинно бути випадків зв’язків, що перетинають більше однієї межі метарівня.

3.5 Моделювання шаблону web-сервісу на основі CMS

Мова CMS-ML (точніше, метамодель моделювання шаблону веб-сайту, як показано на рисунку 3.5) надає набір орієнтованих на CMS елементів моделювання, оскільки CMS часто надають певну підтримку для цих елементів – шаблону веб-сайту. Дизайнери можуть використовувати для визначення своїх шаблонів для веб-додатків на основі CMS. Модель шаблону веб-сервісу, що складається з екземплярів цих елементів CMS, визначається відповідно до наступного набору представлень (подано на рисунку 3.6):

- gepегляд структури , яка визначає структурні компоненти веб-сервісу;
- gepегляд «Ролі», яка стосується набору обов’язків, які веб-сервіс очікує від своїх користувачів;
- gepегляд дозволів, у якому вказується, які ролі мають доступ до структурних компонентів веб-сервісу.

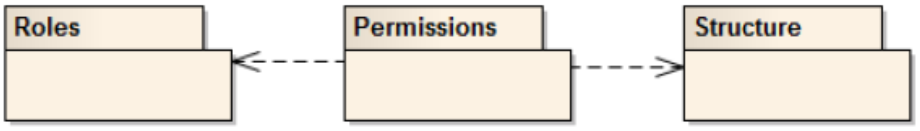


Рисунок 3.6 – Представлення, що застосовуються у визначенні шаблону веб-сайту

Фокус цих переглядів показує, що шаблон веб-сайту має справу зі структурними проблемами веб-сервісу і тому він буде використовуватися в основному для налаштування CMS, коли веб-сервіс розгортається на ній (наприклад, створення нових сторінок і ролей, якщо вони не існують). Проблеми поведінки, з іншого боку, визначені лише в наборах інструментів, оскільки:

- поведінка веб-сервісу на основі CMS зазвичай визначається веб-компонентами, доступними в CMS (наприклад, веб-компонент HTML поводитиметься інакше ніж ForumWebComponent),
- адміністратори CMS зазвичай не можуть змінити саму поведінку CMS (якщо вони не володіють навичками програмування та не мають доступу до вихідного коду платформи CMS) і можуть змінити лише деякі її параметри.

Перегляд «Структура» є найважливішим, оскільки він одразу передає структуру сторінки веб-сервісу за допомогою набору концепцій, орієнтованих на CMS:

1. Веб-сайт, який представляє екземпляри веб-програми та служить одночасно контейнером для динамічних веб-сторінок і як елемент, який імпортуватиме набори інструментів;
2. Динамічна веб-сторінка, що представляє динамічно створені сторінки (в тому сенсі, що їхній вміст можна змінювати через інтерфейс CMS), до яких користувачі матимуть доступ;
3. Контейнер, який моделюється в межах певної області динамічної веб-сторінки та містить набір веб-компонентів;

4. WebComponent, що представляє функціональні одиниці (наприклад, блог або форум), з якими користувач взаємодіятиме.

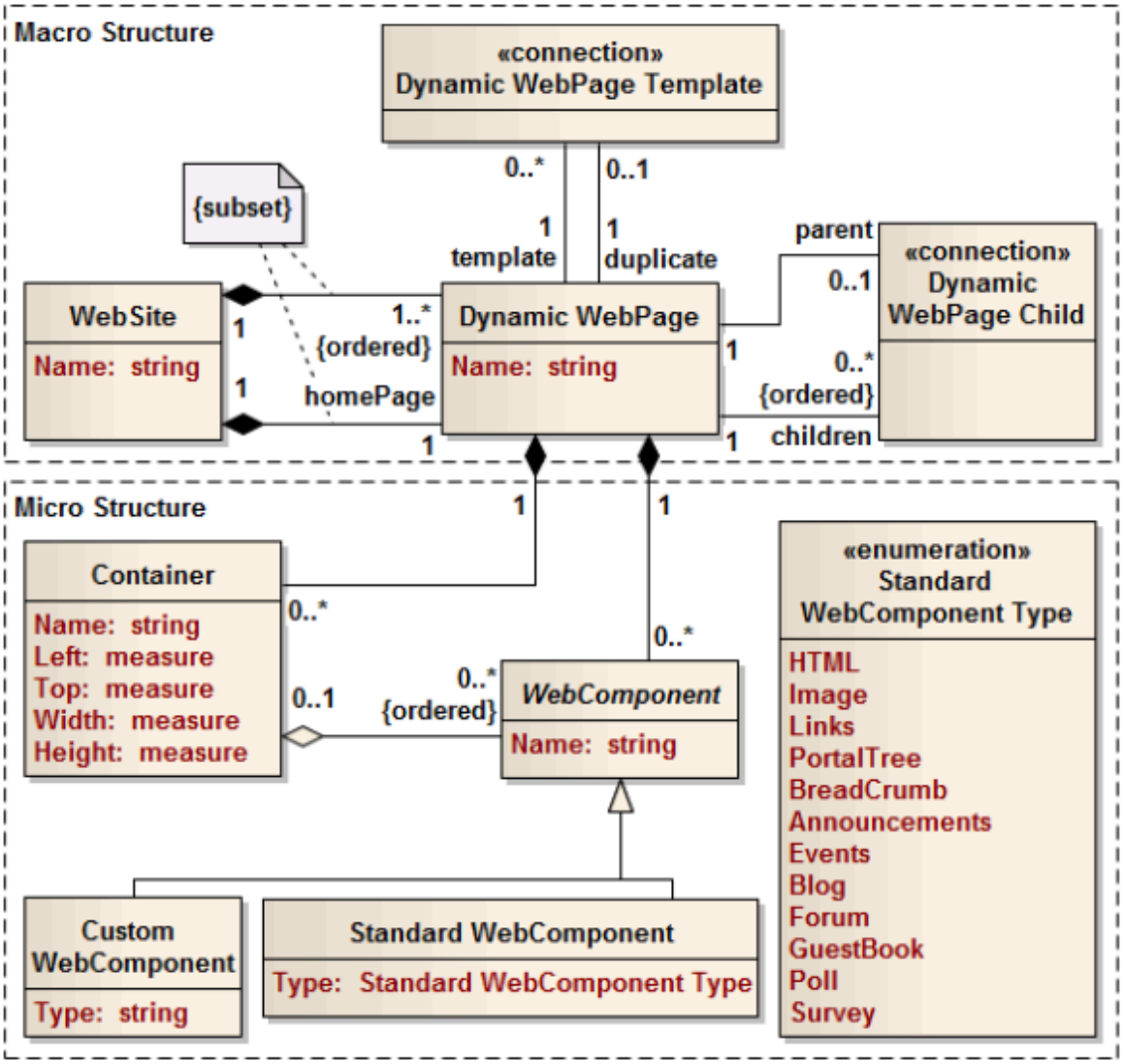


Рисунок 3.7 – Абстрактний синтаксис для перегляду структури шаблону веб-сервісу

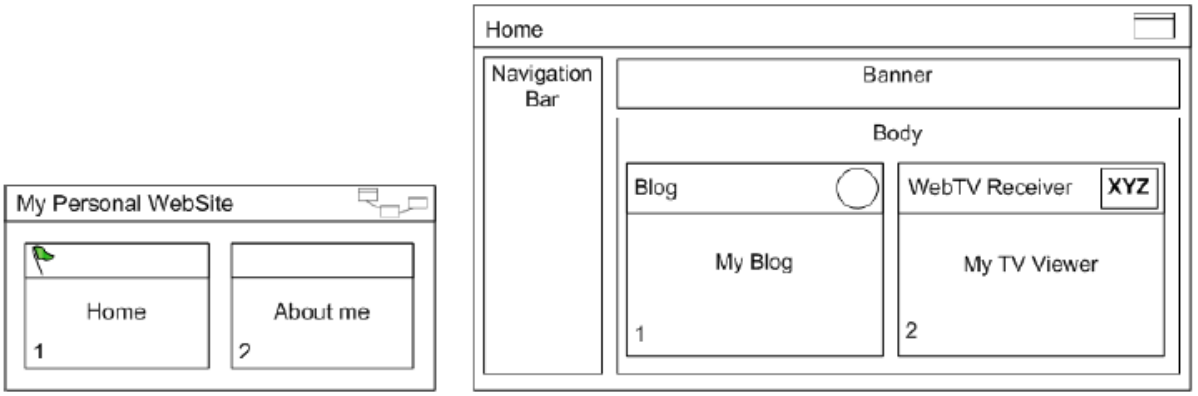
Це подання далі розділено на два менших подання: подання «Макроструктура» та подання «Мікроструктура». Подання макроструктури визначає огляд веб-додатку, моделюючи лише динамічні веб-сторінки та зв’язки між ними, тоді як подання мікроструктури вказує внутрішню структуру кожної динамічної веб-сторінки (тобто, які веб-компоненти є в

кожній динамічній веб-сторінці їхні розташування та їх порядок відносно один одного).

На рисунку 3.7 представлено абстрактний синтаксис перегляду структури, де показано концепції переглядів макроструктури та мікроструктури і зв'язки між ними.

З іншого боку, на рисунку 3.8 зображено два приклади конкретного синтаксису представлення структури: на рисунку 3.8 а показано подання структури макросу, а саме простий веб-сайт, мій особистий веб-сайт, який містить дві динамічні веб-сторінки - Домашня сторінка та Про мене.

На рисунку 3.8 б показано визначення вищезгаданої динаміки в поданні Micro Structure: домашня сторінка, а саме її три контейнери – банер, основний вміст і навігаційна панель і дві компоненти – мій блог і мій переглядач ТВ. Ці приклади також показують, що конкретний синтаксис CMS-ML було визначено з метою легкого розуміння та рисування вручну, не вимагаючи явного використання спеціалізованих інструментів моделювання для створення моделей.



а) представлення макроструктур б) представлення мікроструктура

Рисунок 3.8 – Конкретний синтаксис для перегляду структури шаблону сайту

Подання «Ролі» описує різні типи очікуваних обов'язків, які користувачі веб-програми на основі CMS мають мати під час взаємодії з нею. Це представлення визначає дві концепції, Роль і Делегування Ролі: перша моделює ці очікувані обов'язки, тоді як остання визначає, чи можуть такі

обов’язки також виконуватися іншими ролями, тобто чи роль ділитиме свої обов’язки з іншими ролями. На рисунку 3.7 показано абстрактний синтаксис перегляду ролей.

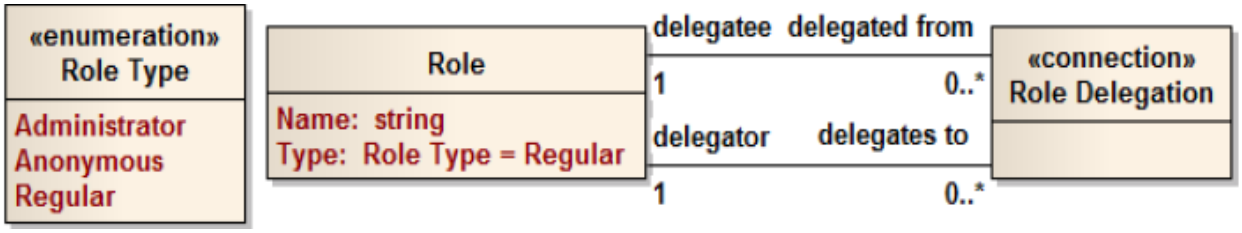


Рисунок 3.7 – Абстрактний синтаксис перегляду ролей шаблону веб-сайту

Слід зробити деякі зауваження щодо концепції делегування ролей, а саме щодо її зв’язку з концепцією узагальнення UML (яка також використовується в діаграмах варіантів використання), а також делегування обов’язків між ролями .

Перше зауваження полягає в тому, що делегування ролей не є еквівалентом узагальнення UML.

Це пояснюється тим, що генералізація зазвичай має семантику елемента, який успадковує характеристики (наприклад, атрибути елемента) від іншого елемента та потенційно їх вдосконалює. З іншого боку, єдина мета делегування ролі — вказати, що роль може нести ті самі обов’язки, що й інша роль, що відрізняється від того, що « роль А є роллю Б », а не вдосконалення ролі.

Крім того, делегування ролей не надає можливості частково делегувати обов’язки. Це пояснюється тим, що якщо таке делегування є необхідним, його можна легко досягти шляхом повторного визначення існуючих ролей у результаті делегування «меншими» ролями у тому сенсі, що менші ролі мають менше обов’язків, ніж існуючі. Більш конкретно, це робиться наступним шляхом:

- створення кількох менших ролей, кожна з яких представляє відповідальність, яку потрібно делегувати;

- делегування цих нових ролей уже існуючим ролям;
- делегування деяких із цих нових ролей іншим ролям відповідно до бажаного часткового делегування обов'язків.

Перегляд «Дозволи» встановлює відображення між структурою веб-сервісу та її ролями, а саме для визначення того, хто і що може робити. Він визначає два поняття, дозвіл на динамічну веб-сторінку та дозвіл на веб-компонент, які відповідно дозволяють створювати посилання «Роль – динамічна веб-сторінка» та «Роль – веб-компонент». Рисунок 3.8 представляє дані поняття.

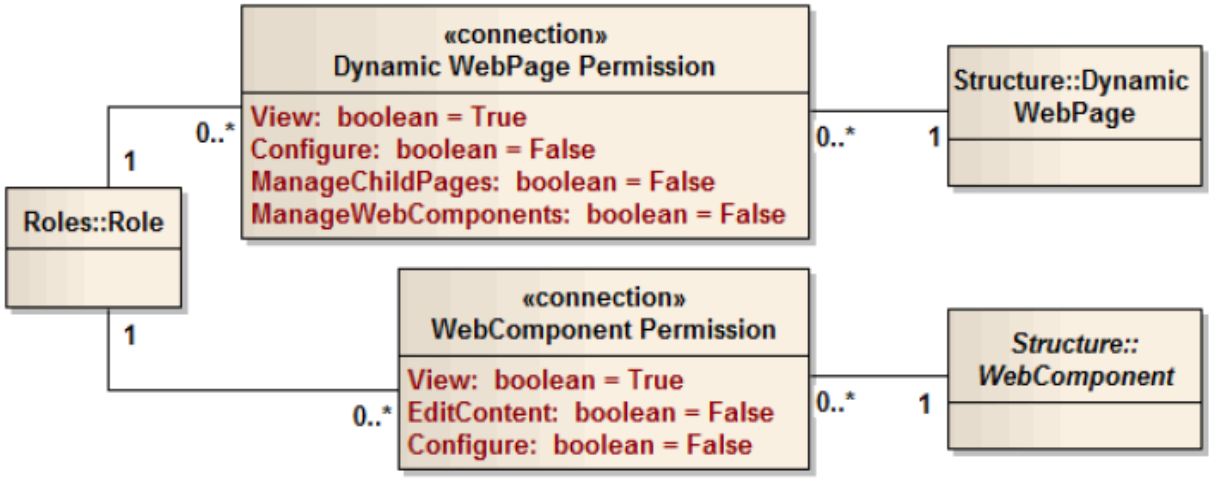


Рисунок 3.8 – Абстрактний синтаксис перегляду дозволів шаблону

Перш ніж перейти до опису перегляду дозволів, важливо розрізнити терміни «конфігурація» та «керування» в контексті цього перегляду. Ця відмінність полягає в наступному:

- налаштування динамічної веб-сторінки або веб-компонента полягає в установці нових значень для його внутрішніх характеристик (тобто його властивостей), таких як ім'я або зрозумілий URL;
- керування динамічною веб-сторінкою не включає її конфігурацію, а скоріше змінює деякі її зовнішні характеристики тобто її асоціації з іншими елементами, наприклад, веб-компоненти, включені до динамічної веб-сторінки.

Дозвіл динамічної веб-сторінки визначає дії, які роль може виконувати над нею. Він визначає дозволи для перегляду та налаштування динамічної веб-сторінки, а також керування її веб-компонентами та дочірніми сторінками. З іншого боку, дозвіл веб-компонента визначає дії, які роль може виконувати над веб-компонентом.

Він також визначає дозволи для перегляду, а також для його налаштування та редагування вмісту.

Кожен із визначених дозволів має значення за замовчуванням, отримане з того, що зазвичай дозволено будь-якому користувачеві, який переглядає веб-сайт, а саме дозволи на перегляд за замовчуванням мають значення True, тоді як дозволи на редагування, конфігурацію та керування за замовчуванням мають значення False.

Перегляд дозволів може бути представлений двома альтернативними способами: графічним способом або набором матриць дозволів. Графічний спосіб є більш придатним для передачі простіших наборів дозволів, тоді як матричне представлення дозволяє представляти більші набори дозволів у більш компактний та ефективний спосіб.

Щодо представлення перегляду дозволів за допомогою набору матриць дозволів, це представлення може бути представлено:

- матрицею, що містить зіставлення дозволів між ролями та динамічними веб-сторінками, яка називається матрицею дозволів сторінки;
- для кожної динамічної веб-сторінки – матриця, що містить відповідності дозволів між ролями та веб-компонентами сторінки, яка називається матрицею дозволів веб -компонентів.

Значення дозволу вказуються галочками, залежно від того, яке значення є True або False відповідно.

Назви та значення атрибутів дозволу (наприклад, View, Configure) необхідно вказувати явно, лише якщо для них не встановлено значення за замовчуванням, інакше їх можна пропустити (тобто залишити «порожніми»).

3.6 Процес моделювання анотацій web-сервісу

Як згадувалося раніше, модель анотацій веб-сервісу (яка використовує концепції, визначені в метамоделі моделювання анотацій) дозволяє розробникам шаблонів веб-сайтів «додати» анотації (що представляють теги, властивості або загальні обмеження) до шаблону веб-сайту. Ці анотації можуть передавати будь-яку інформацію, наприклад параметри конфігурації вмісту тобто тег Allows Content Subscription, застосований до WebComponent, може вказувати, що він повинен надавати читачам канал RSS або подібний механізм або дані, що стосуються платформи розгортання, наприклад канонічна домашня адреса веб-сайту.

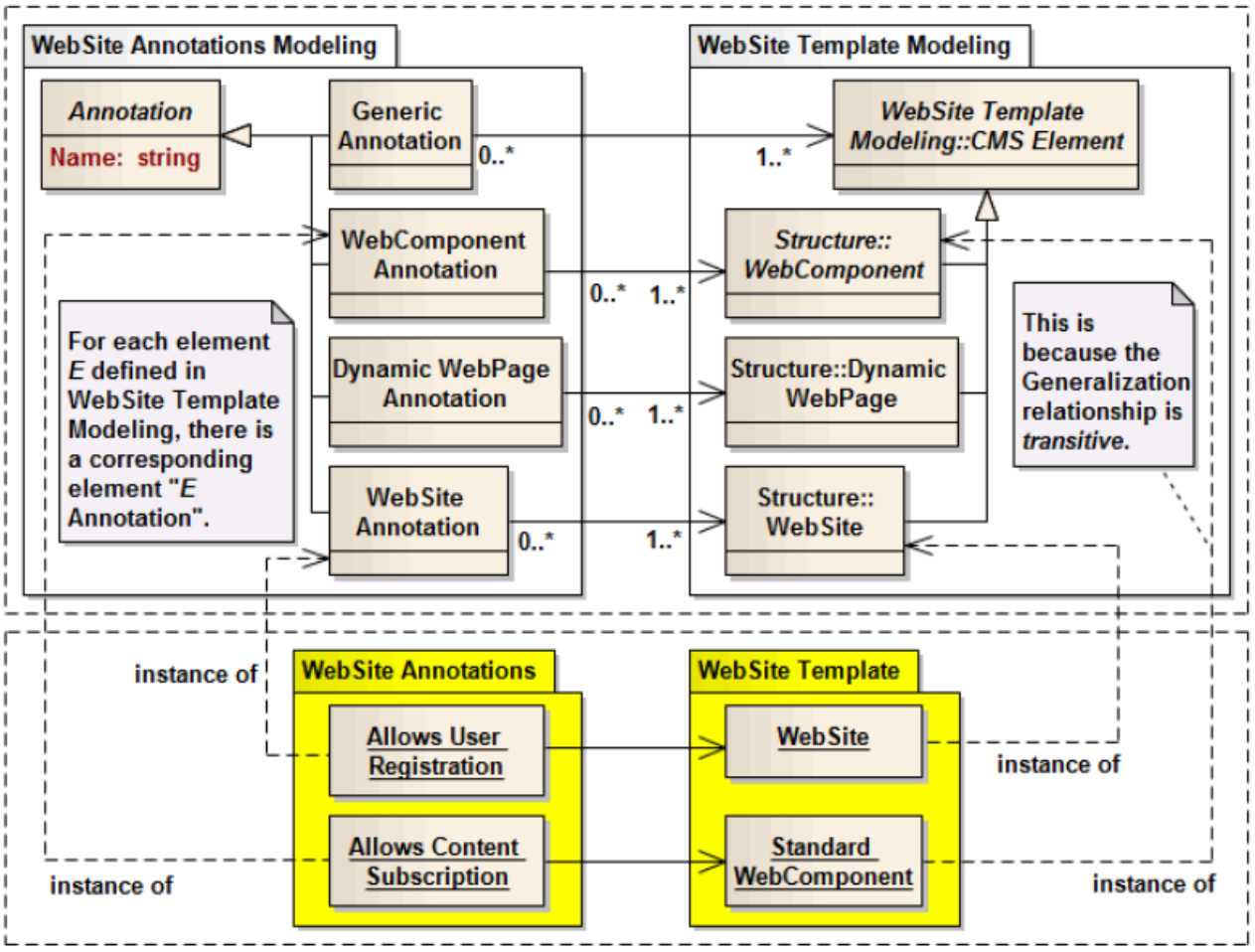


Рисунок 3.9 – Абстрактний синтаксис для моделі анотацій веб-сервісу

На відміну від додаткових, ці анотації призначені для інтерпретації системою CMS, у якій розгорнуто модель. Рисунок 3.9 ілюструє поняття, визначені метамоделлю моделювання анотацій веб-сервісу.

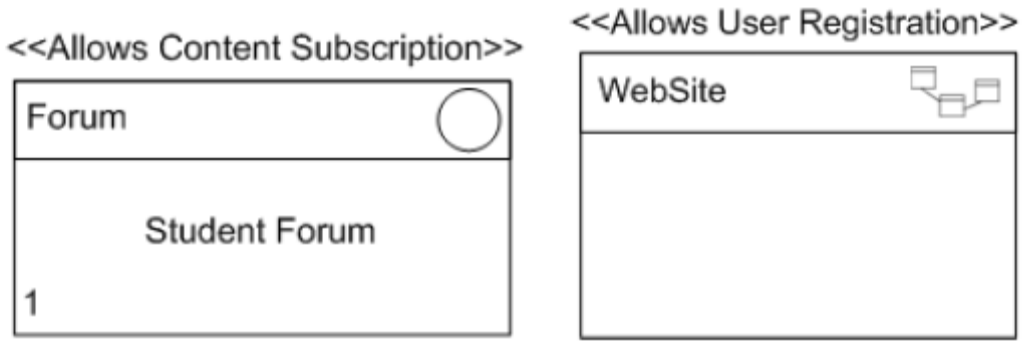
Концепція анотації спеціалізується на інших концепціях у цій метамоделі. Найпростішою є загальна анотація, яку можна пов’язати з будь-яким елементом CMS від якого успадковуються всі елементи моделювання шаблону веб-сайту. Крім того, для кожного визначеного елемента моделювання шаблону веб-сайту, наприклад, Role, WebComponent, DynamicWebPage CMS-ML також визначає елемент анотації під назвою E Annotation, який можна асоціювати лише з екземплярами E. Іншими словами, анотацію ролі можна застосувати лише до ролі шаблону, але не до веб-компонента. Єдина анотація, яку можна застосувати до елементів різних типів, це загальна анотація.

На рисунку 3.10 наведено кілька прикладів елементів анотації, застосованих до елементів моделі шаблону веб-сайту, а саме:

- рисунок 3.10 а ілюструє анотацію WebComponent, позначену Allows Content Subscription, яка застосовується до стандартного WebComponent CMS типу Forum і вказує, що WebComponent має дозволяти користувачам підписуватися на оновлення його вмісту (через такий механізм, як RSS, залежно від CMS). При цьому анотація веб-компонента може бути застосована до стандартного веб-компонента, оскільки останній є спеціалізацією веб-компонента, а зв’язок узагальнення/спеціалізації є перехідним;
- на рисунку 3.10 б зображено анотацію веб-сайту «Дозволяє реєстрацію користувачів», застосовану до веб-сайту та вказуючи, що CMS у якій розгорнуто модель CMS-ML має дозволяти користувачам реєструватися в ній щоб стати звичайними користувачами, а не анонімним.

Слід зазначити, що модель анотації веб-сайту доповнює модель шаблону веб-сайту, дозволяючи дизайнерам визначати специфічні властивості CMS, не загромаджуючи модель шаблону деталями, що

стосуються платформи. Тим не менш, з практичної точки зору, ми очікуємо, що моделювання шаблону веб-сайту виконуватиметься інтегрованим способом: інструмент моделювання не представлятиме дизайнеру шаблонів дві різні робочі моделі (шаблон і анотації), а скоріше одну модель, яка від поєднання цих двох моделей (тобто від оформлення шаблону анотаціями). Власне, це та перспектива, яку також передають приклади на рисунку 3.10.



а) б)

Рисунок 3.10 – Конкретний синтаксис для анотацій

а) Анотація стандарту WebComponent

б) Анотація Web-Site

Можна помітити, що конкретний синтаксис анотації схожий на представлення стереотипу UML. Насправді це представлення було обрано тому, що з точки зору дизайнера з інструментом моделювання UML цей механізм анотації дуже схожий на механізм стереотипів UML. Тим не менш, з концептуальної точки зору, ці два механізми насправді дуже різні: стереотипи UML призначені для розширення метакласів UML (наприклад, Class, Association) і тому їх екземпляри визначені на метарівні M2, де знаходяться метакласи UML – while анотації CMS-ML призначені для доповнення метакласів шаблону веб-сайту (наприклад, WebComponent, Role) і, отже, їх екземпляри визначені на тому ж метарівні, що й екземпляри концепцій шаблону веб-сервісу.

Висновки до розділу 3

Отже, в цьому розділі запропоновано MDE-орієнтований підхід до розробки веб-додатків на основі CMS. Цей підхід відрізняється від інших існуючих підходів до розробки веб-додатків тим, що він використовує кілька мов моделювання (замість однієї мови, такої як UML або WebML) і механізм його модельної синхронізації.

Також використано концепцію CMS-ML, високорівневу та незалежну від платформи мову моделювання, яка має на меті надати нетехнічним зацікавленим сторонам необхідні елементи для моделювання веб-сервісів на основі CMS. На відміну від інших мов моделювання, CMS-ML надає механізм для свого розширення, який дозволяє дизайнерам контрольовано додавати нові елементи моделювання.

ВИСНОВКИ

В кваліфікаційній роботі досліджено методи, концепції та моделі здійснення моделювання контенту в web-рішеннях та сервісах. Запропоновано вдосконалення MDE-орієнтованого підходу до розробки веб-додатків на основі моделей CMS. Дана концепція зосереджена на розробці веб-додатків, які базуються і розгорнуті на системах CMS і відрізняються від інших використанням кількох мов моделювання, а також використанням механізму синхронізації моделі для забезпечення узгодження коду та моделі. Запропонований підхід базується на двох CMS-орієнтованих мовах, CMS-ML і CMS-IL, які розміщені на різних рівнях абстракції. CMS-ML розглядає моделювання на високому рівні, CMS-IL використовує концепції низького рівня, хоча мова не є специфічною для конкретної системи CMS, щоб забезпечити спільну основу для створення мов вищого рівня, таких як CMS-ML.

Використовуючи запропонований підхід зацікавлені сторони такі як бізнес-дизайнери можуть використовувати мову CMS-ML, щоб представити власну точку зору на те, якою має бути система. Коли різні зацікавлені сторони мають бачення системи, з яким вони погоджуються, розробники можуть удосконалити відповідну модель CMS-IL, шляхом додавання або налаштування функцій, які не можуть бути визначені бізнес-дизайнерами через недостатню виразність мови CMS-ML. Коли модель CMS-IL стане достатньою то її можна розгорнути в CMS шляхом генерації вихідного коду або надання моделі CMS-IL як вхідних даних для компонента CMS Model Interpreter, який оброблятиме середовище виконання виконання змодельованого веб-додатку. Хоча цей підхід не усуває потреби в ітераційному процесі розробки, але він надає спосіб пришвидшити додаткову роботу необхідну мінімізації невідповідності між вимогами зацікавлених сторін і розробником.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Atkinson, Colin and Ku¨hne, Thomas. Meta-level Independent Modelling. In International Workshop on Model Engineering at the 14th European Conference on Object-Oriented Programming (ECOOP 2000), pages 12–16. June 2000.

2. Atkinson, Colin and Ku¨hne, Thomas. The Essence of Multilevel Metamodeling. In Martin Gogolla and Cris Kobryn, editors, UML 2001 – The Unified Modeling Language, Modeling Languages, Concepts, and Tools, Fourth International Conference, Proceedings, volume 2185 of Lecture Notes in Computer Science, pages 19–33. Springer, October 2001.

3. Atkinson, Colin and Ku¨hne, Thomas. Profiles in a Strict Metamodeling Framework. Science of Computer Programming, 44(1):5–22, July 2002.

4. Atkinson, Colin and Ku¨hne, Thomas. Model-Driven Development: A Metamodeling Foundation. IEEE Software, 20(5):36–41, September/October 2003.

5. Atkinson, Colin and Ku¨hne, Thomas. Concepts for Comparing Modeling Tool Architectures. In Lionel Briand and Clay Williams, editors, Model Driven Engineering Languages and Systems: 8th InternationalConference, MoDELS 2005, volume 3713 of Lecture Notes in Computer Science, pages 398–413. Springer Berlin/Heidelberg, October 2005.

6. Atkinson, Colin and Ku¨hne, Thomas. Reducing accidental complexity in domain models. Software and Systems Modeling, 7(3):345–359, July 2008.

7. Amar, Bastien, Leblanc, Herv´e, and Coulette, Bernard. A Traceability Engine Dedicated to Model Transformation for Software Engineering. In Jon Oldevik, Gøran K. Olsen, Tor Neple, and Richard Paige, editors, ECMDA Traceability Workshop Proceedings (ECMDA-TW 2008), pages 7–16. SINTEF, June 2008.

8. Abelson, Harold and Sussman, Gerald Jay. Structure and Interpretation of Computer Programs. The MIT Press, second edition, July 1996.

9. Bariˇsi´c, Ankica, Amaral, Vasco, Goul˜ao, Miguel, and Barroca, Bruno. Quality in Use of Domain Specific Languages: a Case Study. In Craig Anslow, Shane Markstrum, and Emerson Murphy-Hill, editors, Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2011), co-located with SPLASH 2011, pages 65–72. ACM, New York, NY, USA, October 2011.

10. Bariˇsi´c, Ankica, Amaral, Vasco, Goul˜ao, Miguel, and Barroca, Bruno. Evaluating the Usability of Domain-Specific Languages. In Marjan Mernik, editor, Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, pages 386–407. Information Science Reference, September 2012.

11. Byron, Angela, Berry, Addison, Haug, Nathan, Eaton, Jeff, Walker, James, and Robbins, Jeff. Using Drupal. O'Reilly Media, December 2008.

12. Brambilla, Marco, Comai, Sara, Fraternali, Piero, and Matera, Maristella. Designing Web Applications with WebML and WebRatio. In Gustavo Rossi, Oscar Pastor, Daniel Schwabe, and Luis Olsina, editors, Web Engineering: Modelling and Implementing Web Applications, Human- -Computer Interaction Series, pages 221–261. Springer London, October 2007.

13. Bariˇsi´c, Ankica, Monteiro, Pedro, Amaral, Vasco, Goul˜ao, Miguel, and Monteiro, Miguel. Patterns for Evaluating Usability of Domain-Specific Languages. In Proceedings of the Pattern Languages of Programs Con- ference (PLoP 2012), co-located with SPLASH 2012. ACM, New York, NY, USA, October 2012.

14. Boiko, Bob. Content Management Bible. John Wiley & Sons, Hoboken, New Jersey, U.S.A., December 2001.

15. Brooks, Jr., Frederick P. No Silver Bullet – Essence and Accidents of Software Engineering. IEEE Computer, 20(4):10–19, April 1987.

16. Carmo, Joa˜o Leonardo Vicente do. Web Content Management Systems: Experiences and Evaluations with the WebComfort Framework. Master's thesis, Instituto Superior T´ecnico, Portugal, December 2006.

17. Ceri, Stefano, Fraternali, Piero, Bongio, Aldo, Brambilla, Marco, Comai, Sara, and Matera, Maristella. Designing Data-Intensive Web Applications. Morgan Kaufmann, December 2002.

18. Cook, Steve, Jones, Gareth, Kent, Stuart, and Wills, Alan. Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley Professional, June 2007.

19. Cox, Brad J. and Novobilski, Andrew J. Object-Oriented Programming: An Evolutionary Approach. Addison-Wesley, second edition, 1991.

20. Cao, Lan, Ramesh, Balasubramaniam, and Rossi, Matti. Are Domain--Specific Models Easier to Maintain Than UML Models? IEEE Computer, 26(4):19–21, 2009.

21. Costa, Marco and Silva, Alberto Rodrigues da. RT-MDD Framework – A Practical Approach. In Jon Oldevik, Gøran K. Olsen, and Tor Neple, editors, ECMDA Traceability Workshop Proceedings (ECMDA-TW 2007), pages 17–26. SINTEF, June 2007.

22. Ducasse, St´ephane, Gˆırba, Tudor, and Favre, Jean-Marie. Modeling Software Evolution by Treating History as a First Class Entity. Electronic Notes in Theoretical Computer Science, 137(3):75–86, September 2005.

23. Diskin, Zinovy. Mathematics of Generic Specifications for Model Management, I. In Laura C. Rivero, Jorge Horacio Doorn, and Viviana E. Ferraggine, editors, Encyclopedia of Database Technologies and Applications, pages 351–358. Idea Group, 2005.

24. Diskin, Zinovy. Mathematics of Generic Specifications for Model Management, II. In Laura C. Rivero, Jorge Horacio Doorn, and Viviana E. Ferraggine, editors, Encyclopedia of Database Technologies and Applications, pages 359–365. Idea Group, 2005.

25. Drivalos, Nicholas, Paige, Richard F., Fernandes, Kiran J., and Kolovos, Dimitrios S. Towards Rigorously Defined Model-to-Model Traceability. In Jon Oldevik, Gøran K. Olsen, Tor Neple, and Richard Paige, editors, ECMDA Traceability Workshop Proceedings (ECMDA-TW 2008), pages 17–26. SINTEF, June 2008.

26. Essalmi, Fathi and Ayed, Leila Jemni Ben. Graphical UML View from Extended Backus-Naur Form Grammars. In Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006), pages 544–546. IEEE Computer Society, Los Alamitos, CA, USA, July 2006.

27. Ehrig, Hartmut, Engels, Gregor, Kreowski, Hans-J¨org, and Rozenberg, Grzegorz, editors. Handbook of Graph Grammars and Computing by Graph Transformation: Volume 2: Applications, Languages and Tools. World Scientific Pub. Co. Inc., 1999.

28. Favre, Jean-Marie. Foundations of Meta-Pyramids: Languages vs. Metamodels – Episode II: Story of Thotus the Baboon. In Jean B´ezivin and Reiko Heckel, editors, Language Engineering for Model-Driven Software Development, volume 04101 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fu¨r Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany, March 2004.

29. Favre, Jean-Marie. Foundations of Model (Driven) (Reverse) Engineering: Models – Episode I: Stories of The Fidus Papyrus and of The Solarus. In Jean B´ezivin and Reiko Heckel, editors, Language Engineering for Model-Driven Software Development, volume 04101 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fu¨r Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany, March 2004.

30. Favre, Jean-Marie. Megamodelling and Etymology. In James R. Cordy, Ralf La¨mmel, and Andreas Winter, editors, Transformation Techniques in Software Engineering, volume 05161 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fu¨r Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany, April 2005.

31. France, Robert B., Ghosh, Sudipto, Dinh-Trong, Trung T., and Solberg, Arnor. Model-Driven Development Using UML 2.0: Promises and Pitfalls. IEEE Computer, 39(2):59–66, February 2006.

32. Flanagan, David and Matsumoto, Yukihiro. The Ruby Programming Language. O'Reilly Media, February 2008.

33. Fowler, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley, 2003.

34. Gırba, Tudor, Favre, Jean-Marie, and Ducasse, St´ephane. Using Meta-Model Transformation to Model Software Evolution. Electronic Notes in Theoretical Computer Science, 137(3):57–64, September 2005.

35. Gabriel, Pedro, Goul˜ao, Miguel, and Amaral, Vasco. Do Software Languages Engineers Evaluate their Languages? In Xavier Franch, Itana Gimenes, and Juan Pablo Carvallo, editors, Proceedings of the XIII Iberoamerican Congress on Software Engineering (CIbSE 2010), pages 149–162. April 2010.

36. Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John. De- sign Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

37. Ghosh, Debasish. DSL for the Uninitiated. Communications of the ACM, 54(7):44–50, July 2011.

38. Gerber, Anna and Raymond, Kerry. MOF to EMF: There and Back Again. In Michael G. Burke, editor, Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange, pages 60–64. ACM Press, New York, NY, USA, October 2003.

39. Greenfield, Jack, Short, Keith, Cook, Steve, and Kent, Stuart. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, August 2004.

40. Gaffar, Ashraf, Sinnig, Daniel, Seffah, Ahmed, and Forbrig, Peter. Mod-eling Patterns for Task Models. In Pavel Slav´ık and Philippe A. Palanque, editors, Proceedings of the 3rd International Workshop on Task Models and

Diagrams for User Interface Design (TAMODIA 2004), pages 99–104. ACM, New York, NY, USA, 2004.

41. Giese, Holger and Wagner, Robert. From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling, 8:21–43, 2009.

42. Henderson-Sellers, Brian. UML – the Good, the Bad or the Ugly? Perspectives from a panel of experts. Software and Systems Modeling, 4(1):4– 13, February 2005.

43. Haustein, Stefan and Pleumann, Jo¨rg. A model-driven runtime environment for Web applications. Software and System Modeling, 4(4):443–458, 2005.

44. Hammond, Christopher J., Renner, Patrick, and Walker, Shaun. DotNetNuke 5 User's Guide: Get Your Website Up and Running. Wrox, June 2009.

45. Hughes, John. Why Functional Programming Matters. In David A. Turner, editor, Research Topics in Functional Programming (The UT year of programming series), pages 17–42. Addison-Wesley Pub (Sd), June 1990.

46. IEEE. IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology, September 1990.

47. IEEE. IEEE Std 1471-2000: IEEE Recommended Practice for Archi-tectural Description of Software-Intensive Systems, September 2000.

48. Jouault, Fr´ed´eric and B´ezivin, Jean. KM3: A DSL for Metamodel Specification. In Roberto Gorrieri and Heike Wehrheim, editors, Formal Methods for Open Object-Based Distributed Systems, 8th IFIP WG 6.1 International Conference Proceedings (FMOODS 2006), volume 4037 of Lecture Notes in Computer Science, pages 171–185. Springer Berlin/Heidelberg, June 2006.

49. Jenkins, Tom. Enterprise Content Management Solutions: What You Need to Know. Open Text Corporation, April 2005.

50. Koch, Nora and Kraus, Andreas. The Expressive Power of UML-based Web Engineering. In Proceedings of the Second International Workshop on Web-Oriented Software Technology (IWWOST'2002). June 2002.

метадані

Заголовок

Покращення ефективності методів та засобів моделювання контенту в web-рішеннях та сервісах

Автор

Рущак О.Я.. Науковий керівник / Експерт

підрозділ

King Danylo University

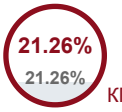
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

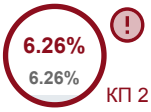
Заміна букв	ß	5
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	377

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КП 2

25
Довжина фрази для коефіцієнта подібності 2

17450
Кількість слів

127260
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	https://ela.kpi.ua/bitstream/123456789/23195/1/Tkachenko_magistr.pdf	70	0.40 %
2	https://ela.kpi.ua/bitstream/123456789/23195/1/Tkachenko_magistr.pdf	55	0.32 %
3	http://repository.ukd.edu.ua/bitstream/handle/123456789/391/%D0%9F%D0%B0%D1%85%D0%BE%D0%BB%D1%8C%D1%87%D1%83%D0%BA%20%D0%9E.%D0%A0.%20%D0%B4%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%D0%BD%D0%B0.pdf?sequence=1	53	0.30 %
4	https://ela.kpi.ua/bitstream/123456789/23195/1/Tkachenko_magistr.pdf	47	0.27 %
5	http://repository.ukd.edu.ua/bitstream/handle/123456789/391/%D0%9F%D0%B0%D1%85%D0%BE%D0%BB%D1%8C%D1%87%D1%83%D0%BA%20%D0%9E.%D0%A0.%20%D0%B4%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%D0%BD%D0%B0.pdf?sequence=1	45	0.26 %