

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

_____ Литвиненко О.Є.

“ _____ ” _____ 2020 р.

**ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

**ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
“МАГІСТР”**

Тема: Технологія розробки вебдодатків на базі CMS

Виконавець: _____ Покляцький В.А.

Керівник: _____ Кучерява О. М.

Нормоконтролер: _____ Тупота Є. В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютера інженерія»

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Литвиненко О.Є.

«_____» _____ 2020 р.

ЗАВДАННЯ

на виконання дипломної роботи

Покляцького Вадима Андрійовича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи «Технологія розробки вебдодатків на базі CMS»

затверджена наказом ректора від 07.09.2020 р. № /ст 1410

2. Термін виконання роботи: з 5 жовтня 2020 року по 2020 року

3. Вихідні дані до роботи: система управління вмістом сайту, його структурою і дизайном *Drupal*, системи керування вмістом *Viagnette*, *WordPress* та *Joomla*.

4. Зміст пояснювальної записки: _____

1) Аналіз проблематики області дослідження

2) Методи дослідження

3) Розробка додатків

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1. Фазовий цикл методу *Action Research*

2. _____

3. _____

4. _____

5. _____

6. _____

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомлення з постановкою завдання дипломної роботи	14.10.2020 – 16.10.20	
2	Аналіз літературних джерел та інтернет-ресурсів	17.10.20 – 18.10.20	
3	Аналіз існуючих проблем	19.10.20 – 24.10.20	
4	Написання «Вступу» та 1-3 розділів	25.10.20 – 07.11.20	
5	Підготовка графічного матеріалу	08.11.20 – 01.12.20	
6	Оформлення пояснювальної записки	2.12.20 – 03.12.20	
7	Оформлення графічного матеріалу	10.12.20 – 13.12.20	

7. Дата видачі завдання: 9 вересня 2020 р.

Керівник дипломної роботи _____ к.ф.-м.н., доцент Кучерява О. М.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Покляцький В. А.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Технологія розробки вебдодатків на базі *CMS*» містить 94 сторінки, 2 таблиці, 8 рисунків, 25 літературних джерела, 1 додаток.

***CMS*-СИСТЕМА, МОВА МОДЕЛЮВАННЯ, ВЕБДОДАТОК, РОЗРОБКА УНІФІКОВАНИЙ ЛОКАТОР РЕСУРСІВ.**

Об'єкт дослідження – розробка вебдодатків.

Предмет дослідження – технологія розробки вебдодатків.

Мета дипломної роботи – технологія розробки вебдодатків на базі *CMS* та автоматизація переходів з рівня *CMS-ML* на *CMS-IL*.

Методи дослідження – система управління вмістом сайту, його структурою і дизайном *Drupal*, системи керування вмістом *Viagnette*, *WordPress* та *Joomla*.

В дипломній роботі здійснено аналіз існуючих систем керування вмістом, здійснено їх порівняльну характеристику, виявлено переваги та недоліки. Досліджено 3 способи функціонування *CMS*-систем (генерація за запитом, генерація при редагуванні та змішаний тип). Створено та проведено тестування системи перетворення з рівня *CMS-ML* на рівень *CMS-IL*.

Матеріали дипломної роботи можуть бути використані розробниками контенту з низьким рівнем володіння мовами програмування, тобто авторам текстів, фотографам, графічним художникам, відеомейкерам.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ	7
ВСТУП	8
РОЗДІЛ 1 РОЗРОБКА ДОДАТКІВ	Ошибка! Закладка не определена. 4
1.1. Загальні відомості, цілі та завдання	14
1.2.1. Критерії аналізу	15
1.2.2 Результати аналізу	17
1.3. Додаткова супутня робота	Ошибка! Закладка не определена.
1.4. Висновки до розділу	22
РОЗДІЛ 2 СИСТЕМИ УПРАВЛІННЯ ВМІСТОМ	23
2.1 Порівняльний аналіз	23
2.2 Порівняльний аналіз	25
2.2.1 Критерії аналізу	28
2.2.2 Результати аналізу	30
2.3. Додаткова супутня робота	33
2.4. Висновки до розділу	34
РОЗДІЛ 3 БАГАТОМОВНА СИСТЕМА УПРАВЛІННЯ ВМІСТОМ.	
МОДЕЛЬНИЙ ПІДХІД	35
3.1 Проблеми	35
3.2. Пропозиційне рішення: багатомовний підхід	37
3.2.1. <i>CMS-ML</i>	38
3.2.2 <i>CMS-IL</i>	39
3.2.3. <i>MYNK</i>	40
3.3. Процес розробки	40
3.4. Вказівки щодо специфікації мови	43
3.5. Висновки до розділу	48
РОЗДІЛ 4 <i>CMS-ML</i> : МОВА МОДЕЛЮВАННЯ <i>CMS</i>	35
4.1. Проблеми <i>CMS-ML</i>	49
4.2. Види моделі та моделюючі ролі	51
4.3. Архітектура <i>CMS-ML</i>	53

4.4. Моделювання шаблонів вебсайтів.....	56
4.4.1. Перегляд структури	57
4.4.2. Перегляд ролей	59
4.5. <i>MYNK</i> : синхронізація моделі	61
4.6. Поточні проблеми трансформації моделі	62
4.7. Огляд <i>MYNK</i>	63
4.8. <i>MYNK</i> і <i>ReMMM</i> метамодель	65
4.9. <i>MYNK</i>	68
4.10. Конкретний синтаксис	71
4.11. Вирішення конфліктів	78
4.12. Характеристики мов, сумісних з <i>MYNK</i>	81
4.13. Висновки до розділу	83
РОЗДІЛ 5 ПЕРЕВІРКА	84
5.1. Порівняння мов вебмоделювання.....	84
5.2. <i>CMS-ML</i>	89
5.3. <i>CMS-IL</i>	89
5.4. <i>MYNK</i>	92
5.4.1. Синхронізація змін <i>CMS-ML</i> та <i>CMS-IL</i>	92
5.5. Висновки до розділу	93
ВИСНОВКИ	93
СПИСОК БІБЛОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ.....	95
Додаток А	97

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API (англ. *Application Programming Interface*) – інтерфейс програмування додатків

CASE (англ. *Computer-Aided Software Engineering*) – набір інструментів і методів програмної інженерії для проектування програмного забезпечення

CMS (англ. *Content Management System*) – система керування вмістом

DSL (англ. *Domain-specific language*) – предметно-орієнтована мова

програмування *DSM* (англ. *Domain Specific Modeling*) – домен-орієнтоване моделювання

ECM (англ. *enterprise content management*) – управління документами та іншими типами контенту.

HTML (англ. *HyperText Markup Language*) – стандартна мова розмітки вебсторінок в Інтернеті.

IL (англ. *Intermediate Language*) – проміжна мова

MDA (англ. *Model Driven Architecture*) – модельно-орієнтований підхід до розробки програмного забезпечення

MDE (англ. *Model-Driven Engineering*) – модель-орієнтована інженерія

ML (англ. *Modeling Language*) – мова моделювання

MOF (англ. *Meta Object Facility*) – стандарт мета-моделювання *OMG*

MYNK (англ. *Model sYNchronization frameworK*) – засіб для синхронізації моделей

OMG (англ. *Object Management Group*) – некомерційна міжнародна організація

SaaS (англ. *Software as a service, SaaS*) – програма, як послуга

UML (англ. *Unified Modeling Language*) – уніфікована мова моделювання

URL (англ. *Uniform Resource Locator*) –адреса ресурсу

XMI (англ. *XML Metadata Interchange*) – стандарт консорціуму *OMG*, для обміну мета-інформацією за допомогою та на основі *XML*

XML (англ. *Extensible Markup Language*) – стандарт побудови мов розмітки ієрархічно структурованих даних

ВСТУП

Актуальність теми. Зі світовим розширенням Інтернету та Всесвітньої павутини в останні роки ми стали свідками стрімкого зростання популярності вебдодатків. Хоча деяка література визначає вебдодаток як виконувану програму, яка широко використовує вебконцепції та технології (такі як запит або *HTTP*) зв'язку. Вебдодаток складається з програми, яка використовує вебтехнології і доступ до них (і використання) здійснюється через веб-браузер. Системи управління вмістом (*CMS*), нещодавно отримали особливу популярність та актуальність, оскільки вони полегшують розповсюдження широкого спектру вмісту нетехнічними користувачами. Хоча деякі з цих систем *CMS* також забезпечують підтримку та розробку більш складних вебдодатків, що базуються на них, ця розробка все ще робиться за допомогою традиційних (і схильних до помилок) методів, орієнтованих на вихідний код. З іншого боку, так само, як парадигми розробки програмного забезпечення змінювались протягом останнього десятиліття, поточна парадигма розвитку змінюється на третє покоління мови програмування на основі модельної інженерії (*MDE*). Це дедалі популярніша парадигма, метою якої є досягнення вищого рівня абстракції, ніж те, що є сьогодні, виступає за використання моделей як найважливіші артефакти в процесі розробки програмного забезпечення. Інші артефакти, такі як вихідний код та Протокол Передачі Гіпертексту – *HTTP* документація, може бути автоматично виготовлена з цих моделей за допомогою моделі механізму трансформації.

У вебдодатоку, процеси розвитку, які доступні сьогодні все ще існують проблеми, а саме їх неадекватність вирішувати різні специфіки різних зацікавлених сторін системи: це, як правило, відповідальність розробника об'єднати ці перспективи в єдину цілісну систему. Це, в свою чергу, призводить до необхідності подальших змін або вдосконалення системи через

неправдоподібність перша ітерація розробленої системи відповідатиме очікуванням усіх її зацікавлених сторін.

Крім того, більшість сучасних підходів *MDE* для розробки вебдодатків все ще враховують вихідний код, а не моделі, як «остаточний» артефакт, який буде розгорнуто або скомпільовано; оскільки дизайн моделі зазвичай розглядається як проміжний етап у процесі, призначеному для отримання вихідного коду, який буде налаштований на пізнішому етапі процесу розробки.

Мета і завдання дипломної роботи. Головною метою дипломної роботи є технологія розробки вебдодатків на базі *CMS* та автоматизація переходів з рівня *CMS-ML* на *CMS-IL*. Розробка вебдодатків у поєднанні з новими перспективами масштабування та розподілу віддають належне додатку (наприклад, *Software-as-a-Service*) якомога більшій кількості користувачів стати способом за замовчуванням створювати нові програми (або оновлювати старі). В даний час є кілька модельованих підходів до розробки вебдодатків. Однак, підхід до розвитку, як правило, визначається виразністю мови моделювання що використовується в цьому підході.

Об'єкт і предмет дослідження. Об'єктом дослідження даної роботи є розробка вебдодатків. Предметом даної роботи є технологія розробки вебдодатків.

Методи дослідження. В дипломній роботі досліджено методи функціонування системи управління вмістом сайту, його структурою і дизайном *Drupal*, системи керування вмістом *Viagnette*, *WordPress* та *Joomla*. Наступні підходи *MDE* для розробки вебдодатків мовою та їх допоміжними мовами *WebML*, *UWE*, *XIS2*, *Microsoft Sketchflow*, *OutSystems Agile Platform*.

Для аналізу було обрано підходи *WebML*, *UWE* та *OutSystems Agile Platform* в цій дипломній роботі (замість інших можливих підходів *MDE*), оскільки вони добре відомі в області розробки вебдодатків. У цьому аналізі також розглядається *XIS2*. Нарешті, хоча *Microsoft Sketchflow* не є підходом *MDE* або мовою як такою, і він не є спеціально орієнтованим у вебдодатках я включу його в цей аналіз, оскільки його можна використовувати для генерації крос-

платформних інтерактивних вебдодатків, і тому, що вони стосуються дуже важливих аспектів, які не розглядаються в інших підходах: забезпечення зацікавлених сторін дизайн додатків.

Наукова новизна отриманих результатів. Дипломна робота спрямована з наміром отримати подальший досвід у галузі в моделі керованої техніки і змінити та вдосконалити спосіб, в якому система управління вмістом вебдодатків на базі розробляються шляхом застосування методів, орієнтованих на *MDE* цього домену. Через характер цієї роботи та враховуючи вищезазначений намір, я вирішив виконати дипломну роботу, використовуючи метод дії дослідження. *Action Research* – це метод дослідження, який зосереджується на постановці запитань, збір даних та отримання відповідних результатів безперервним та циклічним способом. Цей цикл складається з наступних фаз, які виконуються в зазначеному порядку:

1. Діагностика: ця фаза складається з чіткого виявлення та визначення проблеми під рукою;
2. Планування дій: на цьому етапі дослідник вибирає потенційний курс дії, яка, як вважають, адекватно вирішує виявлену проблему, та розробляє відповідний розчин;
3. Вжиття заходів: цей етап бачить дослідник у співпраці з учасниками (наприклад, клієнти), застосовуючи рішення в експерименті;
4. Оцінка: на цьому етапі дослідник збирає дані, отримані в результаті формування експерименту та його аналіз;
5. Конкретизація навчання: саме на цій фазі дослідник визначає уроки дізналися під час експерименту. Це може призвести до одного з таких результатів: дослідження вважається завершеним; або необхідні подальші дослідження для вирішення проблеми. У випадку останнього цикл повторюється, але зараз з додатковими знаннями, отриманими на цьому етапі.

Практичне значення отриманих результатів. Ця дипипломна робота представляє застосування підходу до розробок *CMS* на основі вебдодатків. На відміну від інших підходів до розвитку, які зазвичай використовують одну мову,

таку як *C #*, *Java*, або *PHP 2*, орієнтований на аудиторію технічних розробників, цей підхід фокусується на зверненні до потреби нетехнічних зацікавлених сторін, а також розробників. Для цього пропонується набір моделювання на різних рівнях абстракції та модель синхронізації. Ці механізми забезпечують узгодженості між моделями для різних зацікавлених сторін.

Під час виконання дипломної роботи було використано наступні методи дослідження: аналіз сучасного рівня (який іноді називають дисциплінованим оглядом літератури або систематичний огляд), щоб визначити особливості, які мали значення для нашого дослідження; і техніки, засновані на техніці, а саме техніка проектування та побудови прототипу системи для перевірки гіпотези.

РОЗДІЛ 1

АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБДОДАТКІВ

1.1. Модельовані підходи при розробці вебдодатків

Інтернет став потужною платформою для розміщення різноманітних артефактів та систем, а також змінив спосіб розробки додатків. Це прогрес у розробці – від багатофункціональних програм на настільних комп'ютерах до вебпрограм доступних звичайним веб-браузерам, призвів до появи безлічі веб-орієнтованих фреймворків та бібліотек, які дозволяють розробникам використати потужність інтернет-технологій для досягнення цілей різного роду. Головна відмінність між фреймворком (також відомим як платформа в контексті вебдодатків) та бібліотекою полягає в тому, що останній – це просто код, який розробник може (повторно) використовувати в додатку, тоді як фреймворк зазвичай забезпечує ряд хуків, до яких розробник може (або повинен) забезпечувати функціональність та загальну функціональність, яку може розробник вибрати, щоб замінити або налаштувати. Приклади відомих веб-орієнтованих фреймворків включають *Microsoft ASP.NET* 3, *JavaEE* 4 та *Java Server Pages (JSP)* 5. Все більш популярним результатом цього зрушення є багато веб-орієнтованих та *ECM (Enterprise Content Management)* систем, які зараз доступні, що мають основну мету сприяти управлінню та публікації цифрового вмісту в інтрамережі, або навіть в Інтернеті.

Через різні цілі кожної структури (що, у свою чергу, впливає на характер гачків та функціональність, яку він забезпечує), вони часто супроводжують підходи до розробки простих вебдодатків, заснованих на цій структурі. Однак Інтернет змінився там, де ми розгортаємо додатки, але не як ми розвиваємо ці програми, оскільки їх розробка все ще здійснюється вручну. З іншого боку, парадигма *MDE (Model-Driven Engineering)* має на меті прибрати значення, яке поточні процеси розробки програмного забезпечення все ще надають

вихідному коду, а натомість зосередитися на моделях та концепціях із проблемного та цільового доменів.

Модельовані підходи. Ці підходи стають дедалі популярнішими завдяки появі парадигми модельованої інженерії (*MDE*). Як раніше згадувалося, виступає за використання моделей в якості основних артефактів в програмному забезпеченні процесу розробки, тоді як артефакти, такі як документація та вихідний код, можуть бути отримані з цих моделей за допомогою автоматизованих перетворень моделей. Ця орієнтація до моделей дозволяє розробникам зосередитись на відповідних концепціях. Крім того, залишаючи більшість повторюваних і схильних до помилок завдань для моделювання перетворень підходи *MDE* також мають додаткові переваги, такі як: звільнення розробників від таких проблем, як основна складність платформи або нездатність мов програмування виражати поняття домену; або націлювання на розгортання на декількох платформах, не вимагаючи різних баз коду.

Хоча існують деякі приклади підходів *MDE* – найвідоміший з них, можливо, модель управління архітектурою (*MDA*). Групи управління об'єктами – більшість розробників все ще використовують *MDE* в контексті домен-специфічного моделювання (*DSM*), в якому візуальна мова визначена і орієнтована на конкретний домен, і розроблені моделі будуть використовуватися для автоматичного генерування відповідного вихідного коду. Таким чином, сам процес розробки програмного забезпечення все ще є кодоорієнтованим, хоча деякі його частини спрощені.

Підходи на основі *CMS* можна використовувати не лише для створення вебсайтів, але й як підтримку платформи для розгортання спеціалізованих вебдодатків. Перш ніж продовжити, важливо встановити різницю між поняттями вебдодатків та вебсайтів, в контексті даної роботи: вебдодаток – це програма, яка використовує вебтехнології тоді як вебсайт складається з конкретного використання вебпрограми.

Ці системи управління вмістом (які самі є вебдодатками) можуть ефективно підтримувати динамічне управління вебсайтами та їх вмістом і, як правило, присутні такі аспекти, як розширюваність та модульність, незалежність між змістом та презентацією, підтримка декількох типів вмісту, підтримка управління доступом та управління користувачем, або підтримка визначення та виконання робочого процесу. З іншого боку, системи управління корпоративним вмістом (*ECM*), як правило, орієнтовані на вебпрограми до використання Інтернет-технологій та робочих процесів для збору, управління, зберігання, збереження, і доставки вмісту та документів в контекст організаційних процесів.

Крім того, платформа *CMS* може бути використана для виконання тих самих завдань, що і платформа *ECM*, до тих пір, поки *CMS* надає розробнику адекватні функціональні можливості.

Насправді, деякі системи управління вмістом розвиваються, щоб самі по собі стати справжніми розробниками з набором функцій, які є достатніми для створення вебдодатків з іншими цілями, крім простого управління вмістом. На додаток до цих особливостей, *CMS* система також забезпечує набір понять високого рівня – такі, як користувач, роль або модуль – що полегшує розробку складних вебдодатків.

Однак розробка та розгортання вебдодатків на базі *CMS* все ще є типовою та робиться традиційним способом. Ця розробка передбачає написання вихідного коду вручну який використовує *API* (інтерфейс прикладного програмування), наданий *CMS*, у мові програмування, що підтримується базовою технологією вебсервера *CMS* (наприклад як *C#*, *Java* або *PHP*), компіляція цього вихідного коду та розгортання кінцевого результату до *CMS*, або за допомогою механізму, вбудованого в *CMS*, або вручну копіювання цих артефактів на відповідний вебсервер.

Підходи *MDE* на основі однієї мови. Першою проблемою дослідження є що сучасні підходи до розробки програмного забезпечення, що керуються моделлю (орієнтовані на вебдодатки або іншим чином) зосереджені навколо однієї мови моделювання: або стандарту *OMG UML* або інша мова, специфічна

для підходу. Це не представляло б проблеми якщо розробник був єдиним учасником процесу розробки програмного забезпечення, як мова моделювання буде просто орієнтована на полегшення завдань розробника. Однак у процесі розробки програмного забезпечення беруть участь інші зацікавлені сторони (а саме: кінцеві користувачі програми), кожен з яких має особливу перспективу щодо того, що саме додаток повинен робити. Таким чином, як правило, відбувається те, що інші зацікавлені сторони забезпечити розробника набором вимог, таких як передбачувані функції чи бізнес правил, і відповідальність розробника покладається на те, щоб відобразити ці вимоги до мови реалізації, що використовується (тобто для встановлення вручну відповідності між домен проблеми та домен розв'язання). Це, у свою чергу, часто є помилковим завданням сильно залежить від інтерпретації розробника (і, іноді, неправильної інтерпретації) цих вимог.

Моделювання мови є надто простим або занадто складним. Друге дослідження проблеми полягає в тому, що більшість мов моделювання намагаються абстрагувати дизайнера від низького рівня деталі реалізації – такі як *WebML* або *XIS2* – зазвичай недостатньо виразні, щоб визначити повнофункціональний додаток, і навпаки. Хоча це і зрозуміло (оскільки модель повинна бути абстракцією реальності), це врешті-решт призводить до того, що розробники вручну редагують вихідний код, що є практикою *MDE* також призначений для уникнення. З іншого боку, підходи, що забезпечують дуже виразність мова моделювання – така як мова *OutSystems Agile Platform* – швидше за все, не дасть реальної абстракції щодо деталей реалізації (що є одним із наріжних каменів *MDE*), але лише інший конкретний синтаксис для однакові поняття мови програмування.

Ця проблема пов'язана з попередньою, оскільки ці мови, як правило, є результатом спроби сконцентрувати занадто багато деталей різних рівнів абстракції таких як моделі високого рівня та деталі реалізації – однією мовою. Цей компроміс, в свою чергу, робить мову або дуже складною, але здатною

моделювати найбільш передбачувані програми, або відносно простий, але не може моделювати реальні програми.

Вихідний код все ще є головним артефактом. Третя проблема дослідження полягає в тому, що основним артефактом, що є результатом сучасних підходів до розробки, як і раніше залишається вихідний код вказаний певною мовою програмування (оскільки саме це компілюється), а не моделі. Хоча самі мови програмування можна розглядати як моделі мов, вони, однак, все ще орієнтовані щодо того, як комп'ютер повинен вирішувати проблему, замість того, яку проблему вирішувати. Крім того, хоча більшість підходів вважають, що розробники можуть безпосередньо редагувати породжених артефактів низького рівня, однією з головних цілей *MDE* є саме уникнення цього такий вид редагування, оскільки це призвело б до того, що розробникам все одно доведеться мати справу з проблемами низького рівня. Крім того, якщо процес генерації вихідного коду був запущений знову, то, швидше за все, файл ручні зміни, внесені у вихідний код, будуть втрачені.

Знову ж таки, ця проблема пов'язана з попередньою, оскільки відсутність виразності в більшості мов моделювання впливає з того, що їх творці вже припустимо, що моделювання відбуватиметься лише як проміжна діяльність у програмі процес розвитку, замість того, щоб бути діяльністю з виробництва головного артефакту, який буде бути розгорнутим у виробничому середовищі. Це ще одне питання, в якому підхід *OutSystems* вирізняється. Хоча це так розглянемо можливість генерації та компіляції вихідного коду, ці кроки виконуються «за лаштунками», і розробник не може втручатися в процес. Це, в свою чергу, робить це обов'язковим мова моделювання повинна бути достатньо виразною, щоб задовольнити потреби низького рівня розробники щодо розробки більшості видів вебдодатків.

1.2. Основні аспекти модельованих підходів

Цей аналіз в основному зосереджений на моделюванні, що використовується підходом та стосується набору проблем моделювання, таких як моделювання доменів, бізнес логічне моделювання та моделювання інтерфейсу користувача. Тим не менше, проаналізуємо деякі аспекти щодо генерації частин вебпрограми, таких як використання моделей до перетворення моделі або розглянуте середовище розгортання.

1. Моделювання доменів. Моделювання доменів стосується ідентифікації проблем-доменної концепції та їх подання з використанням мови моделювання (наприклад, діаграми *UML*). Цей аспект аналізується щодо: чи підтримується він мовою і чи це не залежить від постійності та деталей користувацького інтерфейсу (тобто домену моделі не потрібно «налаштовувати», щоб підтримувати ці шари).

2. Моделювання бізнес-логіки. Хоча визначення моделювання бізнес-логіки можна в цій роботі вважати як специфікацію поведінки вебпрограми. Цей аспект аналізується щодо наступних підпунктів: чи підтримує він запити та маніпулювання поняттями домену (а саме використання шаблонів); чи є така підтримка запитів та маніпуляцій низьким рівнем схожих на традиційний вихідний код; та підтримка специфікації процесу. Повинен зазначити, що тема підтримки низького рівня вважається актуальною, оскільки вона часто відображає ситуацію виразності мови: типові мови, орієнтовані на вихідний код (такі як *C* або *Java*), хоч і складні, проте дуже виразні.

3. Моделювання навігаційного потоку. Підтримка підходу для визначення навігації потік (у контексті змодельованої вебпрограми) між різними сторінками *HTML*, або навіть всередині *HTML*-сторінок, це також аспект, що аналізується.

4. Моделювання інтерфейсу користувача. Іншим важливим аспектом є підтримка підходу для моделювання інтерфейсу користувача (*UI*). Проаналізовані такі суб'єкти: мова моделювання інтерфейсу користувача не

залежить від платформи (тобто не вимагає спеціального програмного забезпечення щоб представити інтерфейс користувача); підтримує специфікацію контролю доступу (тобто, показано деякі елементи керування або прихований відповідно до автентифікованого користувача); дозволяє визначити користувацький інтерфейс елементи; дозволяє використовувати шаблони взаємодії (наприклад, створювати, редагувати або асоціювати та дисоціювати); та підтримує зв'язування між елементами інтерфейсу та елементами доменної моделі.

5. Перетворення від моделі до моделі. Цей аспект аналізує, чи підхід використовує (або навіть розглядає) використання перетворень від моделі до моделі. Цей вид перетворення зазвичай використовується для прискорення завдання проектування вебпрограми, за допомогою аналізу моделі та механізмів умовиводу для автоматичного визначення деяких частин модель вебдодатку, тим самим звільняючи дизайнер моделей від деяких повторюваних (і завдання, схильні до помилок).

6. Сформована заявка заповнена. Цей аспект визначає, чи підхід інструментальна підтримка може повністю генерувати додаток (тобто воно не вимагає ручної реалізації певних функцій за допомогою програмістів).

7. Незалежні від середовища розгортання. Нарешті, цей аспект вивчає цільова платформа, що враховується підходом, а саме, чи існує щільне зчеплення між підходом та цільовою платформою.

На рис. 1.1 наведено простий огляд ментальної карти аналізованих аспектів.

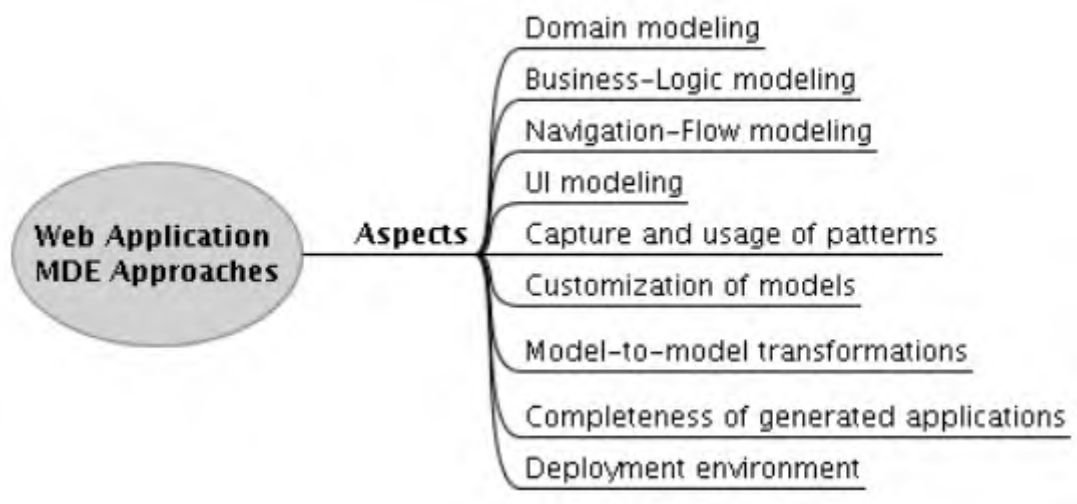


Рис. 1.1. Огляд аспектів та проблем, що стосуються моделювання вебдодатків

1.3. Критерії аспектів моделювання вебдодатків

1. Моделювання доменів. Моделювання доменів є важливим аспектом будь-якої програми підхід до розвитку. Проаналізовані підходи забезпечують незалежність між доменом модель та інші частини дизайну програми, але лише адресний домен *WebML* та *XIS2* моделювання таким чином, що повністю залежить від решти деталей програми, не вимагаючи, щоб модель домену коригувалась за допомогою інтерфейсу користувача або орієнтована на постійність деталі. У випадку з *UWE* модель вмісту повинна бути дещо орієнтована на презентаційна модель (наприклад, у прикладі адресної книги в підручнику *UWE 5*, *AddressBook* клас має атрибут введення, що використовується для головного екрану програми показати невелике вступне повідомлення користувачеві). На платформі *Agile* домен модель змодельована як схема бази даних, хоча з різними термінами (сутності та атрибути замість таблиць і стовпців); хоча ця відсутність абстракції може бути розглядається як погана річ, на практиці вона наділяє дизайнера більшим рівнем контроль реальної схеми бази даних вебпрограми. *WebML* також має спосіб пристосовувати модель домену до конкретних потреб інших рівнів за допомогою

виведення *Model*, а *XIS2* надає *Business Entities View* для вирішення проблем маніпуляцій сутність домену іншими рівнями.

2. Моделювання бізнес-логіки. Моделювання бізнес-логіки розглядається усіма аналізували підходи, хоча і дуже різними способами. *XIS2* розглядає цей аспект засобами суб'єктів господарювання – дозволяючи іншим рівням маніпулювати елементами, які є більш грубими, ніж елементи домену і шаблони (створення / читання / оновлення / видалення операції підтримуються, і дизайнер може додати більше операцій, хоча вони повинні впроваджуватись вручну). *WebML* використовує подібні механізми (*Derivation Model* для пристосування моделі домену до інших рівнів, одиниць вмісту та роботи блоків для запиту та маніпулювання доменом, і конструктор може додати нову операцію Блоки, які повинні бути реалізовані вручну), але дизайнер може організувати їх у подібний до блок-схеми, що дозволяє визначити робочі процеси маніпулювання даними. *UWE*, однак, розглядає цей аспект лише за допомогою діаграми класу *Process Model*, яка представляє взаємозв'язки між різними процесами та діаграмами діяльності, які визначають кроки кожного процесу і обмеження *OCL*; сама реалізація повинна робити вручну. *Sketchflow*, хоча і інструмент для створення прототипів, дозволяє дизайнерам наповнити їх прототипи з деякою поведінкою, за допомогою механізму поведінки; це також може бути розширено за допомогою додаткових моделей поведінки, але створення такої функціональності передбачає певну розробку вручну. Нарешті, *Agile Platform* дозволяє дизайнерам вказати складну бізнес-логіку графічно: шляхом визначення таких дій, як запит домену і шаблони маніпуляцій, дизайнери можуть вказати більшість (якщо не всі) функціональні можливості, які, як правило, доводиться кодувати вручну.

3. Моделювання навігаційного потоку. Моделювання навігаційного потоку розглядається усіма аналізували підходи, оскільки він є фундаментальним для будь-якого виду вебдодатків. На рахунок їхнього характеру моделювання вебдодатків та всього, що це означає (наприклад, існування *HTML*-сторінки, гіперпосилання для навігації між сторінками, використовуючи

параметри запиту або подібний механізм надання необхідних параметрів), всі вони дотримуються однакових вказівок (хоча терміни, що використовуються, дещо відрізняються): малюється спрямований графік, де відповідають вузли до *HTML*-сторінок, а краї відповідають можливим доступним навігаційним потокам користувачеві в контексті певної сторінки. Тим не менше, аналізовані підходи роблять відрізняються тим, чи може дизайнер вказати набори ребер (тобто дії чи посилання) у кожному вузлі. У *WebML* кожен вузол *DataUnit* має чітко визначений набір посилань для введення та виводу, а дизайнер не може вказати додаткові посилання. З іншого боку, *XIS2* дозволяє дизайнеру для визначення дій, пов'язаних з елементами інтерфейсу на сторінці, та навігації згодом потоки будуть пов'язані з тими діями сторінки. Підхід *XIS2* до цього аспекту дуже схоже на те, що можна знайти в *Agile Platform*, *Sketchflow* та *UWE*.

4. Моделювання інтерфейсу користувача. За винятком *WebML*, усі аналізовані підходи адресовані моделюванню інтерфейсу користувача графічним способом *WYSIWYG* (що ти бачиш, те і отримуєш). *WebML* дозволяє лише вказати, які елементи будуть присутні на сторінці, але не там, де вони будуть розташовані. Також підтримка *Agile Platform*, *WebML* та *Sketchflow* створення нових елементів сторінки або інтерфейсу для повторного використання на інших екранах програм. Це дозволяє дизайнерам лише один раз вказати певні розділи екрана та імпортувати ці розділи на деякі / всі екрани програми, з очевидною додатковою перевагою, яка змінюється для такого розділу потрібно робити лише одну точку в моделі; хороший приклад таким компонентом буде банер вебсайту або дерево навігації вебсайтом. Аспект поведінки чітко розглядається *XIS2*, *WebML* та *Agile Platform* завдяки захопленню шаблонів інтерфейсу, що дозволяє програмам взаємодіяти з користувачами, а не просто відображення запитаних ресурсів. Цей фокус на захопленні шаблонів, орієнтованих на інтерфейс, є також присутній у *Sketchflow*, у формі поведінки. *Sketchflow* сам по собі є єдиним інструментом, аналізованих, що підтримує додавання нових моделей взаємодії багаторазовим способом.

Крім того, специфікація контролю доступу (тобто, які користувачі чи ролі можуть отримати доступ до якого інтерфейсу компоненти) підтримується у всіх підходах, крім *UWE* та *Sketchflow*. Хоча в ескізний цей недолік зрозумілий (враховуючи його природу як прототипування) інструмент, в *UWE* це перешкоджає його адекватності моделюванню реальних сценаріїв. Всі проаналізовані підходи дозволяють прив'язувати елементи інтерфейсу користувача до елементів домену (наприклад, налаштувати рядки таблиці для відображення значень поля в конкретних випадках). Однак лише *XIS2* і *Agile Platform* дозволяють налаштувати ці прив'язки в моделі (наприклад, змінити клітинку в рядку таблиці, щоб показати різне значення для кожного екземпляра). Хоча ні можливість налаштування цих прив'язок може спростити модель та уникнути помилок дизайнера, на практиці це також обмеження мови, що змушує розробників змінюватися згенерований вихідний код для задоволення конкретних вимог.

5. Перетворення від моделі до моделі. Підтримка перетворень від моделі до моделі це аспект, який, як правило, не розглядається підходами до моделювання вебдодатків; від аналізованих підходи, лише основані на *UML* (*UWE* та *XIS2*) стосуються цього аспекту.

6. Сформована заявка заповнена. З аналізованих підходів лише *Agile* платформа вважає, що вихідний код не слід редагувати вручну – лише саму модель можна редагувати, а згенерований код та бази даних завжди залишаються поза межами досягнення розробника. Усі інші підходи розглядають традиційне програмування (тобто ручне редагування згенерованих артефактів вихідного коду) як діяльність, що відбуватиметься протягом життя розробки циклу, щоб врахувати конкретні вимоги, які не можуть бути виражені в мова моделювання підходу.

7. Незалежність від середовища розгортання. За винятком платформи *Agile*, всі підходи насправді не залежать від середовища розгортання. Тому що їх використання елементів високого рівня, *WebML*, *UWE* та *XIS2* може генерувати код для будь-якої веб-орієнтованої платформа (наприклад, *ASP.NET*, *Apache*

Tomcat); *XIS2* також може генерувати код для настільні платформи. Інтернет-версія *Sketchflow* генерує *Silverlight* програма, яка хоч і потребує встановленого у веб-браузері плагіна *Silverlight*, може розглядається як міжплатформна, оскільки вона доступна для *Microsoft Windows*, *Linux*, та платформи *Mac OS X*. Нарешті, моделі, створені на платформі *OutSystems Agile* можна розгорнути лише на стеках розгортання *OutSystems*, які використовують або *JBoss* сервер додатків та базу даних *Oracle*, або сервер додатків *IIS* від *Microsoft* та *SQL* база даних *Server Express*. Хоча в цьому аспекті *Agile Platform*, мабуть, більше обмежений, ніж інші підходи, саме це дозволяє йому автоматизувати більшу частину Інтернету життєвий цикл розробки додатків, а саме розгортання та оновлення / відкат додатків сценарії.

1.4. Висновки до розділу

У цьому розділі було представлено аналіз вибраного набору вебдодатків, орієнтованих на моделювання підходів та мов. Цей аналіз, у свою чергу, був зосереджений насамперед на аспекти, що мають відношення до такого роду моделювання.

З цього аналізу ми також виділили низку проблем, а також деякі міркування, які слід враховувати, коли пропонуючи рішення.

Сьогодні більшість вебдодатків все ще розробляються вручну (тобто, за допомогою типових завдань програмування), концепція розробки вебдодатків в модельний стиль швидко зростає у популярності. Приклади цієї тенденції зростання можуть можна знайти у кількості мов моделювання вебдодатків, таких як проаналізовано в цьому розділі.

РОЗДІЛ 2

СИСТЕМИ УПРАВЛІННЯ ВМІСТОМ

2.1. Види CMS-систем

Хоча ідея управління контентом існувала з самого початку Інтернету, лише в останні роки ми почали спостерігати вибух CMS системи. Система управління вмістом (CMS) є особливим видом вебдодаток, орієнтований на управління та публікацію вмісту (який може бути майже будь-яким, наприклад, повідомленням у блозі, записом на форумі, деяким текстом *HTML* або відео). Ці системи зазвичай надають адміністраторів та споживачів вмісту (тобто звичайних користувачів), що просто переглядають вебсайт) із набором відповідних аспектів таких як модульність, залежність між вмістом та його поданням, управління доступом, управління користувачем, або настроюваний візуальний вигляд та макет вмісту. Крім того, розвиток CMS-вебдодатків на базі є темою , яка тільки недавно була запропонована, як і більшість CMS до недавнього часу системи не надавали розробникам належного набору функцій.

Проаналізуємо такі системи управління вмістом: *DotNetNuke*; *Drupal*; *Joomla*; *Vignette Content Management*; *WebComfort*.

Хоча доступно постійно зростаюча кількість систем управління вмістом, ці системи були відібрані для аналізу через значні відмінності, які вони демонструють серед них. Інші системи управління вмістом (наприклад, *WordPress*, дуже популярна система ведення блогів, яка є тим не менш розглядається спільнотою як система CMS), хоча настільки ж придатна для аналіз, як раніше згадані, дав би дуже подібні результати - наприклад, аналіз *WordPress* призведе до майже тих самих значень, що і *Joomla*, роблячи їх включення до цього аналізу зайвих.

2.2. Системи управління вмістом з відкритим кодом

У цьому пункті я представлю мій аналіз цих систем управління вмістом. Почну із введення критеріїв аналізу, а потім перейдемо до отриманих результатів отримані в результаті цього аналізу.

DotNetNuke. *DotNetNuke* – це система управління вмістом з відкритим кодом, що має широке спільноту користувачів, на базі *Microsoft ASP.NET2* технології. Вона написана на *Visual Basic .NET (VB.NET)*, і вимагає *Microsoft Internet Information Services (IIS)* та *Microsoft SQL Server* для запуску.

CMS DotNetNuke надає набір нестандартних функцій, які можна знайти в більшості систем управління вмістом (наприклад, форуми, блоги, управління користувачами та ролями), що дозволяють нетехнічним користувачам швидко створювати та налаштовувати вебсайти. Він також підтримує багатокористувацьку оренду як користувачі можуть визначити кілька сайтів в межах однієї інсталяції.

Система представляє сторінковий підхід до управління вебсайтом, як користувачі роблять вміст доступним, визначаючи вкладки та модулі в контексті які вони потім можуть визначити вміст, який повинен відображатися.

DotNetNuke також надає засоби для його розширення та налаштування, а саме через сторонні скіни, модулі та провайдери, що забезпечують додаткові спеціальні функції. Адміністратор установки *DotNetNuke* також може завантажувати та встановлювати додаткові модулі через адміністративні сторінки *DotNetNuke*; після установки модуль доступний на будь-якій сторінці вебсайту. Скін складається з простих *HTML*-файлів (із заповнювачами для вмісту, меню та іншої функції), а також файли підтримки (наприклад, зображення та таблиці стилів), упаковані у *zip*-файл.

Як і більшість систем управління вмістом, *DotNetNuke* не передбачає жодного конкретного підходу (на основі моделі або іншим чином) для розробки модулів. Розробка модулів здійснюється через традиційне програмування. Це

передбачає розробку вихідного коду (у *VB.NET* або *C#*) використовує функції, надані фреймворком *DotNetNuke* та *API*.

Drupal. *Drupal* – це широко використовувана *CMS* із відкритим кодом, написана на *PHP* і здатна працювати на будь-якому вебсервері, який може запускати *PHP*, наприклад *Apache* або *Microsoft IIS*. Що стосується стійкості, він надає повну підтримку баз даних *MySQL* та *PostgreSQL*.

Стандартний випуск *Drupal* відомий як *Drupal core* і пропонує функції, які зазвичай можна знайти в системах управління вмістом (наприклад, можливість налаштування макета сторінки, доступні канали синдикації, блоги, форуми). *Drupal* також підтримує багатокористувацьку оренду (тобто можливість визначення декількох сайтів за допомогою однієї інсталяції).

На відміну від *DotNetNuke*, *Drupal* дотримується контент-орієнтованого підходу до управління вебсайтами. Хоча адміністратори можуть налаштувати деякі частини структури вебсайту, основний акцент робиться на виробництві вмісту; будь-які сторінки, визначені адміністратором будуть лише засобом, за допомогою якого користувачі можуть отримати доступ до вмісту (хоча існують і інші, а саме механізм меню). Крім того, *Drupal core* може бути розширено за допомогою додаткових функцій та поведінки засоби модулів плагінів, надані членами спільноти користувачів. Такі модулі будуть відповідати за вирішення різних питань, таких як інтерпретація *URL*-адрес визначити, який вміст показувати. *Drupal* не надає жодного конкретного підходу до розробки модулів плагінів. Розробка таких модулів здійснюється за допомогою традиційного програмування, що вимагає знання самого *Drupal*, мови програмування *PHP* та *API* модулів *Drupal*.

Joomla. *Joomla* – це ще одна *CMS* із відкритим кодом, написана на *PHP* і здатна працювати на вебсерверах *Apache* або *Microsoft IIS*. На відміну від інших систем управління вмістом, стандартний випуск *Joomla* не забезпечує більшості типові функції *CMS* (наприклад, можливість налаштування макета сторінки, доступна синдикація канали, блоги, форуми) нестандартні, хоча такі функції доступні як безкоштовні доповнення. Крім того, сама *Joomla* ще не підтримує

багатокористувацьку оренду, але деякі безкоштовні доповнення підтримують та забезпечують цю особливість. Манера, дуже схожа на *Drupal*, *Joomla CMS* відповідає змісту підходу до управління вебсайтами. Адміністратори можуть налаштувати деякі частини структури вебсайту, але тим не менше основний акцент робиться на виробництві вмісту.

Joomla можна розширити додатковими функціями та поведінкою за допомогою плагіна модулі, які вносять члени спільноти (насправді, більшість основних *CMS* функцій, надані *Joomla*, отримуються за допомогою таких модулів, які необхідно встановити адміністратором). Як і *Drupal*, *Joomla* не розглядає жодного конкретного підходу до розвитку модулі плагінів: розробка модулів *Joomla* здійснюється за допомогою традиційного програмування за допомогою мови програмування *PHP* та *API Joomla*.

Vignette. *Vignette*, розроблена корпорацією *Vignette*, є широко використовуваним комерційним пакетом програмне забезпечення для управління вмістом, порталом, співпрацею, документами та записами інструменти. Він підтримує платформи *JavaEE* та *Microsoft.NET* і працює на *IBM DB2* базі даних. *Vignette* також підтримує багатоквартирне житло. Для моєї роботи особливе значення мають дві її програми *Content Management* та *Portal*. Додаток *Content Management* – це, по суті, бек-офіс дозволяє нетехнічним користувачам створювати, редагувати та відстежувати вміст у робочих процесах. З іншого боку, програма *Portal* дозволяє публікувати цей вміст у *Vignette* вебсайт. Хоча філософія *Vignette* полягає у тому, щоб завжди концентруватись на вмісті, а не на вебсайті структура, його підхід до управління вмістом насправді є сумішшю контент-орієнтованих та сторіноцентрична. Вміст створюється та редагується користувачами (іноді за допомогою робочих процесів), але публікується за допомогою механізму на основі шаблону, який отримує набір вмісту та повертає вебсайт, дозволяючи користувачам отримати доступ до цього вмісту. *Vignette* не розглядає жоден особливий підхід до розробки розширень: розробка здійснюється за допомогою традиційних програмування за допомогою мови програмування *Java* та *API Vignette*.

WebComfort. *WebComfort* – це система управління вебвмістом та програмами. Вона базується на основі технології *Microsoft ASP.NET* і може працювати на будь-якому сервері *Microsoft IIS*. Що стосується стійкості, він надає повну підтримку для баз даних *Microsoft SQL Server*, *PostgreSQL* та *Oracle*. *WebComfort* не підтримує багатокористувацьку оренду. Тим не менше, він не відмовився від своїх коренів системи управління вмістом, а також пропонує нестандартну підтримку різноманітних функцій, які можна знайти в типових системах управління вмістом (наприклад, оголошення, управління посиланнями, управління користувачами та ролями). *WebComfort* використовує спосіб, подібний до *DotNetNuke* та багатьох інших систем управління вмістом орієнтованим на сторінки підхід до управління вебсайтами. Адміністратори та користувачі мають визначення доступна структура вебсайту, що складається з вкладок та модулів, а також вміст в контексті цих модулів.

Як і *DotNetNuke*, *WebComfort* не передбачає жодного конкретного підходу до розробки модулів або наборів інструментів. Хоча *WebComfort* і надає певну підтримку для їх розробки та розгортання (а саме через *API WebComfort* та *Toolkits* механізм), ці завдання виконуються за допомогою традиційного програмування та ручної обробки за допомогою мов програмування *C#* або *VB.NET*

2.3. Переваги та недоліки CMS-систем

Цей аналіз в основному зосереджений на характеристиках, отриманих з нашого власного досвіду з використанням цих систем під час нашого дослідження та знань, набутих за час розробки *WebComfort CMS* та деяких вебдодатків, які підтримуються ним, наприклад *WebC-Docs*. Більш конкретно, використані критерії охоплюють аспекти, що варіюються від адміністративних можливостей (наприклад, конфігурація вебсайту структура або підтримка багатоарендності) до орієнтованих на розробника функцій, таких як

розширюваність або надання конкретного підходу до розробки вебдодатків на базі *CMS*.

1. Підхід управління. Системи управління вмістом зазвичай використовують одне з двох систем управління підходи: орієнтована на сторінку та контент-орієнтована. Сторінково-орієнтований підхід вважає, що повинна бути визначена структура вебсайту (тобто набір сторінок та компонентів) поперше, а потім зміст визначається в контексті цієї структури. З іншого боку, контентно-орієнтований підхід диктує, що сам контент повинен бути визначений, і згодом адміністратор може вказати структуру сторінок, яка відображатиме щось із цього змісту. Цей аспект аналізує, який із цих підходів до управління використовується *CMS*.

2. Настроювана структура вебсайту. Цей аспект визначає, чи буде система управління вмістом система дозволяє своїм адміністраторам користувачам таким чином налаштовувати структуру вебсайту широкий перелік існуючих систем управління вмістом доступний за адресами що ефективно дозволяє відвідувачам сприймати організацію вебсайту як (структуровану) ієрархічний набір сторінок. Хоча цей аспект може здатися неактуальним (оскільки, схоже, він надає перевагу системи управління вмістом із сторінково-орієнтованим підходом управління), він фактично використовується для позначення чи підтримує *CMS* специфікацію структури вебсайту; ця структура, в свою чергу, часто є основним для допомоги відвідувачам під час навігації вебсайтом та доступу до опублікованих зміст.

3. Настроюваний візуальний макет сторінки. Окрім можливості налаштування структуру вебсайту, адміністратори також повинні мати можливість налаштувати візуальний дизайн вебсайту макет (тобто зовнішній вигляд вебсайту, наприклад, використовувані кольори або відносне розташування кожен контейнер, який сторінки використовуватимуть для показу вмісту). Цей аспект визначає, чи *CMS* підтримує будь-який такий візуальний механізм.

4. Підтримка мультидоступності. Багаторічна оренда складається з того, чи можливо щоб створити логічний набір вебсайтів в рамках тієї ж інсталяції *CMS*.

Іншими словами, цей аспект визначає, чи може одна фізична інсталяція *CMS* підтримувати визначення декількох логічних вебсайтів (наприклад, персональний вебсайт та вебсайт електронної комерції), кожна з яких зазвичай доступна за іншою *URL*-адресою (*Universal Resource Locator*).

5. Кілька видів наполегливості. *CMS* система, за його динамічний характер, потрібно зберігати (десь зберігати) десь свою інформацію (наприклад, базу даних або навіть файлову систему). Хоча на перший погляд цей аспект може здатися лише деталлю, пов'язаною з технологіями, він є важливо відзначити, що важливість цього аспекту впливає з роз'єднання вебдодатків логіка і механізм наполегливості, в свою чергу, вводить уявлення про те, що моделювання доменів мовою, орієнтованою на *CMS*, не повинно залежати від або навіть, припустимо, специфічні для технології деталі (наприклад, подання бази даних).

6. Може бути продовжено третіми сторонами. Цей аспект уважно вивчає механізми, які надаються системою *CMS* для її розширення сторонніми сторонами (наприклад, чи *CMS* надає *API* для розробки нових функцій). Цей аналіз особливо зосереджений на наступних пунктах: які мови (мови програмування чи інші) можна використовувати; чи можна використовувати обмежувальні функції, надані системою управління вмістом можливі дії (або показати додаткову інформацію); чи змінюється система управління вмістом дозволена поведінка за замовчуванням; та якщо можливо додати нову поведінку (наприклад, нову *CMS* компоненти або додатковий код для запуску, коли в системі відбуваються певні події).

7. Підхід до розвитку. Цей заключний аспект аналізує тип підходу (якщо такий є), що *CMS* підтримує розробку вебдодатків на її основі (навіть якщо вебдодаток складається лише з налаштувань, розширень або будь-чого, що змінює поведінка системи за замовчуванням). Більш конкретно, я визначу: якщо *CMS* все-таки враховує будь-який конкретний підхід до розробки таких вебдодатків; і якщо це так розглянемо такий підхід, незалежно від того, чи він базується на моделях, чи якщо він застосовується у більшій кількості традиційний спосіб керування вихідним кодом.

Проаналізовано деякі системи управління вмістом відповідно до аспектів, які, як правило, мають значення при розробці розширення *CMS*, налаштування або навіть вебдодатки на базі *CMS* зі значним ступінь складності. На рисунку 3.2 наведено простий огляд аналізованої ментальної карти аспекти.

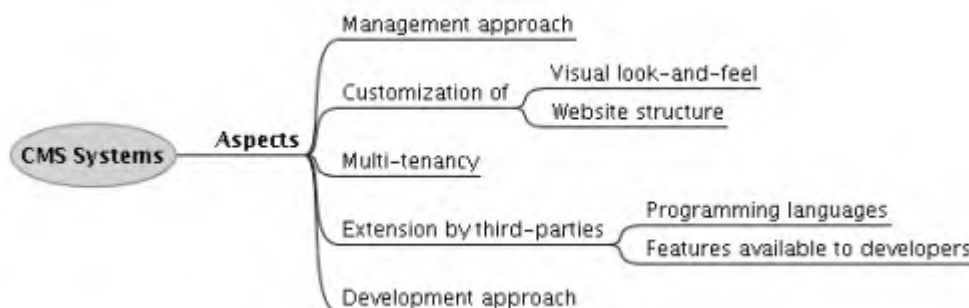


Рис. 1.2. Огляд аспектів та проблем щодо систем управління вмістом.

2.4. Доцільність використання *CMS*-систем при розробці вебдодатків

1. Підхід управління. Щодо управлінського підходу, який використовується аналізуючи системи управління вмістом, ми помічаємо, що обидва підходи (орієнтовані на сторінку та контент-орієнтовані) використовуються. Більш конкретно, системи *DotNetNuke* та *WebComfort* використовують орієнтовану на сторінку сторінку підхід, при якому визначається структура вебсайту (тобто набір вкладок та модулів) перший, а потім зміст задається в модулях. Тим не менше, ці дві системи управління вмістом підтримувати обмін вмістом між модулями за допомогою функції копіювання модулів *DotNetNuke* та копію модулів і довідкові функції модулів *WebComfort*. З іншого боку, системи *Drupal* та *Joomla* використовують контентно-орієнтований підхід, при якому адміністратор спочатку визначає вміст, який буде відображатися користувачеві, а потім визначає структуру сторінки, які відображатимуть певні частини доступного вмісту. Сама система *Vignette* може можна розглядати як поєднання цих двох підходів, оскільки зміст визначається незалежно і представлений користувачеві за допомогою механізму

шаблонів Vignette, який використовує а заздалегідь визначена структура вебсайту для представлення наявного вмісту. Тим не менше, ми розглядаємо Vignette як головним чином змістовна, завдяки тому, що вона робить більший акцент на визначення вмісту, а не на визначення структури вебсайту.

2. Настроювана структура вебсайту. Усі аналізовані системи управління вмістом дозволяють міністрів для налаштування структури вебсайту як ієрархічного набору сторінок або вузлів (у поєднанні з механізмами зв'язку, такими як меню *Joomla*), залежно від підхід управління, що використовується *CMS*.

3. Настроюваний візуальний макет сторінки. Можливість налаштування вебсайту візуальне розміщення (тобто зовнішній вигляд вебсайту, наприклад, використовувані кольори або відносна інформація) розташування кожного контейнера, який сторінки використовуватимуть для показа вмісту) підтримується всіма аналізували системи управління вмістом.

4. Підтримує багатоорендність. Деякі з проаналізованих підтримують багатокористувацьку оренду системи управління вмістом. Більш конкретно, лише *Joomla* та *WebComfort* не мають підтримки мульти-орендної плата, хоча в обох випадках це аспект, над яким зараз розробники працюють. Тим не менше, ми спостерігали, що системи управління вмістом, які підтримують це функцію зазвичай робить це, визначаючи інший префікс таблиці бази даних для кожного вебсайта, який означає, що різні логічні вебсайти часто повністю незалежні один від одного.

5. Декілька видів наполегливості. Щодо використовуваних механізмів стійкості, лише деякі системи управління вмістом (а саме *Drupal* та *WebComfort*) підтримують різні типи механізми стійкості. Більш конкретно, ця функція поставляється у формі підтримки для різні види СУБД (Система управління базами даних), такі як *MySQL*, *PostgreSQL*, або *Microsoft SQL Server*.

6. Може бути продовжено третіми сторонами. Усі ці сисмтеми управління вмістом дозволяють розширення сторонніми розробниками. Аспекти, які можуть бути використані або розширені розробниками, різняться

між системами, хоча деякі є загальними, наприклад, функції захисту (користувач та роль управління). Однак, як і підтримувані механізми стійкості, конкретні деталі (наприклад, використовувані мови програмування) різняться залежно від різних систем управління вмістом системи, що ще більше свідчить про те, що орієнтовані на *CMS* мови повинні бути такими, як незалежно від технологій. Крім того, усі ці системи підтримують додавання функцій і поведінку, але лише деякі підтримують зміну існуючої поведінки за замовчуванням; в останньому у цьому випадку це зазвичай робиться за допомогою шаблону розробки стратегії (або його варіанту цього) та використання вбудованої поведінки за замовчуванням, лише якщо інша стратегія недоступна.

7. Підхід до розвитку. Жодна з аналізованих систем управління вмістом не враховує підхід для налаштування або розробки розширень. Хоча, як уже згадувалося в попередньому параграфі, кожна з цих систем надає певну підтримку розробникам у різних формах, але зазвичай складається з *API* для розробки вихідного коду – підходу до розробки саме це залишається для розробників визначати спеціально.

Окрім систем управління вмістом, проаналізованих у цьому розділі, ми також знайшли деякі останні роботи щодо підходів до розробки вебдодатків на базі *CMS*. Робота визначає загальну метамодель *CMS*, яку можна застосувати для більшості систем управління вмістом, які доступні сьогодні (включаючи аналізовані у цьому розділі). Хоча ця метамодель сильно зосереджена на структурі *CMS*, я повторно використаю частини цієї метамоделі у власній дипломній роботі. Програма *Integration Generator* отримує файл *XML* (що, в свою чергу, призводить до результатів від перетворення *XSLT* файлу моделі *UWE*, зазначеного у форматі *XMI*), і становить відповідь за налаштування цільової системи управління вмістом, а саме шляхом взаємодії з резервне зберігання даних (наприклад, сервер баз даних) та створення необхідної підтримки артефакти (наприклад, сторінки *ASP.NET* та елементи керування користувача). Крім того, аналізує придатність об'єктно-орієнтованого вебрішення (*OOWS*) метод розробки *CMS* на основі вебдодатків, а також пропонує деякі

вдосконалення, отримані в результаті розробки ситуаційних методів. Однак підхід лише має справу з аспектами високого рівня, що призводить до відсутності виразності при спілкуванні деталі низького рівня (наприклад, зазначення конкретних класів *CSS*).

Нарешті, підхід *MDE* також спрямований на підтримку бізнес-користувачів. Це робиться шляхом визначення мови моделювання, яка стосується лише деталей високого рівня (на основі частини інших існуючих мов), а також механізм перетворення, який бере модель і генерує файл конфігурації *CMS* (файл *XML*, який буде інтерпретований *CMS*). Цей підхід також страждає від деяких недоліків інших підходів *MDE*, а саме відсутність виразності для роботи з деталями низького рівня; розробники можуть змінюватися згенерований файл конфігурації *CMS* за допомогою редактора *XML*, але цей файл несе лише інформація щодо модельованих концепцій високого рівня (наприклад, кроки певного бізнес-процесу), а не деталі, характерні для домену *CMS* (наприклад, роль, користувач, візуальна тема або модуль). Я вважаю, що це тому, що автори мають також намагалися стримуватися до єдиної мови, що в підсумку приносить типово компроміс між деталями низького рівня, які можна змодельовати, та простотою навчання та використання мову.

2.5. Висновки до розділу

Використання систем управління вмістом як базової платформи для визначення нових вебсайтів швидко зростає популярність. Приклади цього буму можна знайти в кількості *CMS* доступні системи та розширення кількості підтримуваних ними функцій, якими є неухильно збільшується.

РОЗДІЛ 3

МОДЕЛЬНИЙ ПІДХІД ПРИ РОЗРОБЦІ БАГАТОМОВНОЇ СИСТЕМА УПРАВЛІННЯ ВМІСТОМ

3.1. Підходи *MDE*

В даний час існує кілька підходів *MDE* для розробки вебдодатків, немає такого підходу для вебдодатків на базі *CMS*. Це зрозуміло, бо ідея використання систем управління вмістом як платформ для більш складних вебдодатків є досить справедливою останнім часом, але факт залишається фактом, що ці системи забезпечують певний ступінь розширюваності, який би бути вартим експлуатації. Більше того, більшість підходів *MDE* для розробки вебдодатків не передбачають адекватна підтримка різних перспектив, які мають зацікавлені сторони програми щодо цієї заяви. Ці перспективи часто фіксуються вручну (використовуючи такі засоби, як документи щодо вимог та зустрічі із зацікавленими сторонами), вручну інтерпретується командою розробників програми (включаючи архітекторів систем та домен експерти), а потім надано програмістам програми для впровадження в конкретна система. Однак більша частина цієї обробки виконується вручну через різні види зацікавлених сторін, які беруть участь.

Інша проблема сучасних підходів *MDE* полягає в тому, що вони: включають занадто багато деталей низького рівня в моді моделювання, намагаючись зробити її більш виразною; або намагатися включати якомога менше деталей низького рівня до мови, щоб полегшити її навчання та використання. Хоча це здавалося б очевидним (оскільки це було б дуже складно, якби ні неможливо визначити мову моделювання, яка була б достатньо виразною для потреб кожного зацікавленого учасника), він фактично виявляє наслідки компромісу, який повинен врешті-решт бути прийнятим через проблему, згадану в попередньому пункті.

Нарешті, кінцевою метою більшості сучасних підходів *MDE* все-таки є отримання вихідного коду, замість моделей. Це часто призводить до того, що розробники «обманюють», змінюючи створений вихідний код, тоді як сама модель залишається незмінною (і, отже, несинхронізованою з вихідний коду). Хоча існують методи, що дозволяють вручну змінювати вихідний код зберігаючи синхронізацію з моделлю, факт залишається фактом.

3.2. Багатомовний підхід

Для вирішення виявлених проблем запропоновано *MDE*-орієнтований підхід до розвитку *CMS* на основі вебдодатків. Цей підхід заслуговує на увагу відмінності від інших *MDE*-орієнтованих підходів до розробки вебдодатків, а саме: він базується на використанні декількох мод моделювання; і він використовує модель механізм синхронізації, щоб забезпечити узгодженість між моделями різних мов, і дозволяти зацікавленим сторонам одночасно змінювати різні типи моделей (що відповідають до різних точок зору бажаного вебдодатка), не турбуючись можлива втрата інформації. На рисунку 6.1 наведено спрощений огляд запропонованого підходу.

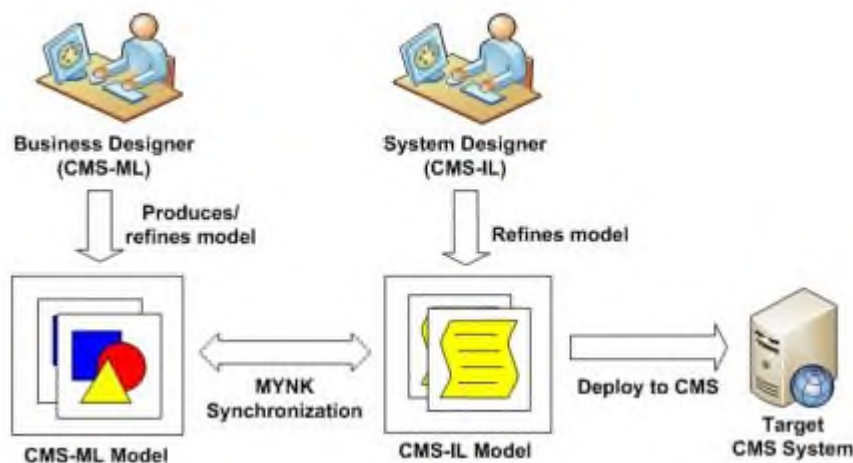


Рис. 3.1. Огляд запропонованого підходу до розвитку, орієнтованого на *MDE*.

Замість визначення єдиної орієнтованої на *CMS* мови моделювання, цей підхід визначає дві такі мови: *CMS-IL* (проміжна мова *CMS*), мова низького рівня,

що забезпечує спільну основу для специфікації вебдодатків на базі *CMS*; та *CMS-ML* (Модель моделювання *CMS*), що забезпечує набір елементів, які можуть використовуватись для визначення моделі вебдодатку високого рівня. На додаток до *CMS-ML* та мов *CMS-IL*, підхід також розглядає *MYNK* (Model sYNchronization frameworK), що підтримує модель механізму синхронізації. Попередньо слід зазначити, що мови *CMS-IL* та *CMS-ML* не мають на увазі бути достатнім для підтримки всіх зацікавлених сторін, оскільки така мета не була б практичною. Натомість, і для того, щоб підтвердити моє твердження у цій дипломній роботі, я зосереджусь на: дуже обмежений набір типів зацікавлених сторін, які я вважаю достатньо репрезентативними зацікавленими сторонами, які зазвичай беруть участь у подібних проектах з розробки програмного забезпечення.

Через це можливе обмеження щодо розглянутих типів зацікавлених сторін, ми також запропонувати короткий набір керівних принципів для створення нових модельних мов, які можуть бути використовується з цією моделлю механізму синхронізації. Ці рекомендації були використані у визначенні двох мов і можуть бути використані в визначення нових мод моделювання відповідно до потреб додаткових видів зацікавлених сторін.

3.2.1. Застосування графічного моделювання *CMS-ML*

CMS-ML – це мова графічного моделювання, що стосується специфікації *CMS* вебдодатки на високому рівні та незалежні від платформи. Він був розроблений дозволити нетехнічним зацікавленим сторонам швидко моделювати вебдодаток, що підтримується *CMS*, а саме, дозволяючи таким зацікавленим сторонам легко читати, розуміти та змінювати модель щоб воно точніше відображало їх наміри. Крім того, *CMS-ML* також має механізм розширення, спрямований на підтримку конкретних вимог зацікавленої сторони та моделювання вебдодатків з вищим ступенем складності.

Цільова аудиторія *CMS-ML* – це, як уже зазначалося, нетехнічні зацікавлені сторони, хто змоделює, яким має бути вебдодаток. Однак розширюваність мови механізм дійсно представляє деякі елементи моделювання, які більше орієнтовані на те, як вебпрограма повинна виконувати свою роль. Ці елементи можуть вимагати від зацікавлених сторін деяке технічне розуміння того, як повинен функціонувати вебдодаток; тим не менше, використання таких елементів є абсолютно необов’язковим.

3.2.2 Текстова мова *CMS-IL*

CMS-IL – це текстова мова низького рівня, з вищим рівнем виразності, ніж *CMS-ML*, що тим не менше не залежить від будь-якої конкретної платформи *CMS*. Ця мова забезпечує елементи структурного моделювання, але, на відміну від *CMS-ML*, воно також більше зосереджує увагу на тому, як вебдодаток має працювати. Таким чином, він визначає набір подібних до мови програмування елементи моделювання, що дозволяють специфікувати вебдодаток на базі *CMS* на рівень абстракції майже до впровадження.

Кінцевою метою *CMS-IL* є забезпечення загальної основи для специфікації з *CMS* на основі вебдодатків, в відміну від *CMS-ML*, яка покликана забезпечити спосіб виробництва прості моделі вебдодатків.

Цільовою аудиторією *CMS-IL* є технічні зацікавлені сторони (а саме розробники). Звичайно, ці зацікавлені сторони повинні добре знати концепції, які зазвичай можуть можна знайти в системах управління вмістом, таких як ті, що представлені в довідковій метамоделі системи управління вмістом визначається (наприклад, *DynamicWebPage*, *WebComponent*, *Visual Style* або *Workflow*).

3.2.3. Текстова мова *MYNK*

MYNK – це текстова мова, яка підтримує модель механізму синхронізації між Моделі *CMS-ML* та *CMS-IL*. Цей механізм, у свою чергу, є наріжним каменем. Це пояснюється тим, що бажане збільшення продуктивності і нижче кількість помилок моделювання залежить від використання автоматизованої синхронізації моделі s або перетворення моделі; альтернативою було б ручне перетворення *CMS-ML* моделі до *CMS-IL* (і навпаки), що призведе до зниження продуктивності та збільшення кількості помилок моделювання, допущених дизайнерами, через схильність до помилок характеру цього завдання.

Цю мову можна розглядати як модель запиту, орієнтовану на модель (хоча така опис не зовсім правильний). Він вільно заснований на аналізованій трансформації моделі механізми та мова *SQL*. З колишній, він успадковує концепцію встановлення відображень між розглянутим моделюванням мовами, тоді як остання є натхненням для створення модельних запитів, які можуть просто отримати елементи моделі або фактично змінити їх.

На відміну від *CMS-ML* та *CMS-IL*, мова *MYNK* не призначена для використання зацікавленими сторонами які беруть участь у розробці вебпрограми . Це тому, що ці зацікавлені сторони як правило, не потрібно знати про те, як модель синхронізації між *CMS-ML* і виконується *CMS-IL*. Натомість ця мова призначена для творців мов, які тоді можуть використовувати *MYNK* для розширення цього підходу за допомогою додаткових мов моделювання (орієнтованих на *CMS* або інакше). Можливою причиною визначення нових мод моделювання може бути підтримка додаткові види зацікавлених сторін.

3.2.4. Робочий процес розробки

Запропонований підхід також розглядає робочий процес, який ми розглядаємо як той, який буде використовуватися найчастіше. Цей робочий процес розглядає два основних типи зацікавлених сторін, яких ми називаємо бізнес-дизайнером та системою дизайнер.

Бізнес-дизайнер – загальний термін для ідентифікації нетехнічних зацікавлених сторін – може розпочати робочий процес, створивши модель *CMS-ML*, яка представляє передбачуваний вебзастосунок, відповідно до деяких задалегідь визначених бізнес-вимог. Після використання *MYNK* для отримання моделі *CMS-IL* (що відповідає вищезазначеному *CMS-ML*), системний конструктор який, на відміну від бізнес-дизайнера, є термін для ідентифікації технічних зацікавлених сторін – повинен визначати, чи є ця модель *CMS-IL* задовільним, а саме шляхом виявлення будь-яких конкретних вимог, які неможливо вирішити лише *CMS-ML*; якщо такі вимоги існують, отримана модель *CMS-IL* повинна бути додатково модифікований конструктором системи для їх вирішення. Поки *System Designer* змінює модель *CMS-IL*, механізм синхронізації *MYNK* повинен підтримувати моделі *CMS-IL* та *CMS-ML* (які відповідають одній і тій же цільовій вебпрограмі, розглядаються з точки зору різних зацікавлених сторін), що узгоджуються між собою.

Після процесу вдосконалення модель *CMS-IL* повинна бути точним поданням того, якою має бути передбачена вебпрограма. Однак, якщо бізнес-дизайнер це зробить не погоджуючись з моделлю *CMS-ML* (яка тим часом була автоматично оновлена, через *MYNK*, щоб відобразити зміни, внесені конструктором системи до відповідного моделі *CMS-IL*), тоді бізнес-дизайнеру доведеться змінити модель *CMS-ML*, активуючи черговий раунд механізму синхронізації *MYNK* ще раз.

Коли модель *CMS-IL* (і, відповідно, відповідна модель *CMS-ML*) є вважається задовільним, він розгортається на цільовій системі управління вмістом одним із двох способи, залежно від самої *CMS*: розгортання до інтерпретатора моделей *CMS* (або просто Інтерпретатор), який вже доступний в системі *CMS*, або покоління артефакти низького рівня та подальша установка.

Компонент *CMS Model Interpreter* – це компонент, який інстальовано на *CMS* система і несе відповідальність за отримання вхідної моделі *CMS-IL* та розгортання отримана модель, а саме шляхом налаштування системи управління

вмістом для відображення висловлених фактів в моделі. Цей компонент інтерпретатора створений на основі, який, хоча і ні орієнтована на системи управління вмістом або розробку вебдодатків, визначає час роботи на основі моделі (*MDR*) середовище, яке отримує модель (вказану за допомогою *UML*-класу та діаграм діяльності, анує *OCL* обмежень) і інтерпретує його під час виконання, що дозволяє ефективно дизайнерів запускати свої моделі.

Ми вважаємо, що перша альтернатива розгортанню - використання перекладача моделей *CMS* компонент – як правило, кращий, оскільки він вимагає лише адміністратора *CMS* (тобто, користувач з адміністративними привілеями) завантажує модель *CMS-IL* в інтерпретатор. Тим не менше, ця перша альтернатива не буде здійсненою на платформах *CMS*, які цього не роблять мати такий компонент у наявності; у таких випадках друга альтернатива вимагає втручання розробника програмного забезпечення – для компіляції згенеровані артефакти у форму, яка виконується *CMS* та *CMS* адміністратор, щоб як розгорнути зібрані артефакти, так і зробити все необхідне зміни конфігурації.

Хоча такий підхід не знімає необхідності в ітеративному процесі розробки, ми вважаємо, що це потенційно пом'якшує додаткову роботу з обробки невідповідностей

між перспективами зацікавлених сторін та їх розумінням розробником. Нарешті, слід зазначити, що ми не виключаємо можливий сценарій, за якого Бізнес-дизайнер визначає модель *CMS-ML*, а потім використовує її на компонент інтерпретатора моделі *CMS* цільової системи управління вмістом, минаючи модель доопрацювання конструктором системи (наприклад, для швидкого отримання прототипу бажаного вебдодаток). Однак це можна легко вирішити, дозволивши вищезазначене Компонент перекладача для отримання вхідної моделі *CMS-ML*, внутрішнього отримання відповідної моделі *CMS-IL* та розгортання отриманої моделі *CMS-ML*. Таким чином, ми не буде розглядати цей конкретний сценарій у цій дипломній роботі.

3.3. Вказівки щодо специфікації мови

Як уже зазначалося у цьому розділі, підхід, запропонований у моїй дипломній роботі розглядається лише дві мови, *CMS-ML* та *CMS-IL*. Однак це було б радше наївно просто припускати, що цих двох мов було б достатньо для підтримки будь-якого типу зацікавлених сторін у розробці реального вебдодатку. Це, в свою чергу, збільшує ймовірність випадків, коли підхід повинен бути розширений додатково орієнтованими на *CMS* моделювання мов, покликаних підтримати нові перспективи зацікавлених сторін. Тому стає необхідним визначити, як створити такі мови, і зробити їх придатними для використання з *MYNK*.

Слід зазначити, що деякі з цих рекомендацій базувались на визначенні мови *DSM* правила, тоді як інші ґрунтувались на попередніх досвідах з розробка мови *XIS2* та найкращих практик щодо *DSL* дизайн.

Визначені рекомендації пояснюються нижче (передбачається, звичайно, що дизайнер вже шукав *DSL*, що стосується бажаного проблемного домену, і не знайшов адекватних мов). Визначити цільову аудиторію для мови. На мій погляд, це один із найважливіші орієнтирів, оскільки його результати повинні бути головним фактором більшості мови дизайнерські рішення, які будуть наступними. Зокрема, важливо встановити:

1. Основні проблеми аудиторії щодо цілей моделювання мови;
2. Очікуваний рівень технічної експертизи аудиторії (враховуючи, що мета використання мови буде зрештою отримати модель, яка точно представляє програмне забезпечення для запуску в комп'ютері).

Хоча ця настанова, мабуть, здається очевидною для читачів, проте вона важлива, тому що кінцевою метою моделі є відповідь на запитання про систему, яку вона представляє. Отже, здоровий глузд диктує, що важливо визначитись спочатку на які питання потрібно відповісти, і лише після цього визначте спосіб відповіді на них питання.

Визначте проблемний домен, на який повинна бути адресована мова. Так само, як тант, як попереднє керівництво (і тісно пов'язане з ним), правильне та

точне визначення домену проблеми є важливим, щоб дизайнер мов міг знати, що слід моделювати, а те, що не має значення у моделі для бажаного вебдодатку. Правильна ідентифікація, у свою чергу, може допомогти мінімізувати істотну складність мовою, уникаючи включення непотрібних елементів, які, ймовірно лише сприятимуть ускладненню вивчення та використання мови. Крім того, ми є впевнений, що залежно від таких факторів, як суттєва складність проблемного домену та досвід дизайнера мов щодо цього домену, це правильна ідентифікація може допомогти запобігти випадковій складності мови взагалі.

Це ідентифікація, як правило, призводить до визначення анотації мови синтаксис (або формально з мовою метамодельовання, або просто неформальний текстовий засіб визначення), саме тому дизайнери можуть вважати, що основними видами діяльності цього керівництва є власне ідентифікуючи: концепції проблемного домену, які повинна підтримувати мова; взаємозв'язки між цими поняттями; та будь-які розділи або подання, які є підходить для цільової аудиторії мови (якщо застосовується).

Визначити ступінь розширюваності, на яку повинна звертатись мова. Після ідентифікуючи проблемний домен, розробник мови повинен подумати, чи буде він бути необхідним для зацікавлених сторін для зміни мови (наприклад, шляхом додавання нових елементів або зміна елементів, які вже присутні в мові). У стендіш виділяє три різні підходи до мови програмування розширюваність: перефразування, в якому нові функції додаються до мови шляхом визначення їх лише з точки зору ознак, які вже є в цій мові; ортофраза, в якому нові функції додаються до мови, не вимагаючи тих функцій, які є вже включено в мову; та метафраза, в якій інтерпретується мова правила змінюються, що призводить до того, що існуючі мовні функції обробляються в інший манера. Хоча ці підходи були визначені в контексті мови програмування дизайну, можна вважати, що такі мови програмування самі по собі є текстовими випадки моделювання мов. Таким чином, ці підходи може також застосовуватися для розширюваності при моделюванні мовного дизайну, хоча практичний використання кожного підходу може бути предметом обговорення як з точки зору доцільності, так і щодо додана

вартість у порівнянні з потенціалом користувачів робити помилки під час моделювання.

Слід зазначити, що, оскільки мови повинні постійно розвиватися, це так можливо слідувати цим рекомендаціям після первинного випуску мова, що будується, щоб уникнути позолоти мови та, таким чином мінімізувати його випадкову складність. Однак на таких ранніх стадіях дизайнери мов слід враховувати цільову аудиторію та обмірковувати можливість того, що користувачі мовиможливо, доведеться розширити його новими функціями, які дизайнер мов не міг передбачити. Це може допомогти мовному дизайнеру вказувати мову таким чином що його можна розширити в майбутньому без необхідності відтворювати всю мову.

Беручи до уваги виявлений проблемний домен, визначте модель мови - рівнями та їх ієрархією. Хоча тісно пов'язані з попереднім керівництвом Що стосується розширюваності, то ці два принципи дещо відрізняються: попередній Керівництво спрямоване насамперед на визначення того, які аспекти мови слід розширювати користувачами, в той час як поточний настанова - це те, де дизайнер мови формально структурується концепції мови (з проблемної області) на різні рівні моделювання (тобто, метарівні). Більш конкретно, конструктор мов повинен враховувати різні поняття та відповідні об'єкти (попередньо виявлені під час аналізу проблемної області), і визначати будь-які зв'язки між ними та їх склад; іншими словами, дизайнер мови повинен проаналізувати виявлені елементи (поняття та об'єкти), і визначити, що класифікує що, та що є будівельними матеріалами для чого.

Це впливає з поняття стратифікованого підходу до проектування. При такому підході, Складна система повинна бути структурована відповідно до набору рівнів, описаних з використанням різних мови. Кожен рівень визначається поєднанням елементів, які вважаються примітивними цей рівень, і такі примітиви самі по собі є результатом комбінацій, виконаних у попередній рівень. Автори також заявляють, що, таким чином, повинна міститися мова

кожного рівня примітиви, а також комбінація та механізми абстракції, відповідні для передбачуваний рівень деталізації.

Явне визначення цих рівнів моделювання (метарівні в метамodelюванні номенклатури) дозволяє мовному дизайнеру встановити ієрархічну структуру між ними метарівні. Ця ієрархія, в свою чергу, забезпечує практику суворого метамodelювання, якщо дизайнер мови гарантує, що: ні екземпляри зв'язків присутні на кожному метарівні; єдиний примірник взаємозв'язок, що виникає між елементами на різних метарівнях; та всі елементи *E L1* які трапляються на метарівні *ML 1*, пов'язані (через екземпляр зв'язку) з елементами *E L2* на іншому метарівні *ML 2*.

Визначимо будь-які обмеження, які можуть обумовити вибір метамodelьного ланцюга гейдж. Окрім концепцій та взаємозв'язків проблемного домену, дизайнер мови також повинен визначити будь-які обмеження, які можуть спричинити певні мови метамodelювання непридатний для вказівки абстрактного синтаксису мови.

Хоча більшість мов метамodelювання не накладає значного набору обмежень через притаманну їм простоту, дизайнери мов все одно повинні ідентифікувати відповідні обмеження (пов'язані або з проблемною областю, або з ідентифікованими рівнями моделювання), і оцінити, чи здатна обрана мова метамodelювання змодельовати їх або задовольнити обмеження задовільним чином. Можливими прикладами таких обмежень є використання багаторазового успадкування (наприклад, мова метамodelювання *Kermeta* підтримує декілька спадкування шляхом включення механізму вирішення конфліктів щодо оператора та особливостей перевизначення), або необхідність вказувати конкретні обмеження щодо конкретної моделі (наприклад, до модельовано той факт, що автомобіль повинен мати чотири колеса).

Окрім визначених обмежень, дизайнери мов також повинні зважити переваги та недоліки відповідних питань при вирішенні питання про використання, а мова моделювання загального призначення, така як *UML* або мова, орієнтована на *DSM*, для метаметамodelь мови. Кожен із цих підходів

матиме свої переваги і недоліки. Одним із цих важливих питань є підтримка інструментів, оскільки прийняття мова загального призначення замість *DSM* може полегшити діяльність зі створення моделей інструмент. Іншим прикладом є ремонтпридатність мови моделей (тобто, можливість нових користувачі правильно розуміють і змінюють модель), щодо яких вказує перевага у використанні *DSM* замість *UML*.

Визначте конкретний тип синтаксису, в якому цільова аудиторія є найбільш комфортною. Цей посібник призначений для вирішення дилеми, чи є мова повинен мати графічний конкретний синтаксис , текстовий, використовувати змішаний підхід до мови синтаксису бетону (забезпечуючи мову , що містить як графічний і текстовий елементи), або навіть шляхом надання декількох конкретних синтаксисів. Це, в свою чергу, має вирішальне значення для забезпечення якості та зручності використання мови, і, зрештою, його прийняття користувачами.

Для цього мовний дизайнер повинен враховувати цільову аудиторію та бажану спосіб представлення бажаної системи. Технічний досвід аудиторії також може бути фактором, що впливає на це рішення: відповідно до значна частка розробники віддають перевагу текстовим мовам через їх діяльність, що стосується традиційних мови програмування та сценаріїв, тоді як загальна сукупність зазвичай віддає перевагу графічні мови, якщо вони відображають форми реальних сутностей, які є представляється в моделі. Частота використання мови також слід враховувати: мова, яка часто використовується, повинна спробувати надати конкретне синтаксис, який гарантує, що користувачі можуть швидко і безпомилково визначати моделі ймовірно, не проблематично, що мова, яка використовується лише зрідка (і коротко), ні забезпечити синтаксис, за допомогою якого користувачі можуть швидко визначати моделі.

3.4. Висновки до розділу

У цьому розділі ми представили наш підхід до розвитку, орієнтований на *MDE* з *CMS* на основі вебдодатків. Цей підхід відрізняється від інших існуючих підходи до розробки вебдодатків шляхом використання кількох мов моделювання (замість однієї мови, такі як *UML* або *WebML*) та його модель механізму синхронізації. Більш конкретно, підхід визначає мови *CMS-ML*, *CMS-IL* та *MYNK*. *CMS-ML* та *CMS-IL* призначені для використання бізнес-дизайнерами (зацікавленими сторонами бізнесу) та *System Designers* (програмісти) відповідно. З іншого боку, модель *MYNK* мова синхронізації призначена для використання розробниками мов (наприклад, творці *DSL* які хочуть вказати нові мови, орієнтовані на *CMS*, відмінні від *CMS-ML* та *CMS-IL*) як спосіб, щоб дозволити додаткові види зацікавлених сторін для участі в веб - додатки з процес розвитку.

Ми також представили невеликий набір керівних принципів для визначення нових мод моделювання які можна використовувати з *MYNK*. Ці вказівки дозволяють створювати нові мови які краще відповідають потребам інших зацікавлених сторін, що може виявитися корисним у випадках де *CMS-ML* та *CMS-IL* недостатньо адекватні для створення передбачуваної мережі застосування.

Наступні розділи присвячені тому, щоб представити кожен з цих мов у більшій мірі рівень деталізації (хоча і не вичерпно), а саме обґрунтування вибору і компроміси, зроблені при розробці мов. Зокрема, у наступному розділі ми представляємо *CMS-ML*, мову моделювання цього підходу для бізнес-дизайнерів створити специфікацію високого рівня вебпрограми на базі *CMS*, яка повинна забезпечувати рівень підтримки деяких (або всіх) завдань організації.

РОЗДІЛ 4

CMS-ML: МОВА МОДЕЛЮВАННЯ *CMS*-СИСТЕМ

4.1 Види моделі та моделюючі ролі

Визначте конкретний тип синтаксису, в якому цільова аудиторія є найбільш комфортною. Цільова аудиторія не складається з розробників (які вже звикли справу з текстовим програмуванням та мовами сценаріїв), і тому візуальне моделювання мова, ймовірно, кращий. Тим не менше, для цієї мови не обов'язково бути суто графічним, оскільки може містити деякі фрагменти тексту. Крім того, деякі елементи цільової аудиторії можуть бути дещо знайомими мови візуального моделювання (для таких цілей, як моделювання доменів або специфікація робочого процесу), а саме *UML* та його діаграми класів та активності.

Після отримання цих вказівок мова *CMS-ML* була визначена відповідно до їх. Решта цього розділу присвячена презентації цієї мови та пояснення деяких дизайнерських рішень, що беруть участь у цьому процесі. Моделювання за допомогою *CMS-ML* в основному зосереджене на трьох різних і взаємодоповнюючих моделях типи: Шаблони вебсайтів, Анотації вебсайтів та Набори інструментів. Рисунок 4.1 підказує, як ці моделі пов'язані між собою: як шаблони вебсайтів, так і набори інструментів може посилалися на інші Набори інструментів (як це обговорюється далі в цьому розділі), але на анотації вебсайтів може прикрашати лише шаблони вебсайтів.

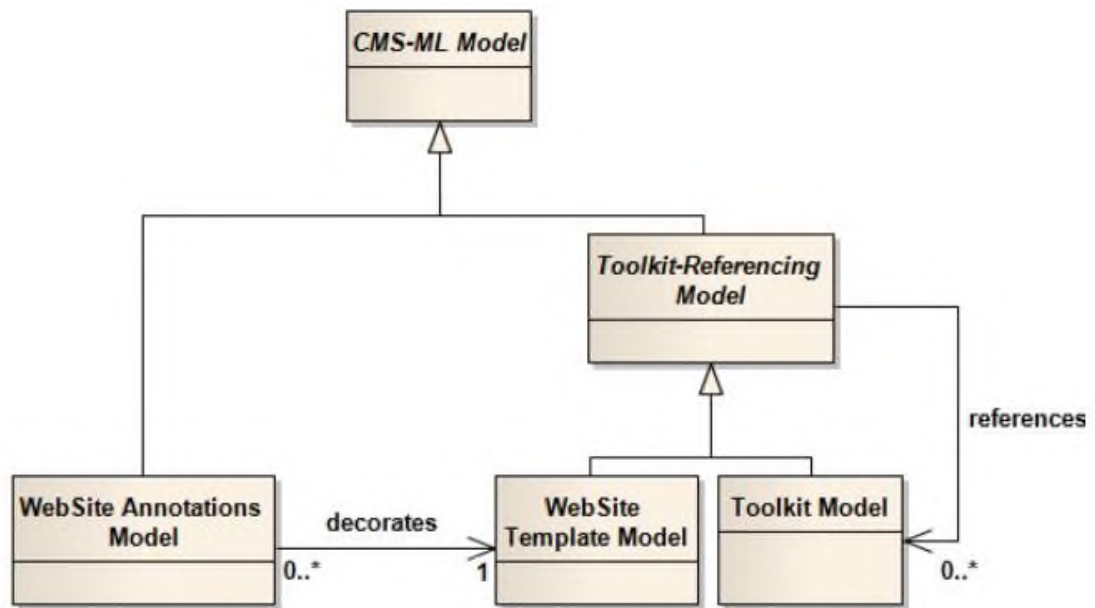


Рис. 4.1. Зв'язок між різними моделями *CMS-ML*.

Шаблон вебсайту (для простоти його також називають "Шаблон") – це модель, яка відображає передбачувану структуру вебпрограми; цей шаблон змодельовано за допомогою надантх елементів *CMS* – такі як *Role*, *Dynamic WebPage* та *WebComponent* від *CMS-ML*. З іншого боку, *Toolkit* дозволяє визначити нове моделювання, орієнтоване на *CMS* елементів, а саме шляхом вказівки моделі домену, інтерфейсу користувача та відповідної поведінки.

Як вже згадувалося раніше, шаблон вебсайту може посилатися на набір інструментів (або набір *Toolkits*), що, в свою чергу, робить елементи *Toolkit* доступними для використання в шаблоні. Крім того, набір інструментів також може посилатися на інші набори інструментів, дозволяючи можливість сценарії, в яких набір інструментів вдосконалює та розширює функціональність, яка була раніше визначено в іншому наборі інструментів. Нарешті, елементи шаблону вебсайту можна анотувати за допомогою вебсайту модель анотацій (або просто *Annotations*). Ця модель прикрашає шаблон вебсайту, дозволяючи дизайнерам шаблонів вказувати властивості, характерні для *CMS* (наприклад, налаштування конфігурації) без забруднення самого шаблону деталями, специфічними для платформи. Таким чином, з практичного Перспектива, дизайнери шаблонів вебсайтів *CMS-ML* не розглядають дві різні моделі – шаблон та анотації – а

скоріше одна модель, яка є результатом комбінування ці дві моделі (тобто модель, яка насправді є результатом декорування шаблону *Annotations*).

Необов'язково, щоб один дизайнер *CMS-ML* мав навички створювати обидві моделі вебсайтів та набори інструментів. Натомість я вважаю, що розробка *CMS-ML* часто виконується згідно з наступними моделюючими ролями, як пропонується на малюнку 4.2:

1. Дизайнер інструментарію, який моделює інструментарій;
2. Конструктор шаблонів вебсайтів (як правило, призначений просто як Конструктор шаблонів, для простоти тексту), який моделює шаблон вебсайту та, за бажанням, анує його з моделлю аотацій *WebSite*;
3. *WebSite Creator*, який створює екземпляри різних елементів, визначених у *WebSite* шаблон.

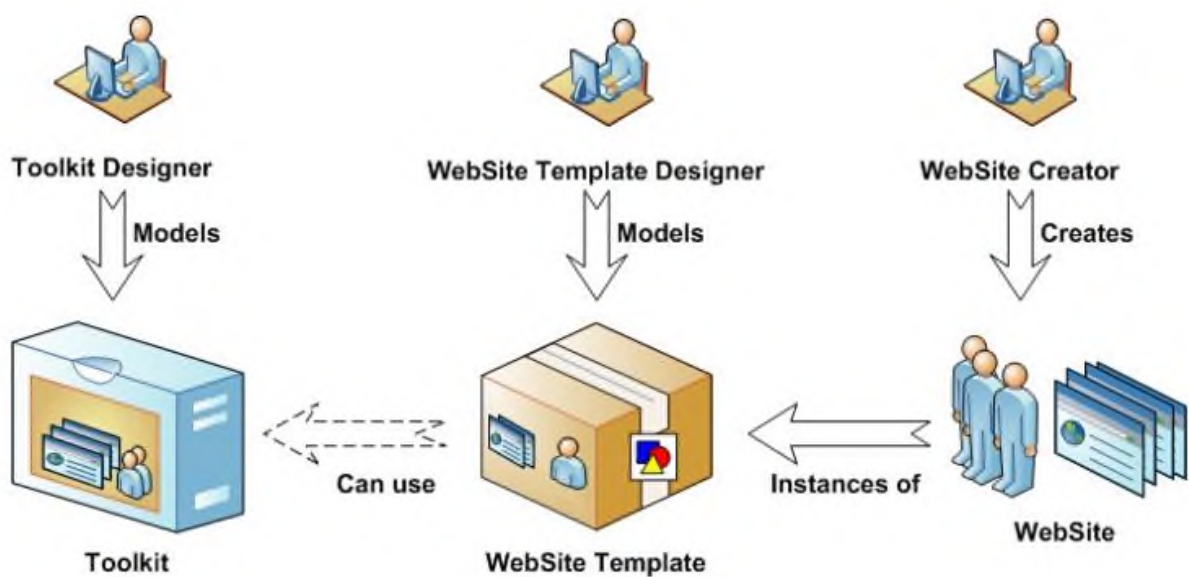


Рис. 4.2. Моделювання ролей та артефактів, розглянутих *CMS-ML*

4.2. Архітектура мови *CMS-ML*

Перш ніж розпочати опис *CMS-ML*, важливо виділити цей вебсайт шаблони та набори інструментів розташовані на різних рівнях . Хоча шаблони вебсайтів є призначений для створення абстракцій (тобто моделей) конкретних

вебдодатків за допомогою *CMS*-орієнтованих елементів, набори інструментів використовують загальні елементи моделювання для створення нових орієнтованих на *CMS* елементи моделювання. Оскільки деякі концепції інструментарію також є спеціалізацією *WebSite* концепції шаблонів (і тому екземпляри цих концепцій *Toolkit* розглядаються автоматично як екземпляри відповідних концепцій вебсайту), дизайнери шаблонів можуть потім використовувати ці концепції інструментарію для створення шаблонів вебсайтів так само, як і коли використовуючи заздалегідь визначені елементи моделювання шаблону. На рисунку 4.3 зображені метарівні, які розглядаються *CMS-ML*:

1. *Metalevel ML3* містить модель Моделювання інструментарію, яка забезпечує визначення загальних елементів моделювання *Toolkit*, які будуть використані для визначення моделей *Toolkit*. Цей метарівень не може бути змінений дизайнерами будь-якого виду;
2. *ML2* метауровня містить шаблон сайту моделювання та *WebSite Annotation*.

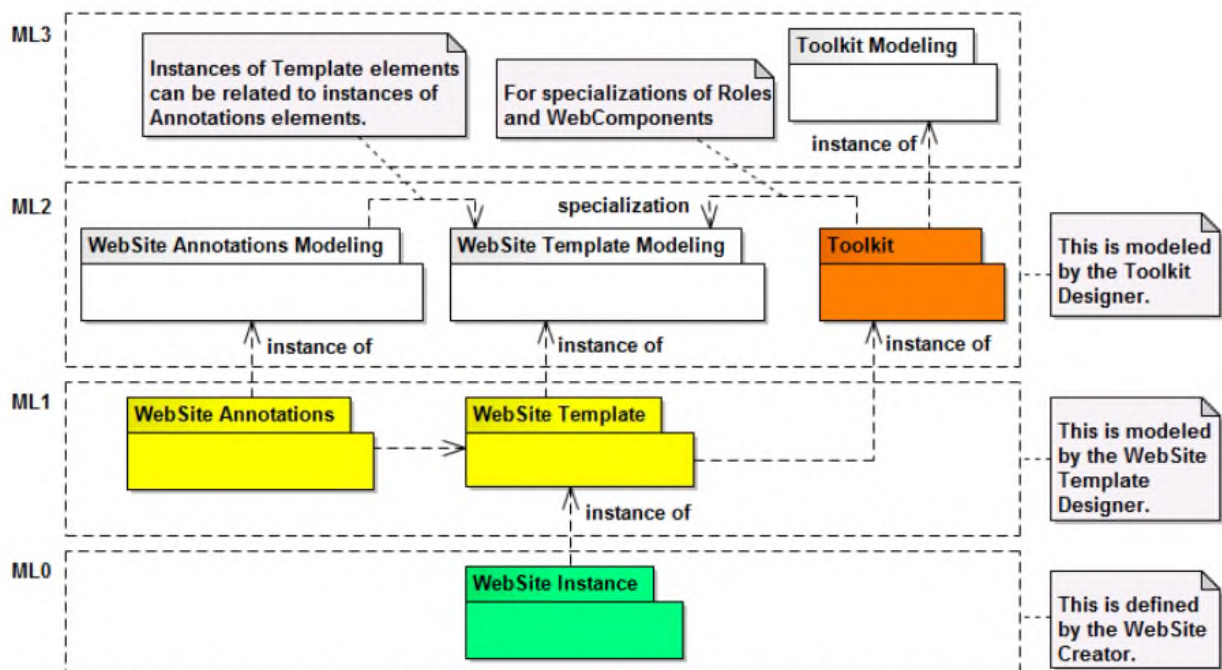


Рис. 4.3. Метарівні, розглянуті *CMS-ML*

Ці моделі, які забезпечують елементи моделювання, до яких ми звикли визначити моделі шаблонів вебсайтів та анотацій вебсайтів. Крім того,

інструментарій дизайнери можуть створювати екземпляри елементів загального моделювання, розташованих у *ML3*, в з метою визначення нових елементів, що спеціалізуються на елементах моделювання вебсайтів. Однак, як і моделі інструментальних моделей у *ML3*, моделювання вебсайтів та моделі моделювання анотацій вебсайтів виправлені і не можуть бути змінені будь-ким.

На метарівні *ML1* дизайнери вебсайтів можуть створювати шаблони вебсайтів та моделі анотацій вебсайтів за допомогою елементів моделювання, визначених у *ML2*. Ці елементи включають не тільки ті, що надані моделюванням шаблонів *WebSite* та моделі моделювання анотацій вебсайтів, а також елементів, визначених будь-яким набором інструментів моделі, доступні для шаблону *WebSite* за допомогою імпорту інструментарію концепції.

Нарешті, на метарівні *ML0* використовує *WebSite Creator* (а не конструктор шаблонів) елементи, визначені в моделі шаблону вебсайту (разом із його оформленням модель анотацій вебсайтів, якщо така існує) для налаштування конкретної інсталяції *CMS*.

Зазвичай для цього потрібно мати певний механізм управління вмістом, який встановлює відображення між екземпляром та елементом моделі (наприклад, стовпець у таблиці бази даних користувачів що для кожного рядка / користувача *CMS* ідентифікує відповідного користувача шаблону). Зверніть увагу, що є певна схожість між *ML1* і *ML0* і *M1* метарівень, знайдений в специфікації *UML OMG*, оскільки їх призначення полягає практично однакові. Головна відмінність полягає в тому, що, перебуваючи в екземплярах *UML* класи і об'єкти знаходяться на одному метарівні (*M1*), в *CMS-ML* вони знаходяться в *ML1* та *ML0* метарівнів відповідно.

Обґрунтуванням цієї архітектури метарівнів було вирішення проблеми мови розширення у простій, але елегантній манері; для зменшення випадкової складності що зазвичай впливає з використання моделей моделювання, подібних типу екземпляру в тому самому метарівні; та дотримуватися суворой доктрини метамоделювання, в якій не повинно бути екземпляру відносин, що перетинають більше, ніж одну межу рівня мета.

4.3 Моделювання шаблонів вебсайтів

Мова *CMS* надає набір елементів моделювання, орієнтованих на *CMS* – як правило називаються елементами *CMS*, оскільки системи *CMS* часто забезпечують певну підтримку цих елементів, що дизайнери шаблонів WebSite можуть використовувати для визначення своїх шаблонів для вебсайтів на базі *CMS* додатків. Модель шаблону вебсайту, що складається з екземплярів цих елементів *CMS*, визначається відповідно до набору подань (проілюстровано на рисунку 4.4):

1. Представлення структури, яке визначає структурні компоненти вебпрограми.
2. Перегляд ролей, який стосується набору обов'язків, які виконує вебпрограма очікує від своїх користувачів припущення.
3. Дозволи перегляду, вказуючи , які ролі мають доступ до вебдодатків.
4. Структурні компоненти.



Рис. 4.4. Погляди, що беруть участь у визначенні шаблону вебсайту

Фокус цих поглядів показує, що шаблон вебсайту має справу зі структурним проблеми вебпрограми, і тому вона буде використовуватися в основному для налаштування системи управління вмістом коли вебпрограма розгорнута на ній (наприклад, створіть нові сторінки та ролі, якщо вони це роблять не існує). Проблеми поведінки, навпаки, вказані лише в Наборах інструментів (описано в наступному розділі), оскільки поведінка вебпрограми на базі *CMS* є зазвичай визначається *WebComponents*, доступними в *CMS* (наприклад, *HTML WebComponent* буде поводитися інакше, ніж вебкомпонент

форуму), і навіть адміністратори *CMS* зазвичай не можуть змінити поведінку *CMS* (якщо вони не мають програмного забезпечення навички та доступ до вихідного коду платформи *CMS*), і може лише змінюватися деякі його параметри.

4.4.1 Структура вебсторінки на базі *CMS*

Вид структура є найбільш важливим, так як він відразу ж передає в вебдодаток структура сторінки за допомогою набору орієнтованих на *CMS* концепцій: *WebSite*, який представляє екземплярів вебдодатків і служить одночасно контейнером для *Dynamic WebPages* та як елемент, який імпортує набори інструментів (пояснюється далі в цьому тексті); динамічний *WebPage*, що представляє динамічно генеровані сторінки (в тому сенсі, що їх вміст можуть бути змінені через інтерфейс *CMS*), до якого користувачі матимуть доступ; контейнер, який є змодельовано в межах певної області динамічної вебсторінки і містить набір вебкомпонентів; та *WebComponent*, що представляє одиниці функціональності (наприклад, блог або форум) з яким користувач буде взаємодіяти.

Цей вигляд далі поділяється на два менших подання, перегляд структури макросів та подання мікроструктури. Подання структури макросів задає огляд інтернету додаток, що моделює лише динамічні вебсторінки та взаємозв'язки між ними у вікні мікроструктури вказано внутрішню структуру кожної динамічної вебсторінки (тобто, що таке вебкомпоненти на кожній динамічній вебсторінці, їх розташування та порядок відносно один одного). На рисунку 4.5 представлений абстрактний синтаксис для подання структури, де Макроструктура та Мікроструктура розглядають концепції (та взаємозв'язки між ними) можна спостерігати.

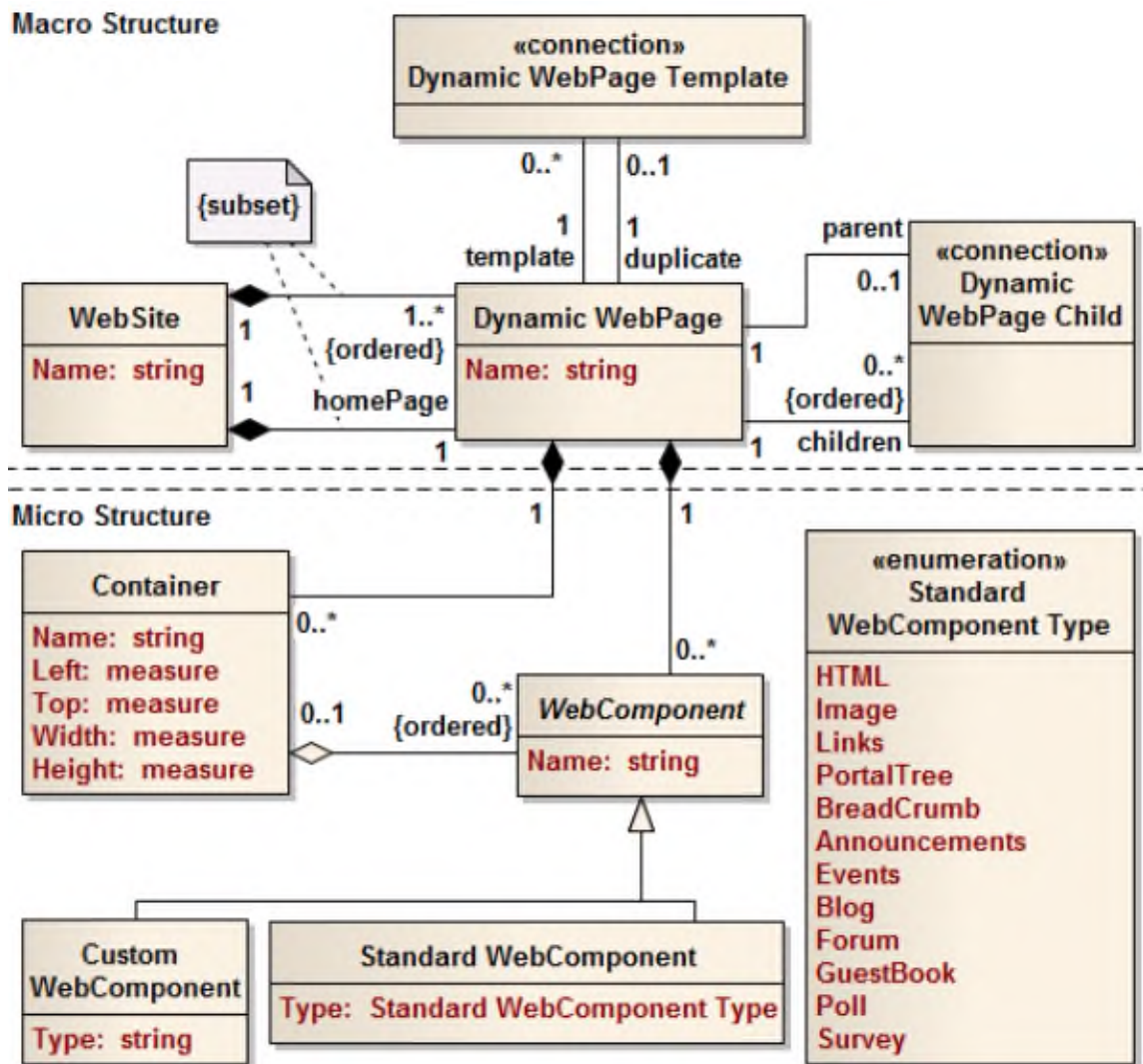


Рис. 4.5. Абстрактний синтаксис для подання структури вебшаблону

З іншого боку, на рисунках 4.6-4.7 зображено два приклади конкретного вигляду структури синтаксис: Рисунок 4.6 ілюструє подання структури макросів, а саме простий вебсайт, мій персональний вебсайт, який містить дві динамічні вебсторінки – Дім та Про мене, поки на рисунку 4.7 показано визначення вигляду мікроструктури вищезазначеної динаміки Домашня сторінка *WebPage*, а саме три контейнери – банер, тіло та панель навігації і два вебкомпоненти – Мій блог та Моя програма перегляду телевізорів. Ці приклади також показують, що *CMS-ML* конкретний синтаксис був визначений для того, щоб його було легко зрозуміти та намалювати вручну, не вимагаючи явного використання спеціалізованих інструментів моделювання для створення моделей.

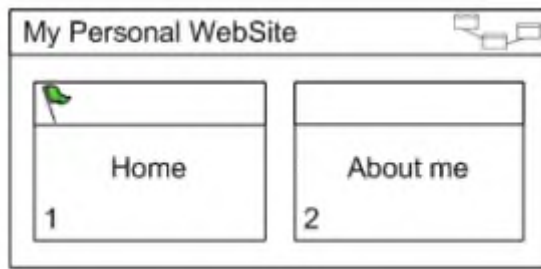


Рис. 4.6. Макроструктура синтаксису макросів

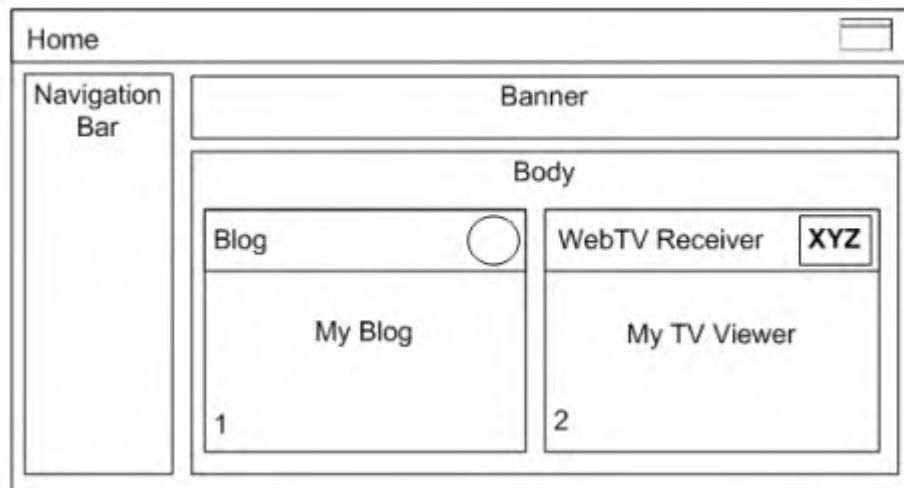


Рис. 4.7. Мікроструктура синтаксису макросів

4.4 MYNK: синхронізація моделі

У *CMS-ML* і мови *CMS-IL* є невід’ємними компонентами рішення запропонований раніше. Однак, хоча ці мови можуть вирішити питання моделювання потреби їх цільової аудиторії при створенні нових вебдодатків на базі *CMS*, як і раніше є необхідність в механізмі, який підтримує отримання *CMS-IL* моделей від моделей *CMS-ML*, а гарантує, що обидва типи моделей відповідають одній іншій. Відсутність такого механізму означало б, що моделі різними мовами (наприклад, *CMS-ML* та *CMS-IL*) повинні були бути синхронізовані вручну, що мало б недоліки: вимагання додаткових зусиль для виконання цього завдання ручної синхронізації; та перетворення завдання на помилки через його повторюваність.

Мова синхронізації моделі *MYNK* (*Model sYNchronization framework*), інша складова цього підходу до розробки була створена для задоволення цієї потреби. *MYNK* дозволяє отримати не лише модель *CMS-IL* із моделі *CMS-ML* (і навпаки), але також гарантує, що зміни в одній із моделей або навіть в обох моделях буде розповсюджено на іншу модель, щоб зберегти узгодженість між ними. У цьому розділі ми пропонуємо короткий опис *MYNK*. Більш конкретно, ця глава описує: обґрунтування визначення мови *MYNK*; підхід, який використовує *MYNK* підтримувати моделі, узгоджені між собою; Відносини *MYNK* з *ReMMM* метамодель; абстрактний синтаксис мови та конкретний синтаксис; і деякі аспекти щодо вирішення конфліктів та сумісності між *MYNK* та моделювання мов. Для отримання додаткової інформації щодо цієї мови ми також радимо читачі проконсультуватися в “Посібнику користувача *MYNK*”.

4.5 Огляд *MYNK*

Це особливо доречне обмеження для пропонованого вебдодатку на базі *CMS* підхід до розвитку, який передбачає, що обидва бізнес-дизайнер та системний дизайнер (ролі, передбачені цим підходом) можуть одночасно вносити зміни до відповідних моделей. Очевидно, що так би було бути неприйнятним, якщо явні зміни, внесені будь-якою роллю, просто втрачаються в перекладі. Цю проблему також можна знайти у відповідних роботах, явно орієнтованих на підходи до синхронізації моделі. Один із цих підходів використовує *ATL* для виконання моделі синхронізації способом, подібним до *QVT*, шляхом визначення відносно невеликих перетворень яка потім змінить цільову модель на еквівалент вихідної моделі. З іншого боку, звертається до двоспрямованої моделі синхронізації та чітко визначає Поняття кореспонденції, яке встановлює взаємозв'язок між джерелом і ціллю моделі. Однак усі ці підходи все ще несуть за собою втрату інформації у цільовій моделі, якщо ця інформація якимось чином конфліктує з інформацією в модель джерела; крім того, оскільки підхід є двонаправленим, це втрата інформації також може поширюватися на

вихідну модель, що ще більше ускладнює справи. Ці потенційні втрати розглядаються шляхом визначення понять синхронізації Дія прийняття рішень та синхронізації, яка складається з пунктів прийняття рішень (і можливих дій), що дозволяють користувачеві вибирати напрямок дій при виявленні таких конфліктів.

Однак ці пункти прийняття рішення пов'язані лише з елементами моделі (Артефакти), які може спричинити проблеми, коли рішення можуть охоплювати декілька таких елементів.

У контексті запропонованого підходу до розробки *MYNK* звертається до наступного цілі: забезпечення того, щоб дві моделі *CMS-ML* та *CMS-IL* були семантично еквівалентними один до одного; та забезпечення збереження цього відношення семантичної еквівалентності протягом усіх змін моделі, незалежно від того, які зміни застосовуються до однієї або обох моделей. Крім того, *MYNK* вважає модель не просто спрощеним поданням певної реальності у її останньому стані, а скоріше сума її історії (тобто різних змін що вона страждала, поки не дійшла до нинішнього стану). Звичайно, це не заважає визначення знімків моделі (у конкретні моменти життєвого циклу моделі), які можна використовувати кимось зрозуміти реальність, що моделюється. Важливо заздалегідь зазначити, що *MYNK* не призначений для забезпечення зворотного рейсу інженерні сценарії з урахуванням а модель A_1 (вказана певною мовою A), модель B_1 отримується за допомогою якогось механізму (наприклад, перетворення моделі $TA-B$, з A_1 в B_1); так само, використовуючи модель перетворення $TB-A$, модель A_2 отримана з B_1 . Якщо перетворення $TA-B$ і $TB-A$ точно симетричні (тобто вони протилежні один одному), тоді A_1 і A_2 повинні бути семантично еквівалентними (в тому сенсі, що вони повинні мати однакове значення).

Однак такий сценарій представляє застереження щодо вимоги обох мов мати того самого рівня виразності, оскільки в іншому випадку інформація буде втрачена між різними перетворення (і тому $TA-B$ ніколи не може бути симетричним до $TB-A$, і навпаки). Натомість *MYNK* націлений на створення таких сценаріїв. Це сценарій починається з двох семантично еквівалентних

моделей, $ML\ 1$ та $IL\ 1$, змодельованих за допомогою мови $CMS-ML$ та $CMS-IL$ відповідно. Після того, як бізнес-дизайнер внесе зміни (позначений $C\ ML1$ на малюнку) до $ML\ 1$, щоб отримати іншу модель $ML\ 2$, модель механізм синхронізації отримає ці зміни та сформує відповідний набір змін (з позначкою $C\ IL1$) у мові $CMS-IL$. З $IL1$ потім застосовуються до $IL\ 1$, що відбуваються інша модель $IL\ 2$, яка має семантично еквівалентна $ML\ 2$.

(a) Інжиніринг в обидва кінці.

(b) Синхронізація змін моделі.

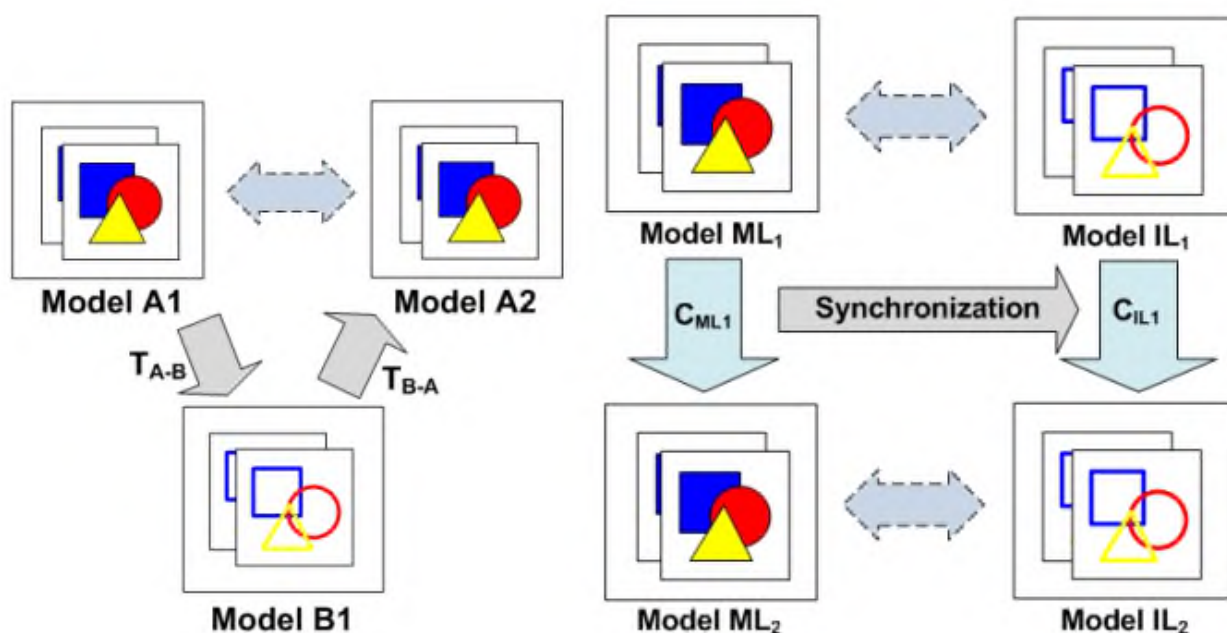


Рис. 4.8. Можливі сценарії синхронізації.

4.8. Метамоделі *MYNK* та *REMMM*

Однак на практиці не впевнено, що будуть дві моделі $CMS-ML$ та $CMS-IL$ синхронізовано лише після поширення однієї зміни (насправді це найімовірніше). Це відбувається тому все ще можуть бути застосовані зміни до моделі $CMS-ML$, виведені із змін внесених до моделі $CMS-IL$ (розробником системи) або передбачуваних змін початковою операцією розповсюдження зміни $CMS-ML \rightarrow CMS$. Більш конкретно, операція синхронізації моделі повинна складатися з а $CMS-IL \rightarrow CMS-ML \Rightarrow CMS-ML \rightarrow CMS-IL$ цикл, яку слід

виконувати до досягнення фіксованої точки (тобто програми набору змін до певної моделі призводить до тієї самої моделі).

Спосіб функціонування *MYNK* можна вважати аналогічним різноманітним інструментам та практики, з яких ми виділяємо кілька ілюстративних прикладів у наступних параграфах. Поняття тестування захоплення-відтворення складається із запису подій та їх застосування знову ж таки, пізніше. Ці тести часто використовуються в автоматизованому тестуванні програмного забезпечення сценарії (зокрема, при регресії та тестуванні інтерфейсу користувача), і виконуються шляхом запису та послідовності подій та їх очікувані результати, моделюючи їх виникнення записані події та оцінка того, чи результати еквівалентні очікуваним.

Аналогію можна знайти в *MYNK*, коли такі події розглядаються як зміни моделі. Процес розповсюдження змін *MYNK* також дуже схожий на те, що відбувається в редакції система управління (*RCS*), така як *Subversion* або *Mercurial*: єдина Інформація, яка надходить або надходить від робочої копії розробника, являє собою набір дельт (тобто відмінності) між останньою версією артефакту та версією, яка ввімкнена робоча копія розробника. Розробник може внести зміни до власної робочої копії, та об'єднати зміни з робочих копій інших розробників з власною, щоб синхронізувати з рештою команди розробників.

Ще однією адекватною метафорою цього процесу є пантограф, інструмент, який пов'язує два ручки, щоб будь-який рух до одного з ручок автоматично відтворювався (можливо, за допомогою інший масштаб) іншою ручкою. Ця метафора є особливо цікавою, як *MYNK* працює майже як пантограф, автоматично змінюючи одну модель відтворюючи їх на іншій моделі (наприклад, *CMS-ML* та *CMS-IL* відповідно). *MYNK* припускає, що моделі (разом із їх метамоделями) можуть бути вказані за допомогою метамодель *ReMMM*. Це тому, що *MYNK* "розглядає" будь-яку модель як набір моделі елементи, які пов'язані між собою графічно. На рисунку 4.10 наведено орієнтовна ілюстрація того, як *MYNK* інтерпретує моделі: Рисунок 4.10а відображає приклад ролі

інструментарію *CMS-IL* та відношення до його метакласу з точки зору *UML*, тоді як Рисунок 4.10b відображає це та ж модель, але з точки зору *MYNK*.

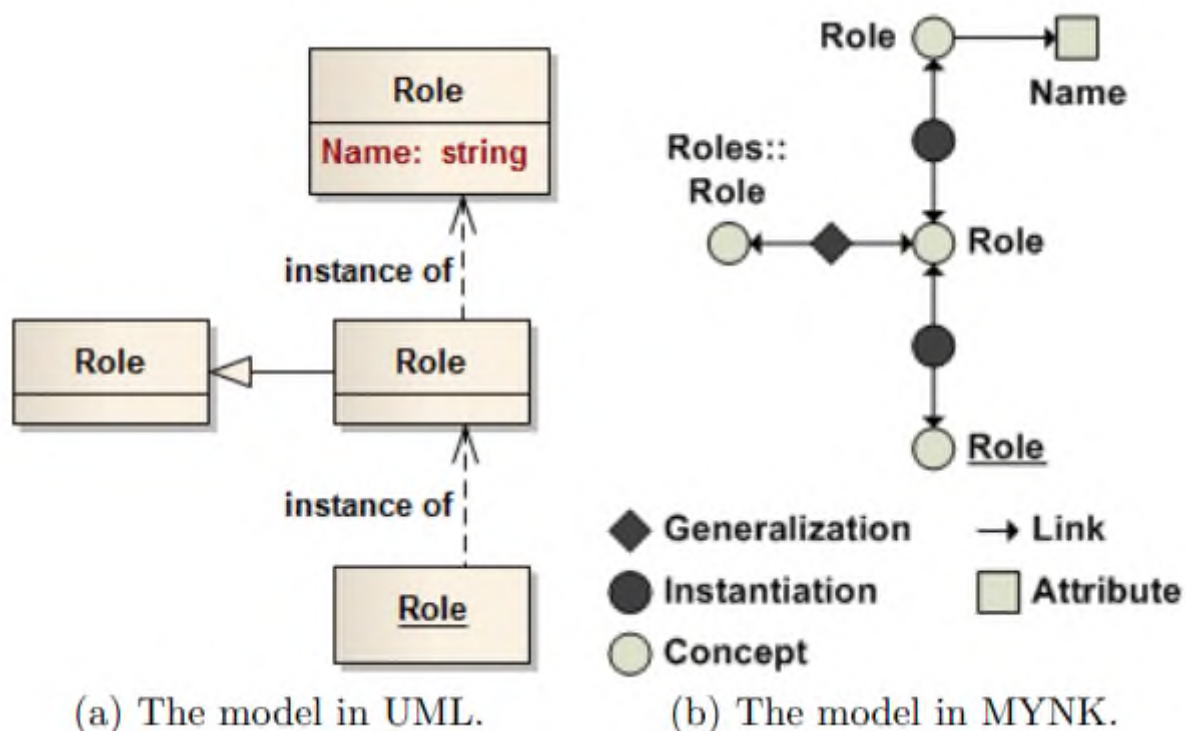


Рисунок 4.9. *MYNK* та *UML* моделі.

Важливо зазначити, що в *MYNK* немає відмінності між елементами різних метарівнів. Як видно з рисунка 4.10, елементи з усіх метарівнів (тобто з модель, метамодель, і т.д.), все вони вважаються в одній і тій же моделі, і пов'язані відносинами *Instantiation* (надані *ReMMM* і представлені на рисунку 6.3b у вигляді чорних кіл). Таким чином, *MYNK* розглядає як онтологічні, як і лінгвістичні типи відношення екземпляра, оскільки підтримується онтологічний екземпляр за співвідношенням *Instantiation*, і лінгвістичний приклад є суміжним з кожним елементом моделі, які інтерпретуються *MYNK* (наприклад, кожен із заповнених чорним колом на малюнку 4.10b представляє екземпляр метакласу *Instantiation ReMMM*, як показано легенда фігури). Зрештою, це застосування метафори бібліотеки, в якому невеликий набір елементів – метамодель *ReMMM* використовується для визначення елементів на різних метарівнях - моделі.

4.9. Модуль *MYNK*

Модуль *MYNK* відповідає за використання концепцій, зазначених у змінах та модулі простежуваності, а також визначення концепцій, що підтримують мову *MYNK* призначення (визначення операцій синхронізації моделі). На малюнку 4.11 наведена спрощена ілюстрація абстрактного синтаксису для модуля *MYNK*.

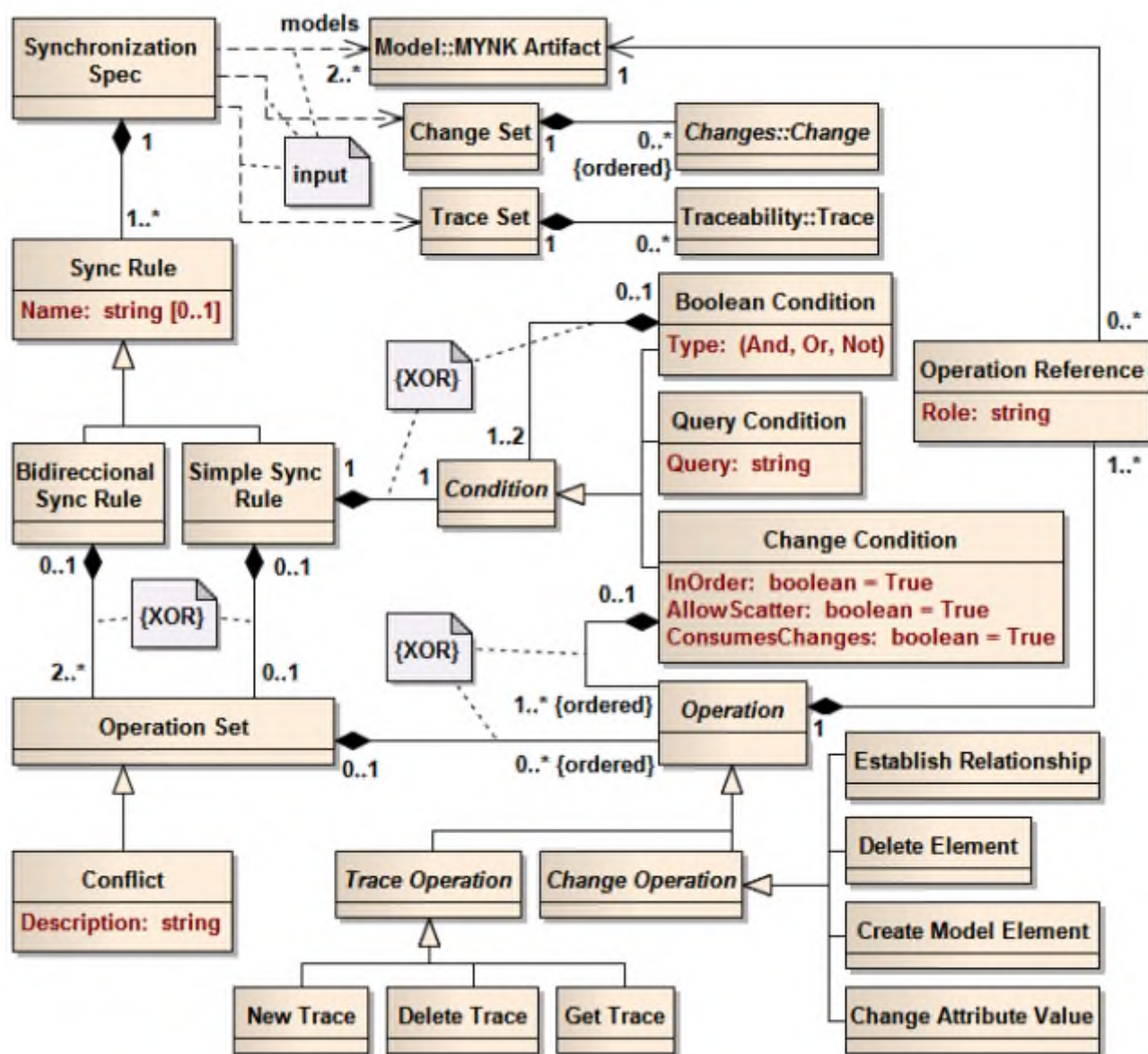


Рис. 4.10. Абстрактный синтаксис для модуля *MYNK*

Основна ідея цих концепцій полягає в тому, що *MYNK* працюватиме шляхом: аналізу множини змін; визначення, які правила синхронізації містять умовні операції, які відповідати деяким із запропонованих змін; отримати

шаблони для набору змін що має бути результатом передбачених змін; та на основі наданого набору слідів, перетворити ці шаблони на зміни, які застосовуватимуться до моделей, що синхронізуються.

Опис цього модуля ми починаємо із специфікації синхронізації (або просто із специфікації синхронізації, для стислості), який є контейнером для всіх інших понять у цьому модулі. Коли визначаючи нову модель синхронізації конструктор починає із зазначення синхронізації Елемент специфікації, в контексті якого будуть визначені інші елементи *MYNK*. Специфікація синхронізації містить набір правил синхронізації, які (як назва концепції пропонує) – це правила, якими керується процес синхронізації моделі. Точніше, там це два типи правил синхронізації - прості правила синхронізації та правила двосторонньої синхронізації – які виконують синхронізацію різними способами. Правило синхронізації також може характеризуватися іменем, яке є необов'язковим рядок, який можна використовувати для позначення мети правила.

На додаток до концепцій *Sync Spec* та *Sync Rule*, цей модуль також визначає ще один фундаментальне поняття, операція. Операцію можна розглядати як «щось, до чого робити». *MYNK* надає різні види операцій, які походять від концепції, визначені в модулях зміни та простежуваність: відстеження операцій та змін операції. Операція трасування складається просто з інструкції для отримання трасування елемент, створити новий елемент трасування або видалити існуючий. З іншого боку, а Операція зміни може розглядатися як шаблон або можливе збіг змін, саме тому наявні конкретні операції зміни визначені дуже подібним чином до конкретних змін у модулі Зміни.

Операції згруповані в набори операцій, які потім можна розглядати як очікувані набори змін (з часом поєднані з інструкціями відстеження) для відповідності при виконанні синхронізації моделі. Набори операцій важливі, оскільки *Sync* правила також містять такі набори. Однак тип правила синхронізації також визначає, як вони діють набори операцій інтерпретуються: 1) правило двонаправленої синхронізації містить лише перелік двох (або більше) наборів операцій. Кожен із цих наборів може виступати як умова застосування

правила синхронізації, так і результат правила забезпечується поєднанням усіх решти наборів операцій. Це слід зазначити, що існує застереження щодо набору, який діє як умова, оскільки операції "Нове трасування" та "Видалення трасування" ігноруються при визначенні чи відповідає набір операцій наданим змінам; 2) з іншого боку, просте правило синхронізації містить лише один набір операцій, який є результатом правила. Натомість умова правила визначається умовою концепція, яка може бути одного з наступних типів: логічний стан, який містить інші Умови та поєднує їх за допомогою булевих операторів (і, або, чи ні); умова запиту, яка містить запит (простий *SQL*-подібний рядок), який можна виконати над вхідними моделями і повертає значення true або помилковий; або умова зміни, яка містить упорядкований набір операцій які повинні бути успішно підібрані до набору змін. Нове трасування та видалення операції трасування не можна використовувати в контексті умов зміни.

4.10. Синтаксис *MYNK*

Як уже згадувалося раніше, *MYNK* є текстовою мовою, з конкретним синтаксисом запитів та трансформації мов, а саме *SQL* та *QVT*. У Додатку А представлений приклад бетону синтаксис мови *MYNK*, що ілюструє деякі відповідні аспекти, які будуть надалі пояснюється в наступних параграфах. Тим не менше, зацікавлені читачі можуть знайти додаткові подробиці щодо конкретного синтаксису *MYNK* у «Посібнику користувача *MYNK*».

Цей приклад починається з декларації про нову специфікацію синхронізації, яка вказана за ключовим словом *Synchronization* у рядку 1. Ця специфікація використовується для синхронізації між двома моделями, як зазначено у рядку 2 змінними *cmsml model* та *cmsil model* (які відповідають моделям *CMS-ML* та *CMS-IL* відповідно). Слід зазначити, що вказівка назви мови для кожної заявленої моделі є необов'язковим, оскільки воно служить лише як нагадування про те, які моделі слід надавати як параметри специфікації.

Всі операції визначаються однаково: назва операції, за якою йде набір аргументів. Приклад можна знайти в наступній операції уявлення:

CreateModelElement (модель: *cmsml_model*, клас: "*Toolkit.Binding*", елемент: *theB*)

Це подання відображає операцію створення елемента моделі, яка вказує на створення нового екземпляра концепції елемента моделі. Ця операція має три параметри, які мають такі ролі: модель, в якій Операція повинна знайти збіг; Елемент моделі, який є класом для нового Елемент моделі; та сам новий екземпляр. Явне визначення цих ролей (представлене на малюнку 9. 8 концепцією Reference Operation) також означає, що порядок, у якому параметри операції не вказані, не важливий, тобто особливо корисно уникнути помилок при визначенні нових правил синхронізації (або при читанні існуючих ті). У випадку з цим конкретним прикладом і тому, що це вказано в рамках зміни умови, ця операція має відповідати створення змінення елемента моделі за допомогою наступні властивості: слід було створити новий екземпляр елемента моделі в рамках моделі *cmsml_model* (надається як параметр для специфікації синхронізації; та новий екземпляр повинен мати прив'язку інструментарію *CMS-ML* онтологічний метакласом (тобто він повинен бути пов'язаний з *Binding* по *ReMMM* відносинам).

Цей приклад операції також піднімає деякі важливі питання, а саме: можливі значення, які можуть бути надані як параметри, та прив'язка змінних. Хоча ці питання тісно пов'язані між собою, вони справді містять деякі застереження, які стосуються *MYNK* розробники синхронізації повинні знати.

Перше питання стосується того факту, що параметр *Operation* може отримати один із параметрів наступне: змінна; *SQL*-подібний запит, який повертає значення; рядок; або конкатенація вищезазначеного. Спосіб інтерпретації кожного з них залежить від тип значення, яке очікується для параметра. Наприклад, параметр, який повинен отримати значення атрибута може отримати будь-яку: змінну, яка може бути або прив'язаною або незв'язаний (пояснюється далі), залежно від того, чи відповідає його відповідне значення -

якщо зв'язане – слід враховувати при визначенні збігів; або рядок. З іншого боку, параметр, призначений для посилання на екземпляр елемента моделі, може отримати змінну, або запит, який повертає певний елемент моделі. Також можна вказати, що певна операція слід виконувати лише за умови дотримання певної умови, використовуючи *WHEN* ключове слово (як показано в рядках 13 -15).

Запити задаються за допомогою ключового слова *SELECT* (натхненного *SQL*). Так само як запити *SQL*, запит *MYNK* складається з фільтра над набором елементів моделі, який в даний час обробляється синхронізація *MYNK*. Крім того, він також може бути використаний для вкажіть, що те, що слідує за ключовим словом *SELECT*, не повинно використовуватися як є, а натомість бути оціненим. У Додатку А подано кілька прикладів використання цього ключового слова, з якого я виділив наступне:

1. Клас параметрів у рядку 6 отримує рядок "*Toolkit.Binding*". Цей рядок є ідентифікатор артефакту *MYNK* – який, у свою чергу, посилається на конкретний елемент моделі екземплярі, тому його потрібно оцінити, щоб повернути відповідну Модель Елемент (замість самого рядка);
2. Один із аргументів у рядку 15, *True*, не є ні рядком, ні змінною, а отже необхідно обчислити, щоб отримати відповідне булеве значення *True*;
3. Рядок 18 містить запит, який повертає один з елементів, що визначає сама мова моделювання *CMS-IL*.

Іншою корисною конструкцією *MYNK* є макрос, приклад якого можна знайти в рядку 18. Ця конструкція (що характеризується маркером \Rightarrow у своєму конкретному синтаксисі) дозволяє розробник синхронізації, щоб вказати запит і призначити йому ім'я. Однак ці запити не оцінюються негайно; натомість макрос розгортається (і обчислюється його запит) лише тоді, коли використовується його ім'я (наприклад, макрос *assocBetweenReminderAndElement*, визначений у цьому конкретному прикладі ми вибрали читабельний ідентифікатор для кожного відповідного артефакту *MYNK* до елементів моделей *CMS-ML* та *CMS-IL*, виключно для ясності. Однак, як було раніше згадано, дизайнери мов можуть вільно застосовувати будь-яку схему іменування для своїх власних елементів.

Рядок 18, розширюється та обчислюється лише тоді, коли його назва використовується в операції, яка трапляється в рядку 19). Це надихається на поняття макросу, яке присутнє в програмуванні мови, такі як C, які зазвичай вважають макрос фрагментом джерела код, який визначається один раз і під час компіляції включається в різні пункти програми.

Друге питання стосується використання змінних (та їх можливих прив'язок) як значень для параметрів роботи. *MYNK* вважає, що змінні (тобто ідентифікатори, які не є рядки, ні запити) можуть бути в одному з двох станів: зв'язаному чи незв'язаному. Змінні обмежені коли їм уже присвоєно значення (тобто вони були прив'язані до цього значення, який може бути рядком або екземпляром елемента моделі), і в іншому випадку не мають зв'язку. Коли зв'язана змінна використовується в операції, вона встановлює обмеження, яке лише змінює це відповідати поточному значенню цієї змінної. З іншого боку, необмежений змінна використовується, щоб вказати, що змінна повинна приймати значення, знайдені в зміні, які також мають значення, що відповідають значенням у зв'язаних змінних операції (і, таким чином, стають пов'язана змінна). Звичайно, це залежить від механізму синхронізації (програми, яка виконує синхронізацію *MYNK*), щоб визначити, чи є якісь зміни, які адекватні збіги для набору операцій. Крім того, при визначенні збігів між операціями та змінами кожне правило синхронізації встановлює область дії, в межах якої всі виконуються перев'язки; коли аналізується інше правило, встановлюється інший обсяг, таким чином втрачаючи всі прив'язки – якщо такі є, які були зроблені раніше під час чергового аналізу правила. Це гарантує, що порядок, в якому пов'язані змінні (у контексті різних правил) не впливає на процес синхронізації.

Механізм прив'язки змінних *MYNK* був натхненний ідеєю зворотного ланцюга, метод умовиводу, який можна охарактеризувати (із спрощеної точки зору) як робота від мети до правил. Дія методів умовиводу базується на знаннях база і набір цілей: база знань (також звана базою правил) складається з набір операторів, що зазвичай вказуються як логічні правила, такі як умова *if* результат, що складають знання, з якими система буде працювати; і цілі є

гіпотетичні твердження, які користувач запитує або має намір побачити доведеними. Висновки двигунів, які використовувати зворотний ланцюжок, починаючи з передбачених цілей, і використовувати їх – разом із правилами бази знань – робити висновки про нову інформацію, доки цілі не будуть доведені (якщо це можливо).

Подібність до прив'язки змінних *MYNK* походить від того, що *Changes and Sync* правила можна вважати аналогічними цілям і правилам бази знань, відповідно, як механізм синхронізації *MYNK* повинен працювати, отримуючи набір змін та, поки обробляючи ці зміни (у тому порядку, в якому вони надані), намагаючись знайти синхронізацію правила, що відповідають цим змінам.

Посилаючись на приклад правила простої синхронізації, механізм синхронізації повинен інтерпретувати його таким чином:

Rule "Sync Rule example: Create Binding to Create Reminder" [

When

*CreateModelElement(model: cmsml model, class: SELECT
"Toolkit.Binding", element: theB)*

*ChangeAttributeValue(element: theB, attributeName: "EntityName",
newValue: newEntityName)*

*ChangeAttributeValue(element: theB, attributeName: "BindingPath",
newValue: newBindingPath)*

*ChangeAttributeValue(element: theB, attributeName: "IsDummy",
newValue: newIsDummy)*

*EstablishRelationship(model: cmsml model, type: Association, end1:
theCMSMLWebElement, end2: heB)*

do

*CreateModelElement(model: cmsil model, class: SELECT
"Toolkit.Reminder", element: theReminder)*

*ChangeAttributeValue(element: theReminder, attributeName:
"Description", newValue: ("Bind to " + newEntityName + "." +
newBindingPath)) WHEN newIsDummy IS False*

```

ChangeAttributeValue(element: theReminder, attributeName:
    "Description", newValue: "Dummy binding. Do nothing.") WHEN
newIsDummy IS True

ChangeAttributeValue(element: theReminder, attributeName:
    "Completed", newValue: SELECT True) WHEN newIsDummy IS
True

NewTrace(theB as "CMS-ML Binding", theReminder as "CMS-IL
Reminder")

GetTrace(from: theCMSMLWebElement, role: "CMS-IL WebElement",
    element: theCMSILWebElement)

assocBetweenReminderAndElement => SELECT WHERE type:
    Association, model: cmsil model, end1: SELECT
    "Toolkit.ToolkitElement", end2: SELECT "Toolkit.Reminder"

EstablishRelationship(model: cmsil model, type: Association, class:
    assocBetweenReminderAndElement, end1: theCMSILWebElement,
    end2: theReminder) ]

```

Таким чином, це правило синхронізації встановлює, що створення прив'язки *CMS-ML* відповідає до створення нагадування *CMS-IL*. Це нагадування має різний опис залежно від того, чи є палітурка манекеном (детальніше про палітурки доступно на «Посібник користувача *CMS-ML*»). Крім того, якщо Пов'язка є манекеном, то для властивості Нагадування Завершено також встановлено значення True. На рисунку 4.12 наведено (спрощений) ілюстрацію, орієнтована на *MYNK*, як повинні виглядати моделі *CMS-ML* та *CMS-IL* після цього відбувається синхронізація.

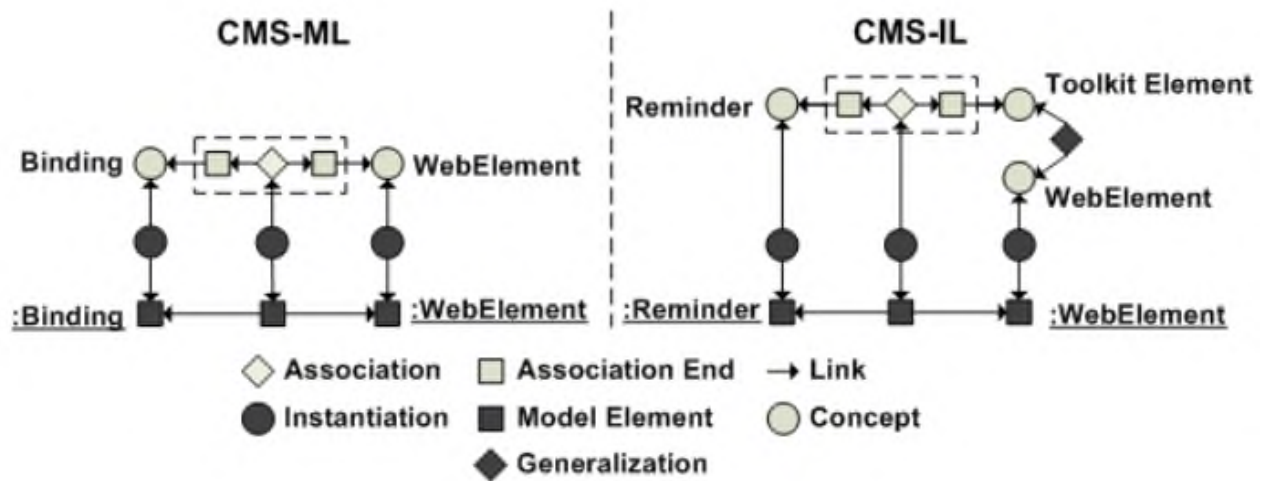


Рис. 4.11. Моделі *CMS-ML* та *CMS-IL* після застосування правила синхронізації

Ще однією важливою темою щодо прикладу в Додатку А є те, що це двонаправлене правило синхронізації особливо корисні для того, щоб уникнути визначення кількох простих правил синхронізації, у кожне правило має умову з набором операцій у моделі та призводить до операції з іншою моделлю, але ці набори операцій семантично еквівалентні між собою. Іншими словами, двонаправлені правила синхронізації спрощують випадки, коли умова і результати правила еквівалентні результату та умові іншого правила *B* (відповідно), шляхом згортання таких семантично еквівалентних правил (*A* і *B*) в єдине. Нарешті, слід зазначити, що різні інструменти, що підтримують *MYNK* (тобто інструменти, що підтримують записувати різні зміни, внесені в модель), можуть надавати зміни в іншому порядку, ніж той, у якому певне правило синхронізації (до якого призначені ці зміни бути визначеним) визначає його операції (наприклад, замість зміни значення атрибута операції з іменем, а потім описом – як у рядках 26–27 Додатку А, інструмент міг надати набір змін, в якому опис змінюється перед іменем). Тим не менше, як було раніше згадане, умова зміни може вказати це, а не узгоджуючи його операції в точному порядку, в якому вони були вказані, їх порядок повинен не враховувати при визначенні збігів. Хоча, з теоретичної точки зору, це це могло б призвести до матчів, які не мали б сенсу (наприклад, зміни значення атрибута на елементі моделі, який ще не створений), на практиці вищезазначений механізм прив'язки змінних

забезпечив би відповідність лише розумних були зроблені (звичайно, припускаючи, що зміни у наданому наборі змін є представлені у правильному та послідовному порядку).

4.11. Вирішення конфліктів

Як уже зазначалося, *MYNK* не виключає можливості виникнення конфліктів під час операцій синхронізації моделі. Насправді таких конфліктів слід очікувати на практиці, оскільки зацікавлені сторони намагатимуться модифікувати свою модель, щоб відобразити власний погляд та інтерпретація бажаного вебдодатку. Таким чином, якщо дві зацікавлені сторони мають різні (і суперечливі) погляди на цю систему, потім їх відповідні зміни до тієї ж моделі ймовірно конфліктують між собою.

Беручи до уваги концепції, визначені метамоделлю *ReMMM*, та різні типи зміни, які передбачає *MYNK*, можна передбачити, які види конфлікту може статися:

1. Операція створення елемента моделі може призвести до конфлікту при ідентифікації *MYNK* Артефакт вказується як рядок, результат запиту або вже зв'язана змінна (тобто вона є не незв'язана змінна), а інший артефакт *MYNK* з тим самим ідентифікатором вже існує;

2. Операція зміни значення атрибута може спричинити конфлікти, коли атрибут має значення вказано старе значення (і це не незв'язана змінна), і той самий атрибут має різне значення в моделі;

3. Операція "Встановлення відносин" може спричинити конфлікти, коли хтось із виникають такі умови:

- а) Новим елементом є узагальнення, і він створив би кругову петлю елементи узагальнення (тобто модель буде стверджувати, що існували *Model* елементи, що успадковуються від них самих, оскільки узагальнення є транзитивним);

b) Новим елементом є інстанція, і його створення створило б кругова петля (подібно до згаданої Узагальнення);

c) Вказано ідентифікатор нового елемента моделі (і він не є необмеженим змінної), а інший артефакт *MYNK* з цим ідентифікатором вже присутній у моделі (як в операції Створення елемента моделі);

d) Новим елементом є асоціація (між двома елементами моделі), і це новий елемент порушує принаймні одне з обмежень, визначених у його метакласі (наприклад, розглядаючи асоціацію між веб - елементом *CMS-ML* та *Binding* – це не повинно бути можливим для екземпляра *WebElement* до бути пов'язаними з двома або більше випадками прив'язки);

4. Операція «Видалення елемента» може призвести до конфлікту, коли модель елемента до видалити не існує.

Мовні специфічні семантичні обмеження – наприклад, примушування елементів моделі мати унікальні імена – поки не підтримуються поточною редакцією *MYNK*.

Іншим можливим джерелом конфліктів для всіх цих операцій є модель елемент посилається (наприклад, як метаклас нового елемента), але він не існує. Наприклад випадок, коли це може статися, є результатом операції отримання трасування: якщо немає елемента *Trace* із передбачуваними характеристиками (наприклад, тому що *Trace* набір, наданий специфікації синхронізації, був неповним), тоді отриманий елемент буде нульовим, що в свою чергу може негативно вплинути на майбутні операції. Існує великий набір ситуацій, які можуть призвести до конфліктів, представлених вище. Таким чином, тому, що не було б практичним вимагати розгляду механізму синхронізації всі ці ситуації та намагатися автоматично вирішувати їх, конфлікти слід вирішувати інтерактивно користувачем, способом, подібним до об'єднання змін у контролі редагування систем.

Я вважаю, що з практичної точки зору можливе врегулювання конфлікту можна класифікувати за двома ортогональними перспективами, на даний момент яка роздільна здатність виконується та використовувана мова моделювання.

момент, в який вирішується конфлікт, стосується етапу в процес синхронізації, при якому насправді відбувається роздільна здатність. Я вважаю є два можливі моменти для вирішення конфлікту, які в свою чергу призводять до відповідних підходів: коригувальний підхід та переважний підхід.

Корекційний підхід полягає у маніпулюванні моделями, які синхронізуються, щоб вони точно відображали очікувані стани після суперечливої операції відбудеться; іншими словами, від користувача очікується обхід виконання і замість цього вручну оновити моделі (моделі) до того, яким буде їх очікуваний стан після проведення операції.

Це схоже на спосіб вирішення конфліктів у *RCS*: коли користувач оновлює свою робочу копію, і конфлікт виявляється у текстовому файлі *F*, це вирішувати користувачеві відредагувати *F* так, щоб він містив не лише його попередній вміст, а й зміни, які є було передано до сховища (тобто, щоб його зміни були об'єднані). Для цієї операції користувач зазвичай має доступ до локальної копії *F*, як це було до оновлення, *F* поточний вміст у віддаленому сховищі та *F* у поточному конфліктному стані, з суперечливі текстові сегменти, обмежені набором маркерів. Однак, перебуваючи в *RCS*, користувач робить це злиття після того, як відбувається операція синхронізації файлів (і *F* знаходиться у конфліктному стані), у *MUNK* це виправлення відбувається замість операції; це для того, щоб уникнути потенційних проблем, які можуть виникнути в результаті виконання операції несподівані зміни в моделі.

З іншого боку, превентивний підхід характеризується тим, що вирішення конфліктів відбувається до того, як конфлікт насправді виникає (звідси і назва). Іншими словами, це вирішення конфлікту виконується шляхом: призупинення синхронізації процес безпосередньо перед тим, як відбувається операція виправлення моделей так, щоб вони знаходились у стані, який сумісний з тим, що очікував; та відновлення процесу синхронізації.

Як коригувальний, так і превентивний підходи вимагають, щоб користувачі мали можливість маніпулювати набором трасування синхронізації, щоб вони могли оновити елементи трасування, які з'єднують ці моделі (завдання,

яке зазвичай виконується в правилі синхронізації за допомогою функції отримання трасування, нова операція трасування та видалення трасування). Крім того, вони також вимагають від користувачів мати певні знання про те, що насправді буде робити операція синхронізації, і що умови, яких він очікує для того, щоб бути правильно виконаними; проте це застереження може бути пом'якшені шляхом анотування операцій з адекватними коментарями та / або наданням належних документацій.

Нарешті, щодо використовуваної мови моделювання, людина, яка виконує синхронізація може вирішити конфлікт, змінивши моделі, використовуючи будь-яку сама мова моделі (наприклад, *CMS-ML* або *CMS-IL*), або моделювання *ReMMM* мови. Цей вибір слід робити відповідно до комфорту та майстерності людини з *ReMMM* проти мови моделі, що, зрештою, є суб'єктивною проблемою, яка залежить на такі фактори, як досвід людини з моделюванням мов та метамоделювання.

4.13 Висновки до розділу

У цьому розділі я представив *CMS-ML*, високорівневу і незалежну від платформи мову моделювання, яка має на меті наділити нетехнічних зацікавлених сторін необхідним елементи для моделювання вебпрограми на базі *CMS* відповідно до їхніх цілей. На відміну від інших моделюючі мови, *CMS-ML* забезпечує механізм її розширення, який називається *Toolkit*, що дозволяє дизайнерам додавати нові елементи моделювання (ролі та *WebComponents*) до мова в контрольованому порядку. Однак *CMS-ML* демонструє відсутність виразності (враховуючи цільовий домен, вебдодатки на базі *CMS*), що є результатом компромісу між вивченням мови і кількість елементів моделювання, передбачених мовою. Це, в свою чергу, робить *CMS-ML* не може звернутися до певних особливостей (наприклад, специфікацій алгоритму), які є очікується від деяких вебдодатків на базі *CMS*. На відміну від *CMS-ML*, *CMS-IL* забезпечує

низький рівень абстракції над концепціями обчислень (у тому сенсі, що вона схожа на мову програмування), хоча вона все ще не залежить від платформи. Мета цієї мови – забезпечити незалежну від реалізації мову, яка може бути використовується для вирішення аспектів обчислень низького рівня, які не можуть бути оброблені *CMS-ML*, та розгорнути модель вебзастосунку на будь-якій платформі *CMS* (припускаючи, звичайно, це платформа може інтерпретувати моделі *CMS-IL*).

Також було представлено мову синхронізації моделі *MYNK*. Цей текстовий мова є наріжним каменем підходу до розробки вебдодатків на базі *CMS*, оскільки вона забезпечує синхронізацію (або в режимі реального часу (із затримкою) декількох моделей, які можуть змінюватися одночасно різними зацікавлені сторони.

РОЗДІЛ 5

ПОРІВНЯННЯ МОВ ВЕБМОДЕЛЮВАННЯ

5.1. Порівняння мов вебмоделювання

У попередніх розділах було представлено запропонований підхід до вебмережі, що базується на *CMS* розробка додатків – а також підтримуваних мов – із значним рівнем деталей. У *CMS-ML* мови моделювання і *CMS-IL* призначені для підтримки різних видів зацікавлених сторін (бізнес-дизайнер та системний дизайнер, відповідно), тоді як мова синхронізації моделі *MYNK* використовується для підтримки автоматичної синхронізації змін між моделями *CMS-ML* та *CMS-IL*.

У цій главі представлені зусилля з перевірки, які були здійснені для визначення ці результати цих досліджень. Цей опис складається з: порівняння на основі ознак між *CMS-ML*, *CMS-IL* та іншою державою мови художнього вебмоделювання; перевірка *CMS-ML* та *CMS-IL* через набір тематичні дослідження; перевірка *MYNK* при виконанні модельної синхронізації між моделі *CMS-ML* та *CMS-IL*; та оцінка про те, чи досягають наші результати задуману мету.

Слід зазначити, що в цій главі не передбачається затвердження *ReMMM* метамоделі або керівних принципів для специфікації мови. Це оскільки ми вважаємо, що визначення мов *CMS-ML* та *CMS-IL* є самим собою підтвердження цих двох внесків. Я здійснив попередню перевірку цього підходу (під час перевірки його компоненти: *CMS-ML*, *CMS-IL* та *MYNK*), щоб визначити, чи підхід може бути використана на практиці.

Важливий фактор у визначенні актуальності *CMS-ML* та *CMS-IL* як вебмоделювання мови – це сукупність функцій, які пропонують ці мови, точніше їх різноманітність та актуальність цих особливостей. Таким чином, одним із зусиль щодо перевірки було функціональне порівняння між *CMS-ML*, *CMS-IL* та мовами вебмоделювання. Результати цього порівняння зображені в Таблиці 5.1.

Моделювання доменів як *CMS-ML*, так і *CMS-IL*, також хоча концепції, надані *CMS-IL*, мають вищий ступінь виразності, ніж *CMS-ML* (наприклад, *CMS-ML* не підтримує концепцію методу *CMS-IL*). Більше того, це слід зазначити, що концепції моделювання доменів, передбачені цими двома мовами, є незалежно від: стійкості, оскільки вони не передають жодної інформації, що стосується бази даних або сховища деталі; та *UI*, оскільки їх моделювання *UI* не вимагає, щоб кожен елемент інтерфейсу був прив'язаний до елемента домену.

Моделювання бізнес-логіки. Хоча і *CMS-ML*, і *CMS-IL* підтримують бізнес логічне моделювання, вони роблять це по-різному. *CMS-ML* розглядає цей аспект у моделях *Toolkit*, оскільки подання *Tasks* дозволяє *Toolkit* дизайнери визначають послідовності дій, які слід виконувати під час використання вебпрограми; ці дії, у свою чергу, є рушієм для орієнтованих на інтерфейс та домен операцій, які можна вказати в інших поданнях (наприклад, у поданні тригерів взаємодії). Крім того, сторона перегляд ефектів дозволяє конструктору інструментів задати маніпулювання доменом за допомогою базової функції *CRUD* (Створення, читання, оновлення та видалення), хоча ця специфікація виконана дуже рудиментарно.

З іншого боку, моделі інструментів *CMS-IL* не орієнтовані на підтримку таких послідовності дій, а скоріше до того, як повинна поводитися система *CMS*, коли відбуваються ті події (наприклад, клацання на певній кнопці). *CMS-IL* також підтримує домен шаблони маніпуляцій: типові шаблони надаються новими, видаляють та зберігають ключові слова, які забезпечують функціонування *CRUD*; та можуть бути власні шаблони вказані за допомогою лямбда та методів у сутності домену. Ця стратегія дозволяє *CMS-IL* бути виразним, як типова мова програмування, подібним чином, як платформа *OutSystems Agile* стосується моделювання бізнес-логіки.

Моделювання навігаційного потоку. На відміну від усіх інших мов вебмоделювання, *CMS-ML* та *CMS-IL* не забезпечують підтримку моделювання навігаційних потоків між вебсторінками (які представлені як динамічні вебсторінки в обох мовах). Це тому що більшість систем управління вмістом не

часто обмежують спосіб, яким можуть користувачі переміщатися по вебсайту. Натомість на кожній вебсторінці користувачам подається символ меню (або схожа конструкція), що відображає структуру вебсайту або його частини – як набір гіперпосилань. Це меню, в свою чергу, автоматично генерується *CMS* системи і, в залежності від *CMS* системи, *CMS* адміністратор може нести відповідальність тільки за вказуючи, коли має з'являтися певне гіперпосилання (наприклад, деякі посилання повинні відображатися лише в певні вебсторінки, тоді як інші посилання мають бачити лише автентифіковані користувачі).

Що стосується потоку навігації між вебсторінками підтримки інструментарію, це зазначено непрямым чином: у *CMS-ML* це представлено дією подання завдання переходу між діями, які можуть підтримуватися різною підтримкою вебсторінки; з іншого боку, у *CMS-IL* це підтримується поданням коду, а саме класом *CMS* та його методом *goToPage*.

Моделювання інтерфейсу користувача. Моделювання інтерфейсу підтримується як *CMS-ML*, так і *CMS-IL*, хоча *CMS-IL* не підтримує це графічно. Щодо доступу керування, його можна вказати як у шаблонах вебсайтів, так і в метарівнях. Я також вважаю, що обидві мови підтримують визначення призначених для користувача елементів інтерфейсу, оскільки Вебкомпоненти (визначені в наборі інструментів) можуть розглядатися як такі користувацькі елементи використовується в шаблонах вебсайтів; однак визначити нові види *WebElements* неможливо наприклад, новий *WebElement*, який складається з тексту поруч із полем введення тексту – як це було б вимагати існування додаткового метарівню (для забезпечення основних елементів *HTML*).

Також підтримуються схеми взаємодії, оскільки обидві мови вважають певними події може відбуватися на кожному *WebElement* (наприклад, подія *Click* може відбуватися за допомогою кнопки). Нарешті, і *CMS-ML*, і *CMS-IL* підтримують прив'язку елементів домену до інтерфейсу користувача елементи, щоб змодельована вебпрограма могла відображати та / або маніпулювати певними екземплярами елементів домену. Однак, і на відміну від деяких сучасних мов, ці мови стосуються налаштування таких прив'язок на дозволяючи

конструкторам та розробникам інструментарію вказувати шляхи прив'язки, що дозволяє такі сценарії, як текстовий *WebElement*, що відображає ім'я певного домену інший екземпляр, тоді як інший текст відображає ім'я пов'язаного екземпляра домену.

Перетворення від моделі до моделі. Хоча *CMS-ML* та *CMS-IL* не є підходи до розробки самі по собі, вони призначені для підтримки запропонованої мережі на базі *CMS* підхід до розробки додатків. Крім того, наріжним каменем цього підходу є використання модельного механізму синхронізації, який може слід розглядати як узагальнення перетворень від моделі до моделі. Таким чином, розглянемо що підтримка перетворень від моделі до моделі вирішується запропонованим підходом (хоча і не традиційним чином, передбаченим іншими аналізованими підходами).

Сформована заявка заповнена. Це ще один аспект, в якому зв'язки між *CMS-ML* та *CMS-IL* найбільш помітні. *CMS-ML* – мова моделювання на високому рівні, недостатньо виразна для моделювання подробиці вебпрограми, яка відповідає усім вимогам, що її бажає зацікавлені сторони. Прикладом такої відсутності виразності є те, що *CMS-ML* не підтримує визначення алгоритмів. У свою чергу, це означає, що модель *CMS-ML* не має достатньо інформації як такої, щоб забезпечити можливість генерації повної вебпрограми (тобто, все одно потрібно буде втручання розробників).

З іншого боку, *CMS-IL* – це мова низького рівня, яка забезпечує всі концепції які необхідні для розробки вебдодатків на базі *CMS* із низьким або середнім ступенем складності. Причина, по якій ми говоримо це, полягає в тому, що концепції *CMS-IL* такі як *Role*, обробник подій або точка змінності – насправді надаються більшістю систем управління вмістом, і використовуються *CMS* на основі вебдодатків (наприклад, плагіни для *Drupal* або *Joomla*), щоб забезпечити додаткова функціональність. Звичайно, концепції, які не підтримуються конкретною системою управління вмістом система повинна емулювати компонент *CMS Model Interpreter*. Таким чином, можна вважати, що

модель *CMS-IL* може бути використана для створення повного вебдодатку на основі *CMS*, оскільки подальше втручання розробника не потрібно.

Незалежно від середовища розгортання. Моделі *CMS-ML* та *CMS-IL* можуть розглядатися як незалежні від середовища, в якому вони розміщені, оскільки вони не надають жодних концепцій, які використовуються лише в конкретній системі управління вмістом вони ідеально призначені для розгортання в системі управління вмістом, яка має систему управління вмістом встановлено компонент *Model Interpreter*. Хоча *CMS-IL* і підтримує це визначення коду, специфічного для платформи (за допомогою платформи *Lambda and Environment Snippet* концепції), розробник *CMS-IL Toolkit* може вказати еквівалентний код для інших платформ, що дозволяє усунути будь-яку специфіку платформи в моделі *CMS-IL*.

5.2. *CMS-ML*

Мова моделювання *CMS-ML* була розроблена для використання нетехнічними зацікавленими сторонами – називаються бізнес-дизайнерами в підході до розробки і зможуть моделювати вебдодатки на базі *CMS* за допомогою помірного ступеня складності. Таким чином, перевірка *CMS-ML* була проведена з використанням набору невеликих ілюстративних тематичних досліджень, які були використані для підтвердження зручності використання нетехнічними зацікавленими сторонами (не навчені впровадженню вебсистеми на базі *CMS* додатки) та його корисність при моделюванні передбачуваних вебпрограм. У цьому розділі ми представляємо найважливіші приклади валідації *CMS-ML*, які були проведені.

5.3. *CMS-IL*

Як і у випадку з *CMS-ML*, було вирішено провести перевірку *CMS-IL* через справу *WebC-Docs* вивчення. Більш конкретно, ми підтверджено його

визначенням в CMS-моделі IL з WebC-Docs вебдодаток. Звичайно, ця модель повинна була відображати оригінальну реалізацію настільки точно, наскільки це можливо. Наступні параграфи ілюструють та пояснюють модель CMS-IL, що виникла в результаті цього зусилля з перевірки. Шаблон вебсайту CMS -IL для WebC-Docs дуже схожий на CMS-ML шаблон і складається з еквівалента набір динамічних вебсторінок і ролей (а також зіставлення дозволів між ними). Лістинг 5.2. забезпечує представлення цього шаблону.

WebSite Template consists of

WebSite "My WebC-Docs Instance"

imports Toolkit "WebC-Docs"

has

HomePage "Home" with

Container "Navigation Bar" at (0.5%, 0.5%, 20%, 99%) with

WebComponent "Company Logo" of Standard type "Image"

WebComponent "Site map" of Standard type "PortalTree"

Container "Body" at (21%, 0.5%, 78.5%, 99%) with

WebComponent "Welcome!" of Standard type "HTML"

Page "Search" follows layout of Page "Home" with

WebComponent "Company Logo" of Standard type "Image" in

Container "Navigation Bar"

WebComponent "Site map" of Standard type "PortalTree" in

Container "Navigation Bar"

WebComponent "Search Documents" ("WebC-Docs"::"Search")

in Container "Body"

Page "Doc Management" follows layout of Page "Home" with

WebComponent "Company Logo" of Standard type "Image" in

Container "Navigation Bar"

WebComponent "Site map" of Standard type "PortalTree" in

Container "Navigation Bar"

WebComponent "Manage Documents" ("WebC-Docs"::"Manage Docs") in Container "Body"

Role "Operator" ("WebC-Docs"::"Operator")

Role "Manager" ("WebC-Docs"::"Manager")

Role "Viewer" ("WebC-Docs"::"Viewer")

Role "Operator" can edit content of WebComponent "Search". "Search Documents"

Role "Operator" can edit content of WebComponent "Doc Management". "Manage Documents"

Role "Operator" cannot configure WebComponent "Doc Management". "Manage Documents"

Role "Manager" can configure Page "Search"

Role "Manager" can configure Page "Doc Management"

Role "Manager" can (configure, edit content of) WebComponent "Search". "Search Documents"

Role "Manager" can (configure, edit content of) WebComponent "Doc Management". "Manage Documents"

Role "Viewer" cannot view Page "Doc Management"

Role "Viewer" cannot view WebComponent "Doc Management". "Manage Documents"

Лістинг 5.2. Шаблон вебсайту CMS-IL вебдокументів.

Структура, функції і види цього шаблону права доступу (представлені в рядках 3-19, 21-23 та 25-33 відповідно) семантично еквівалентні згаданим шаблону CMS-ML, і тому вони не будуть далі обговорюватися тут. Інші подання шаблону CMS-IL (які корисні лише для завантаження програми цілі) не визначені з наступних причин:

1. Перегляд Користувачі непотрібний, оскільки WebC-Docs не вимагає наявності такого конкретний користувач CMS;
2. Перегляд Мови не визначений, оскільки WebC-Docs не залежить від жодного конкретна мова;

3. Перегляди Артефакти та Зміст не є актуальними, оскільки *WebC-Docs* цього не робить вимагати, щоб будь-який з його *WebComponents* призначав певні рядки або вміст їх;

4. Перегляд візуальних тем не є важливим, оскільки *WebC-Docs* не призначений для надання готові альтернативні візуальні макети для своїх вебкомпонентів та допоміжних сторінок.

Набір інструментів. Модель *CMS -IL Toolkit*, яка зображена в Додатку В, може бути розглянута як збіг для моделі *CMS-ML*, (через довжину ця модель *CMS-IL*, лише деякі ілюстративні частини були включені в цей перелік для тексту стислості). Тим не менш, слід зазначити, що погляди інструментарію в обох мовах є суттєво різні.

Ролі перегляду (в рядках 3-7) не вимагає введення, так як семантично еквівалентно його омонімічному погляду у згаданій моделі *CMS-ML Toolkit*. Вигляд коду, у свою чергу, надається як набір лямбда та функцій, які є згадані (та використані) у поданнях, описаних у наступних параграфах. Вигляд подій, зображений у рядках 9-29, подає деякі основні методи управління документами – пов'язані події, а також невеликий набір обробників подій, які викликаються, коли такі події трапляються. Більш конкретно, цей вигляд визначає: функцію, яка називається Журнал документів дія, яка використовується для реєстрації операцій, що відбуваються над документами; Створений документ Подія та обробник подій, що викликає функцію дії журналу документа; аналогічно Подія – документ видалено – і обробник подій; та Керування подією, яку переглядають документи та обробник подій, який реєструє такі випадки.

Вид Мінливість задається в рядках 31-43, і забезпечує дві мінливості *Point* категорії, документи та ролі. Документи Категорія містить одну Мінливість *Point*, а також *Lambda*, яка реєструє нові значення, коли ця точка встановлена. З іншого боку, Ролі Категорія визначає три Мінливості точок, які, під час виконання, визначають які ролі *CMS* вважатимуться екземплярами ролей, визначених у Наборі інструментів Перегляд ролей.

5.5. Висновки до розділу

У цьому розділі описано тематичні дослідження для перевірки мови *CMS-ML*, *CMS-IL* та *MYNK*. Оцінка початкових питань дослідження і ціль (які були викладені в розділі 1) також була виконана для того, щоб визначити чи досягли результати нашої дослідницької роботи цю мету. Ці кейси продемонстрували, що ці мови (і запропонований підхід до розробки) можуть бути використовується для створення вебдодатків на основі *CMS* відносної міри складності.

Мова синхронізації моделі *MYNK* була перевірена через визначення специфікації синхронізації, призначена для синхронізації змін між моделями мов запропонована в цій дипломній роботі (*CMS-ML* та *CMS-IL*). Додаткові зусилля з перевірки були, що складається з невеликої специфікації синхронізації для синхронізації змін між *CMS-ML* та моделі *UML*. Ці характеристики синхронізації не включено в цю роботу, для стислості тексту.

Специфікація синхронізації, яка синхронізує зміни між моделями *CMS-ML* та *CMS-IL*, містить великий набір правил синхронізації, кожне з яких відповідає за відповідність набору змін у моделі *CMS-ML* (або *CMS-IL*) та застосовуючи відповідний набір змін до іншої моделі.

ВИСНОВКИ

Інтернет спричинив зміни у способі розробки та використання більшості програмозгорнуто. Зараз розробник створює вебдодатки, і для цього необхідно мати справу з різними поняттями, такими як запит, файл cookie або гіперпосилання, замість вікна або віджета. Крім того, новою обраною платформою розгортання, як правило, є вебсервер або подібний, замість настільної системи. Це також відкрило шлях для різноманітних рамок підтримки розробки вебдодатків. Особливим видом вебдодатків, що набирає популярності, є Система Управління Вмістом (*CMS*). Цей тип програми, як правило, забезпечує розумне ступінь налаштованості та розширюваності, який мають тенденцію системні адміністратори та кінцеві користувачі на користь, оскільки це пом'якшує необхідність розробки цілих нових програмних систем для підтримки контенту розподіл та / або конкретні завдання користувача; натомість сама система вже доступна, і налаштування полягають у розробці компонентів (або модулів, залежно від номенклатура системи), яка буде встановлена поверх системи *CMS*. Таким чином, ці системи можна розглядати як вебдодатки, так і як фреймворки вебдодатків.

З іншого боку, модельовані інженерні підходи (*MDE*) прагнуть сприяти процес розробки, зосередившись на моделях та концепціях замість вихідного коду. Більше зокрема, парадигма *MDE* спрямована на досягнення більш високого рівня абстракції шляхом адвокатської діяльності ці моделі повинні бути основними артефактами в процесі розробки програмного забезпечення, в той час як інші артефакти (наприклад, вихідний код та документація) можна отримати з цих моделей у автоматичний спосіб, за допомогою модельних перетворень .

Системи управління вмістом можуть стати наступним стандартом вебдодатків рамки. Вони також можуть скористатися перевагами, що надаються парадигмою *MDE*, оскільки ці підходи можуть суттєво прискорити розвиток та розгортання Інтернету додатків та функцій, а також спростити їх обслуговування.

У цій дипломній роботі я представив свою пропозицію щодо нового підходу, орієнтованого на *MDE* для розробки *CMS*- вебдодатків. Ця пропозиція орієнтована на розробку вебдодатків, що базуються (і розгортаються) на системах управління вмістом, та відрізняється від інших завдяки використанню багатьох мод моделювання як використання модельного механізму синхронізації, *MYNK*, щоб переконатися, що різні погляди на систему всі узгоджуються між собою. Запропонований підхід базується на два *CMS*-орієнтовані мови, *CMS-ML* і *CMS-IL*, які знаходяться на різних рівнях абстракції. *CMS-ML* моделює адреси на високому рівні; з іншого боку, *CMS-IL* використовує низькорівневі концепції (хоча мова не є специфічною для певної *CMS* система), щоб забезпечити спільну основу для побудови мов вищого рівня, таких як *CMS-ML*.

Використовуючи цей підхід, деякі зацікавлені сторони (бізнес-дизайнери, як їх ще називають) можуть використовувати мову *CMS-ML* для надання своїх власних погляд на те, якою повинна бути система. Коли різні зацікавлені сторони мають погляд системи, з якою вони погоджуються, розробники (які згадуються як *System Designers*) можуть вдосконалити відповідну модель *CMS-IL*, а саме шляхом додавання або налаштування функцій, які не можуть бути визначені бізнес-дизайнерами (оскільки відсутності виразності мови *CMS-ML*). Після *CMS-IL* модель є задовільний, його можна потім розгорнути в системі управління вмістом, або енергуючи вихідний код або надання моделі *CMS-IL* як вхідних даних для компонента інтерпретатора моделі *CMS*, який буде обробляти виконання виконання змодельованої вебпрограми.

Хоча такий підхід не знімає необхідності в ітеративному процесі розробки (а саме в проектах, які передбачати швидкий розвиток прототипів для представлення кінцевим користувачам програми), це дає спосіб пом'якшити додаткову роботу, необхідну для вирішення невідповідність поглядів зацікавлених сторін та розуміння розробником ці перспективи.

Вважаю, що твердження тези було успішно доведено, оскільки ця дипломна робота пропонує підхід до розробки вебдодатків на базі *CMS* що не

тільки задовольняє потреби різних типів зацікавлених сторін, але й дозволяє їм редагувати свої моделі вебдодатків спільно та без потенціалу втрати інформації, яка могла б спричинити виникнення конфліктів. Ми вважаємо це таким покращенням типового процесу розробки, при якому нетехнічні зацікавлені сторони визначають вимоги вебпрограми, а потім відмовляються від контролю над її розвитком розробникам, повернувши його лише на етапі тестування програмного забезпечення та (можливого) прийняття. Таким чином, неправильне тлумачення вимоги розробником може призвести до значних втрат зусиль і часу, тому що виникне необхідність ще раз розвиватися або змінюватися частини вебпрограми, які вже мали бути розглянуті.

MYNK мови синхронізації моделі можна розглядати як «клей» між цими мовами, орієнтованими на *CMS*. Без *MYNK* зміни між *CMS-ML* і моделі *CMS-IL* потрібно було б розповсюджувати вручну, що зробило б запропонований багатомовний підхід непрактичний та схильний до помилок. Синхронізація *MYNK* складається з набору правил синхронізації, які беруть набори змін і відображають їх у інші набори зміни; іншими словами, правило може розглядатися як відповідь на наступне питання: «Якщо моделі M і N еквівалентні, і M зазнає певних змін, як може бути модель N змінено, щоб знову стати еквівалентом M ?». Це запитання подано загально (і не згадує *CMS-ML* або *CMS-IL*), оскільки *MYNK* вимагає лише використання метамоделі *ReMMM*, яку ми вважаємо досить виразною для моделювання найновішої мови моделювання.

MYNK закладає основу для іншого напрямку досліджень щодо синхронізації та перетворень моделей. Використання змін моделі (замість самої моделі) як рушій для трансформації моделей - стратегія, яка, щоб наші знання, досі не розглядалися у дослідницькій галузі *MDE*, хоча це так поширена практика серед розробників програмного забезпечення, які використовують системи *RCS* - має деякі переваги над іншими існуючими методами трансформації моделі (наприклад, *QVT*, *ATL*), такими як:

1. Стає легше визначати нові правила синхронізації моделі, оскільки модель проєктувальник синхронізації повинен лише думати з точки зору

відносно невеликих наборів змін. Типовою альтернативою є визначення критеріїв, що визначають, чи є два цілих моделі еквівалентні одна одній. Хоча кінцевий результат в ідеалі завжди бути однаковим - можна було б отримати дві еквівалентні моделі M і N – процес визначення правил був би простішим. Крім того, визначення більш простих правил може уникнути потенційних недоліків дизайну, що впливають із правил складність;

2. Це дозволяє уникнути втрати даних у моделях, які синхронізуються, оскільки такі ситуації можна розглядати як конфлікти, які повинні бути врегульовані вручну.

СПИСОК БІБЛЮГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Браун М. Методы поиска информации в Интернете./ М. Браун – М., 2005. – 344 с.
2. Головач В.В. Дизайн пользовательского интерфейса. / Головач В.В. – М. : Наука, 2003. – 752 с.
3. Когаловский М. Р. Перспективные технологии информационных систем. / М. Р. Когаловский. – М. : ДМК Пресс, 2003. – 288 с.
4. Маннинг К. Введение в информационный поиск./ К.Маннинг, П. Рагхаван, Ч. Шютце. . – М.;, 2011. – 528 с.
5. *Microsoft Visual Studio* [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
6. Створення сховищ даних. Технології *OLAP* та *Data Mining* [Електронний ресурс]. – Режим доступу до ресурсу: https://pidruchniki.com/olap_data_mining/
7. Сховище даних [Електронний ресурс]. – Режим доступу до ресурсу: https://pidruchniki.com/74246/informatika/shovische_danih
8. Принцип работы поисковых систем [Електронний ресурс]. – Режим доступу до ресурсу: <https://works.doklad.ru/view/4yqZXQ0Pous.html>
9. Сховище даних [Електронний ресурс]. – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/%D0%A1%D1%85%D0%BE>
10. Сховища даних, архітектура, моделі та основи їх створення [Електронний ресурс]. – Режим доступу до ресурсу: https://studopedia.su/16_177900_shovishcha-danih-arhitektura-modeli-ta-osnovi-ih-stvorenniya.html
11. Технологія сховищ даних *Data Warehousing* [Електронний ресурс]. – Режим доступу до ресурсу: <https://studfile.net/preview/5118185/page:31/>
12. Порівняння і синхронізація дозволів баз даних *SQL Server* [Електронний ресурс]. – Режим доступу до ресурсу: <https://autosyncdb.com/ua/porivnyannya-i-synkhronizatsiya-dozvoliv-baz-danykh-sql-server/>

13. Аналіз найактуальніших серверних систем управління базами даних [Електронний ресурс]. – Режим доступу до ресурсу: http://vlp.com.ua/files/14_22.pdf
14. Порівняння *sql* та *nosql* баз даних [Електронний ресурс]. – Режим доступу до ресурсу: http://www.tech.vernadskyjournals.in.ua/journals/2018/6_2018/part_2/7.pdf
15. Kapadia, A. *Implementing cloud storage with Openstack swift: Design, implement, and successfully manage your own cloud storage cluster using the popular OpenStack swift software.* / Kapadia, A., Varma, S., Rajana, K. // United Kingdom: Packt Publishing, 2014 – pp. 134 – 136.
16. Arnold, J. *OpenStack swift: Using, administering, and developing for swift object storage.* / Arnold, J. // O'Reilly Media, 2014 – pp. 254 – 256.
17. Gormley, *Elasticsearch: The definitive guide.* / Gormley, C., Tong, Z.
18. Toigo, J.W. *The holy Grail of enterprise data storage.* / Toigo, J.W. // Prentice-Hall, 1999 – pp. 64 – 69.
19. Ali Babar. *Guidelines for Building a Private Cloud Infrastructure.* / Ali Babar. // IT-Universitetet i København, 2012 – pp. 274 – 276.
20. Girish L S. *Building Private Cloud using OpenStack.* / Girish L S., Dr. H. S. Guruprasad. // International Journal of Emerging Trends & Technology in Computer Science, 2014 – pp. 134 – 138.
21. Marc Farley. *Rethinking Enterprise Storage: A Hybrid Cloud Model.* / Marc Farley. // Microsoft Press, 2013 – pp. 24 – 28.
22. Kapadia, Amar, Kris Rajana, Sreedhar Varma. *OpenStack Object Storage (Swift) Essentials.* / Kapadia, Amar, Kris Rajana, Sreedhar Varma // Packt Publishing, 2015 – pp. 184 – 187.
23. Beach Daniel. *Building a Search Server with Elasticsearch.* / Beach Daniel. // Packt Publishing, 2015 – pp. 267 – 269.
24. Metz, Sandi. *Practical Object-oriented Design in Ruby: An Agile Primer.* / Metz, Sandi. // Addison-Wesley, 2016 – pp. 150 – 154.

25. *Hartl, Michael. Ruby on Rails Tutorial: Learn Web Development with Rails.*
/ *Hartl, Michael. //Addison-Wesley, 2016 – pp. 194 – 196.*

Додаток А

Synchronization

between (cmsml model in "CMS-ML", cmsil model in "CMS-IL")

consists of {

Rule "Sync Rule example: Create Binding to Create Reminder" [

When

*CreateModelElement(model: cmsml model, class: SELECT
"Toolkit.Binding", element: theB)*

*ChangeAttributeValue(element: theB, attributeName:
"EntityName", newValue: newEntityName)*

*ChangeAttributeValue(element: theB, attributeName:
"BindingPath", newValue: newBindingPath)*

*ChangeAttributeValue(element: theB, attributeName:
"IsDummy", newValue: newIsDummy)*

*EstablishRelationship(model: cmsml model, type:
Association, end1: theCMSMLWebElement, end2:
theB)*

do

*CreateModelElement(model: cmsil model, class: SELECT
"Toolkit.Reminder", element: theReminder)*

*ChangeAttributeValue(element: theReminder, attributeName:
"Description", newValue: ("Bind to " + newEntityName + "." +
newBindingPath)) WHEN newIsDummy IS False*

*ChangeAttributeValue(element: theReminder, attributeName:
"Description", newValue: "Dummy binding. Do nothing.") WHEN
newIsDummy IS True*

*ChangeAttributeValue(element: theReminder, attributeName:
"Completed", newValue: SELECT True) WHEN newIsDummy IS
True*

*NewTrace(theB as "CMS-ML Binding", theReminder as "CMS-IL
Reminder")*

*GetTrace(from: theCMSMLWebElement, role: "CMS-IL WebElement",
element: theCMSILWebElement)*

*assocBetweenReminderAndElement => SELECT WHERE type:
Association, model: cmsil model, end1: SELECT
"Toolkit.ToolkitElement", end2: SELECT "Toolkit.Reminder"*

*EstablishRelationship(model: cmsil model, type: Association, class:
assocBetweenReminderAndElement, end1: theCMSILWebElement,
end2: theReminder)]*

Rule "Bidirectional Sync Rule example: Creation of WebSite" [

*CreateModelElement(model: cmsml model, class: SELECT
"WebSiteTemplate.WebSite", element: theCMSMLWebSite)*

*ChangeAttributeValue(element: theCMSMLWebSite, attributeName:
"Name", newValue: newName)*

*NewTrace(theCMSMLWebSite as "CMS-ML WebSite",
theCMSILWebSite as "CMS-IL WebSite")*

<->

*CreateModelElement(model: cmsil model, class: SELECT
"WebSiteTemplate.WebSite", element: theCMSILWebSite)*

*ChangeAttributeValue(element: theCMSILWebSite, attributeName:
"Name", newValue: newName)*

*ChangeAttributeValue(element: theCMSILWebSite, attributeName:
"Description", newValue: newDescription OR "")*

*NewTrace(theCMSMLWebSite as "CMS-ML WebSite",
theCMSILWebSite as "CMS-IL WebSite")]*

Rule "Bidirectional Sync Rule example: WebSite name change" [

*GetTrace(from: theCMSILWebSite, role: "CMS-ML WebSite", result:
theCMSMLWebSite)*

*ChangeAttributeValue(model: cmsml model, class: SELECT
"WebSiteTemplate.WebSite", element: theCMSMLWebSite,
attributeName: "Name", oldValue: oldName, newValue: newName)*

<->

*GetTrace(from: theCMSMLWebSite, role: "CMS-IL WebSite", result:
theCMSILWebSite)*

*ChangeAttributeValue(model: cmsil model, class: SELECT
"WebSiteTemplate.WebSite", element: theCMSILWebSite,
attributeName: "Name", oldValue: oldName, newValue: newName)]*

Rule "Bidirectional Sync Rule example: WebSite description change" [

<->

*ChangeAttributeValue(model: cmsil model, class: SELECT
"WebSiteTemplate.WebSite", element: theCMSILWebSite, attributeName:
"Description", oldValue: oldDescription, newValue: newDescription)] }*