МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 13.00.00.000 ПЗ

Група ІПм-23-2

Клюка Назар

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Клюка Назар Андрійович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Методи та інструменти оцінки тестування трансформації моделей імплементації

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Клюка Н.А.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Михайлюк Ірина Романівна, к.п.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Клюці Назару Андрійовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Методи та інструменти оцінки тестування трансформації моделей імплементації”

керівник проекту (роботи) Михайлюк Ірина Романівна, к.п.н., доцент

затверджені наказом закладу вищої освіти від “ 18 ” листопада 2024 р. № /7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних та програмних технологій тестування моделей

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження методологій розробки програмного забезпечення на основі моделей
2. Дослідження методів та інструментів оцінки тестування трансформації моделей імплементації
3. Дослідження функціональності інструментів тестування трансформації моделі
4. Застосування інструментів оцінки тестування трансформації моделей імплементації

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Загальний процес трансформації моделі (рис. 1.1)
2. Процес тестування трансформації моделі (рис. 1.2)
3. Методологія огляду літератури по темі дослідження (рис. 1.3)
4. Графічний інтерфейс інструменту TractsTool (рис. 2.1)
5. Реалізація uml.ecore в Eclipse (рис. 2.2)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Дослідження методологій розробки програмного забезпечення на основі моделей.	15.10.2024	виконано
4	Дослідження методів та інструментів оцінки тестування трансформації моделей імплементації	08.11.2024	виконано
5	Дослідження функціональності інструментів тестування трансформації моделі	20.11.2024	виконано
6	Застосування методів та інструментів оцінки тестування трансформації моделей імплементації	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 76 с., 14 рис., 5 табл., 55 джерел.

Тема: Методи та інструменти оцінки тестування трансформації моделей імплементації

Об'єкт дослідження: процеси трансформації моделей у контексті розробки програмного забезпечення.

Мета роботи: аналіз і розробка ефективних методів та інструментів для оцінки тестування трансформацій моделей імплементації, а також визначення критеріїв їх придатності для промислового застосування.

Предмет дослідження: методи та інструменти для тестування і перевірки трансформацій моделей у програмних системах.

Результати дослідження

В роботі розроблено критерії оцінки придатності інструментів для тестування трансформацій моделей, що враховують промислові потреби і проведено комплексне дослідження придатності інструментів TractsTool, Matching Table Builder, USE та Efinder для програмних проектів.

Висновок

Отримані результати можуть бути використані для вдосконалення процесу тестування трансформацій моделей у програмних проектах. Зокрема, розроблені критерії оцінки та рекомендації щодо вдосконалення інструментів можуть бути корисними для розробників програмного забезпечення, що використовують моделі у своїй роботі.

ТРАНСФОРМАЦІЯ МОДЕЛЕЙ, МЕТАМОДЕЛІ, ТЕСТУВАННЯ ТРАНСФОРМАЦІЙ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОЦІНКА МОДЕЛІ, ФОРМАЛЬНА ВЕРИФІКАЦІЯ, АВТОМАТИЗАЦІЯ РОЗРОБКИ, МОДЕЛЬНО-ОРІЄНТОВАНА РОЗРОБКА

ABSTRACT

Master Thesis: 76 pp., 15 fig., 5 tab., 55 sources.

Thesis Subject: Methods and evaluation tools for testing the transformation of implementation models

The object of research: processes of model transformation in the context of software development.

The purpose of the work: analysis and development of effective methods and tools for evaluating the testing of transformations of implementation models, as well as determining the criteria for their suitability for industrial application.

Research subject: methods and tools for testing and verifying model transformations in software systems.

Research results

The work developed criteria for assessing the suitability of tools for testing transformations of models, taking into account industrial needs, and conducted a comprehensive study of the suitability of TractsTool, Matching Table Builder, USE and Efinder tools for software projects.

Conclusion

The obtained results can be used to improve the process of testing model transformations in software projects. In particular, the developed evaluation criteria and recommendations for improving the tools can be useful for software developers who use models in their work.

MODEL TRANSFORMATION, METAMODEL, TRANSFORMATION TESTING, SOFTWARE, MODEL EVALUATION, FORMAL VERIFICATION, DEVELOPMENT AUTOMATION, MODEL-BASED DEVELOPMENT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ATL - Atlas Transformation Language

MDE - Model Driven Engineering

USE - UML-based Specification Environment

M2M - model-to-model

M2T - model-to-text

MTB - Matching Table Builder

ASOME - ASML Software Modeling Environment

DCA - Data-Control-Algorithm

ASSL - A Snapshot Sequence Language

MRs - Metamorphic Relations

EPL - Epsilon Pattern Language

ВСТУП

Актуальність теми.

У сучасному програмному забезпеченні важливу роль відіграють моделі, які використовуються для опису та проектування складних систем. Трансформація моделей, зокрема M2M (Model-to-Model) та M2T (Model-to-Text), є ключовими процесами у розробці програмного забезпечення, що сприяють автоматизації й підвищенню ефективності. Проте, питання тестування коректності цих трансформацій залишається критичним, оскільки помилки на етапі трансформації можуть призвести до значних витрат у майбутньому.

В розробці програмного забезпечення спостерігається значне зростання використання моделей для автоматизації різних етапів розробки, таких як аналіз, проектування та генерація коду. Модельно-орієнтована розробка (MDD) забезпечує ефективність та якість програмних продуктів завдяки використанню трансформацій моделей, зокрема Model-to-Model (M2M) та Model-to-Text (M2T) перетворень. Проте, зростаюча складність систем та різноманітність метамоделей потребують надійних методів тестування трансформацій для забезпечення їх коректності. Помилки на етапах трансформації можуть спричиняти серйозні проблеми на пізніших етапах розробки, що призводить до збільшення витрат і ризиків.

Тестування трансформацій моделей залишається недостатньо вивченим напрямом, що призводить до обмеженої кількості інструментів, здатних адекватно вирішувати проблеми перевірки якості трансформацій. Наявні інструменти часто не забезпечують належної підтримки для промислових випадків або не мають достатньої автоматизації. У зв'язку з цим, актуальним є дослідження методів і інструментів, які б дозволяли ефективно та надійно тестувати трансформації моделей, що використовуються в реальних проектах. Це дослідження спрямоване на виявлення підходів, які забезпечують коректність і надійність трансформацій моделей, а також на

розробку критеріїв оцінки придатності інструментів для промислового використання.

Недостатньо розроблені інструменти для перевірки й тестування трансформацій обмежують здатність розробників гарантувати якість програмних продуктів. Тому дослідження методів і інструментів для оцінки тестування трансформацій моделей є актуальним та необхідним.

Мета дослідження - аналіз і розробка ефективних методів та інструментів для оцінки тестування трансформацій моделей імплементації, а також визначення критеріїв їх придатності для промислового застосування.

Об'єкт дослідження - процеси трансформації моделей у контексті розробки програмного забезпечення.

Предмет дослідження - методи та інструменти для тестування і перевірки трансформацій моделей у програмних системах.

Відповідно до мети роботи було сформовано наступні **задачі:**

- Провести аналіз сучасних методів і підходів до тестування трансформацій моделей.
- Дослідити функціональність інструментів TractsTool, Matching Table Builder, USE та Efinder для оцінки їх придатності до тестування трансформацій.
- Розробити критерії для оцінки інструментів тестування трансформацій.
- Провести дослідження випадків використання зазначених інструментів у невеликих та промислових сценаріях.
- Проаналізувати результати тестування та зробити висновки щодо придатності інструментів для промислового використання.
- Запропонувати шляхи вдосконалення існуючих інструментів для підвищення їх ефективності.

Методи дослідження.

У роботі використано методи порівняльного аналізу для оцінки підходів до тестування трансформацій моделей, експериментальні

дослідження для оцінки ефективності інструментів TractsTool, Matching Table Builder, USE та Efinder, а також методи формальної перевірки для аналізу трансформацій

Наукова новизна отриманих результатів.

Розроблено критерії оцінки придатності інструментів для тестування трансформацій моделей, що враховують промислові потреби і проведено комплексне дослідження придатності інструментів TractsTool, Matching Table Builder, USE та Efinder для програмних проектів.

Практичне значення магістерської роботи.

Отримані результати можуть бути використані для вдосконалення процесу тестування трансформацій моделей у програмних проектах. Зокрема, розроблені критерії оцінки та рекомендації щодо вдосконалення інструментів можуть бути корисними для розробників програмного забезпечення, що використовують моделі у своїй роботі. Виявлення сильних і слабких сторін інструментів дозволяє обґрунтовано вибирати їх для конкретних завдань у проектах розробки ПЗ.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 76 сторінок, і містить 14 рисунків, 5 таблиць, список використаних джерел із 55 найменувань.

РОЗДІЛ 1. АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕННЯ МЕТОДОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ МОДЕЛЕЙ

1.1. Опис предметної області

Model Driven Engineering (MDE) - це методологія розробки програмного забезпечення, яка спрямована на підвищення продуктивності та якості розробки програмного забезпечення. MDE використовує абстрактні моделі як основні артефакти для керування розробкою. Розробники програмного забезпечення в MDE визначають програмні системи в моделях. Ланцюжок трансформації моделей може перетворити моделі на код. Ланцюжок трансформації моделей складається з однієї або кількох трансформацій моделей. Правильність трансформації моделі значно вплине на якість створеного програмного забезпечення. У промисловості тестування трансформації моделей є поширеним методом виявлення дефектів у трансформаціях моделей та забезпечення якості інструменту трансформації моделей. Однак практика тестування трансформації моделей у промисловості стикається з низкою проблем, про що свідчить велике навантаження процесів та помилки в результатах трансформацій. Тим часом у літературі запропоновано кілька підходів, які зосереджуються на тестуванні трансформації моделей. Однак відповідність цих підходів не є зрозумілою. Існує розрив у знаннях між промисловістю та наукою. Придатність підходу, запропонованого науковцями, не є зрозумілою для промисловості, а потреби промисловості не є зрозумілими для науки. Тому ми провели це дослідження, щоб звузити розрив у знаннях між промисловістю та наукою.

Model Driven Engineering (MDE) — це методологія розробки програмного забезпечення, яка використовує моделі як основні артефакти для стимулювання розробки. Вона спрямована на підвищення продуктивності та покращення якості процесу розробки програмного

забезпечення. Замість того, щоб писати код вручну, розробники програмного забезпечення в MDE визначають програмну систему для розробки в абстрактних моделях, виражених відповідною мовою моделювання. Моделі можна аналізувати на правильність, щоб уникнути дефектів у згенерованому програмному забезпеченні, що покращує якість кінцевого продукту. Трансформація моделі є важливою частиною MDE. Трансформація моделі автоматизує розробку програмного забезпечення. Це процес, у якому програмне забезпечення трансформації перетворює вихідні моделі в цільові моделі або код цільовою мовою за певними правилами.

Як і будь-який інший артефакт програмного забезпечення, перетворення моделі необхідно специфікувати, проектувати, впроваджувати та перевіряти на правильність. Одним із способів перевірки є тестування. MDE все ширше застосовується в промисловості, і тестування трансформацій стає важливим завданням у процесі розробки. Сучасні процеси тестування в галузі стикаються з численними проблемами, які демонструють велике навантаження на процеси та помилки в результатах перетворень. У літературі було запропоновано ряд підходів для вдосконалення процесів тестування трансформації моделі. Проте досі недостатньо інформації про придатність цих підходів у промисловому контексті. Отже, проблема полягає в тому, наскільки придатні ці підходи і чому ці підходи зазнають невдачі, якщо вони непридатні.

Дана робота зосереджена на реалізації існуючих підходів для тестування трансформацій моделі та оцінки їх придатності в промисловому контексті. Розглядається випадок проекту, де кілька промислових проектів використовують методи MDE для розробки програмного забезпечення, включаючи перетворення моделей. Формулюються наступні питання дослідження:

1. Які основні виклики в практиці тестування трансформації моделі в галузі?

• Ціль 1: Описати поточний процес тестування трансформації моделі в галузі.

• Ціль 2: Визначити основні проблеми та ключові потреби в поточних процесах тестування трансформації моделі.

• Ціль 3: Описати поточні процеси тестування трансформації моделі і визначити основні проблеми та ключові потреби в цьому процесі.

2. Які існують підходи, які можуть покращити процес тестування в галузі ?

• Ціль 1: Опис існуючих підходів до тестування трансформації моделі.

• Ціль 2: вибрати підходи, які можна використати для покращення поточної практики тестування.

3. Наскільки придатні ці підходи?

• Ціль 1: Запровадити вибрані підходи в невеликих випадках і оцінити їх придатність.

• Ціль 2: Проаналізувати та повідомити про придатність вибраних підходів.

1.2. Основні поняття трансформації моделі

У MDE модель називається доменною моделлю, яка формально описує програмну систему.

Метамодель — це модель мови моделювання [15]. Метамодель визначає набір правил і обмежень, які визначають мову моделювання. Метамодель — це визначення абстрактного синтаксису мови моделювання. Кожна модель, виражена цією мовою моделювання, відповідає метамоделі.

Трансформація моделі – це автоматизований процес генерування моделей або тексту. Він приймає модель як вхідні дані та генерує модель або текст як вихідні дані. На рисунку 1.1 показано загальний процес перетворення моделі. Як видно з цього рисунку, вхідні моделі відповідають метамоделі та набору обмежень. Програма перетворення використовує

вхідну метамодель і вихідну метамодель. Він визначає, як елементи вхідної моделі перетворюються на вихідну модель. Програма трансформації може бути реалізована мовою загального призначення, такою як Java, або спеціальною мовою трансформації моделі, такою як QVTo1, ATL і т.д.

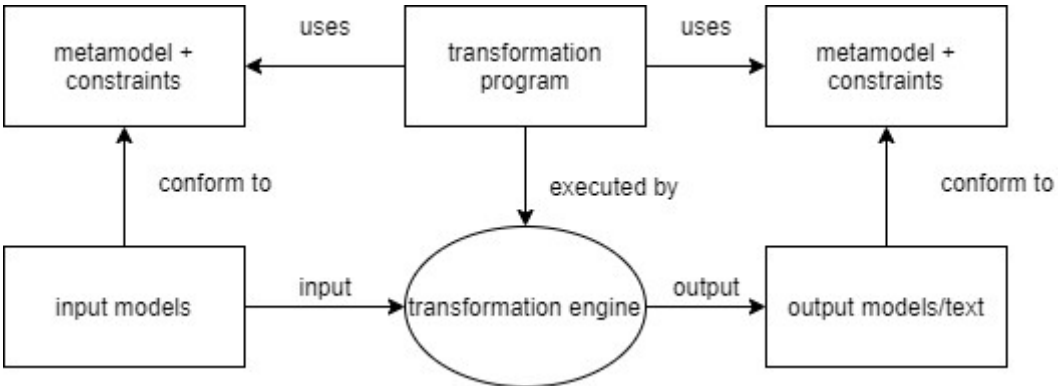

Рис. 1.1. Загальний процес трансформації моделі

Механізм перетворення виконує програму перетворення та генерує, можливо, декілька вихідних моделей, які відповідають їхнім метамоделям. Існує два типи трансформації моделі: трансформація з моделі в модель (M2M) і трансформація з моделі в текст (M2T) . Перетворення M2M генерує моделі, що відповідають метамоделям, тоді як перетворення M2T призводить до тексту, який може бути чим завгодно: програмним кодом, документами, звичайним текстом тощо. Як правило, коли різниця між мовою введення та мовою виведення на рівні абстракція стає великою, монолітні перетворення можуть мати деякі внутрішні проблеми, такі як невеликі можливості повторного використання, погана масштабованість тощо. Монолітне перетворення M2T може бути скомпоновано в ланцюжок перетворень моделі: послідовність менших суб-перетворень, що дозволяє ізольовано тестувати окремі перетворення моделі.

1.2.1. Тестування трансформації моделі

Спочатку ми представляємо чотири основні поняття, які використовуються в даній роботі.

Несправність і несправність. У контексті цього дослідження несправність - це помилка перетворення моделі. Невдача - це несподіваний результат трансформації. Несправність є ознакою несправності. Метою тестування є виявлення помилок у трансформації та управління ризиками збоїв. Загалом виявити всі несправності неможливо. Крім того, тестування не відповідає за виявлення, усунення несправностей або аналіз причин несправностей.

У тестуванні перетворення моделі функції Oracle використовуються для перевірки правильності результату перетворення. Як правило, функції Oracle можуть використовувати очікування користувача, специфікації перетворення, попередні версії тієї ж програми, порівняльні продукти тощо.

Тестерам необхідно виміряти якість набору тестів. Цей процес вимірювання називається оцінкою якості тесту. Тестове покриття та аналіз мутацій зазвичай застосовуються в літературі для кваліфікації набору тестів. Різноманітність моделей менш поширена, але також використовується в літературі.

У цьому дослідженні набір тестів, які використовують набір тестових моделей як вхідні дані, називають набором тестів. Хороший набір тестів має бути достатньо великим, щоб задовольнити певні критерії, але достатньо малим, щоб уникнути зайвого тестування. Надмірне тестування означає використання структурно схожих тестових моделей. Критерій охоплення спрямований на вимірювання якості набору тестів і прийняття рішення про те, коли тестувальники можуть припинити тестування, щоб перетворення моделі були кваліфікованими. Існує три типи покриття, які зазвичай застосовуються в сучасному тестуванні трансформації моделі: покриття трансформації, покриття специфікації та покриття метамоделі.

Покриття трансформації вимірює, скільки реалізації трансформації використовується набором тестів. Наприклад, в [12] прийняли покриття правил, щоб гарантувати, що кожне правило трансформації викликається принаймні один раз набором тестів.

Покриття специфікації. Специфікації є формалізованими вимогами коректності перетворень. Охоплення специфікації вимірює, наскільки будь-які специфікації та комбінації цих специфікацій охоплені набором тестових моделей [31]. Покриття метамоделі вимірює, наскільки вихідна метамодель використовується набором тестів. Наприклад, [18] і [19] прийнято класове покриття. Критерій охоплення класу стверджує, що для кожної концепції, визначеної в метамоделі, принаймні один конкретний екземпляр можна знайти в тестових моделях.

Аналіз мутацій може показати чутливість набору тестів. Аналіз мутацій змінює одиниці в реалізаціях трансформації для створення помилок. Ця помилкова версія трансформації називається мутантною. Якщо хоча б одна тестова модель може виявити мутант, то мутант називається вбитим мутантом. Тестер зазвичай створює набір мутантів, щоб перевірити, чи може набір тестів виявити та вбити всіх мутантів. Частка вбитих мутантів у загальній кількості нееквівалентних мутантів називається оцінкою мутації. Оператори мутації використовуються для створення мутантів. Якщо набір тестів отримує високу оцінку мутації, це вказує на те, що цей набір тестів здатний виявляти штучні помилки системи, що тестується.

Різноманітність моделей вимірює різноманітність набору тестів. Ефективний набір тестів має бути різноманітним, щоб запобігти надмірному тестуванню. Таким чином, різноманітність моделей може вказувати на якість набору тестів. Дослідження [29, 35] дали різні визначення показників різноманітності моделі. Визначені метрики різноманітності моделі були далі застосовані в методах розподілу еквівалентності.

1.3. Процес трансформації моделі в проектах розробки програмного забезпечення

Для розуміння питання трансформації моделі зосередимося на проекті ASML Software Modeling Environment (ASOME). Проект ASOME

спрямований на моделювання та побудову програмних систем, які відповідають шаблону архітектури Data-Contro l-Algorithm (DCA). Інструмент, який також називається ASOME, був розроблений для моделювання та створення частини програмного забезпечення ASML у цьому проекті. Зокрема, інструмент дозволяє моделювати структури даних і генерувати сховища програмного забезпечення для керування даними.

Щоб зрозуміти поточну практику тестування трансформації моделі та її основні проблеми будо проведено дослідження, відповідно до якого була сформована блок-схема процесу тестування. Крім того, представлено пояснення блок-схеми.

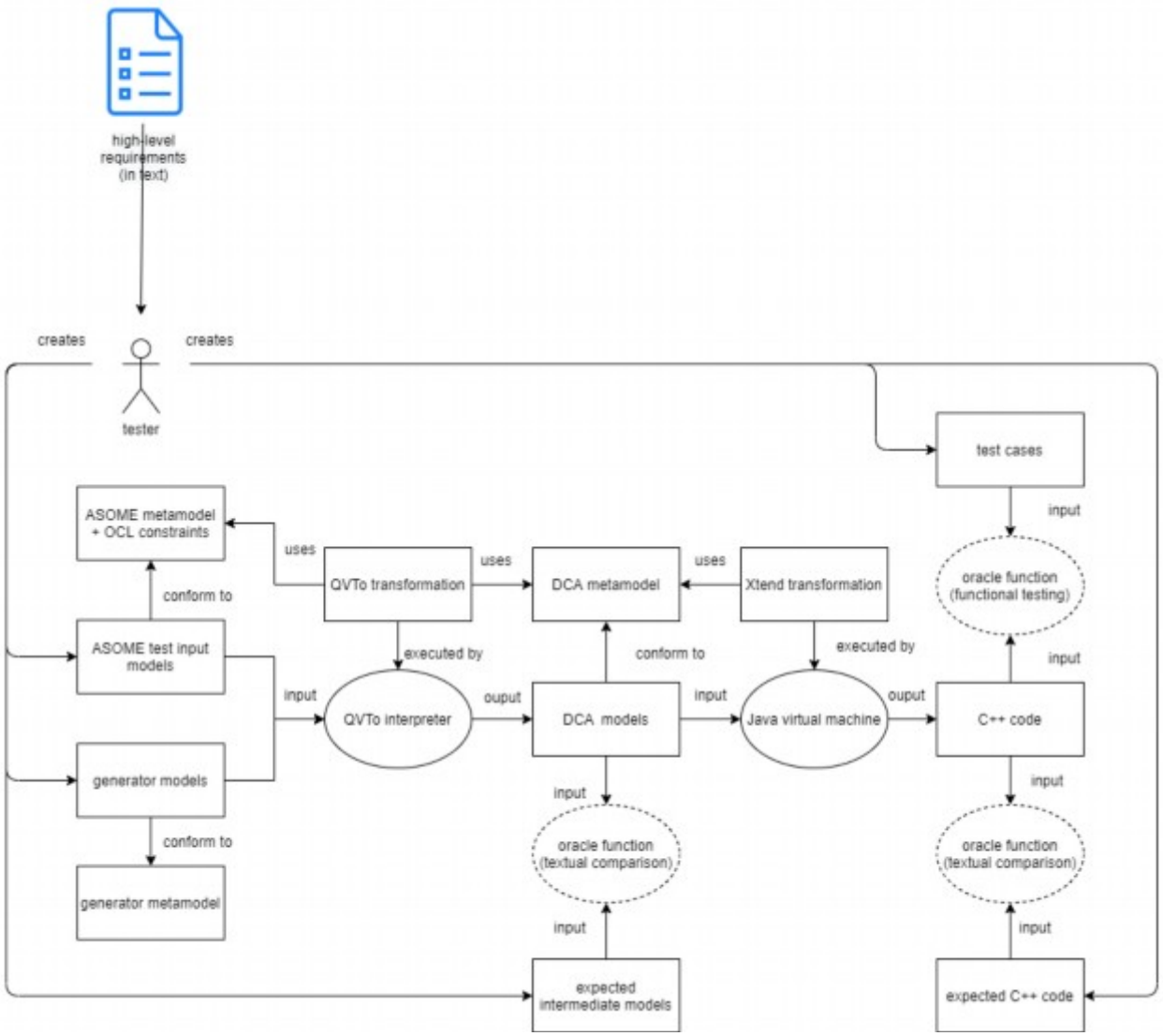

Рис. 1.2. Процес тестування трансформації моделі

Рисунок 1.2 показує ланцюжок трансформації та процес тестування в проекті. Ланцюжок трансформації моделі проекту складається з двох трансформацій моделі. Перше перетворення моделі — це перетворення моделі M2M у QVTo. Друге перетворення моделі — це перетворення моделі M2T у Xtend. Тестові моделі разом із моделями генераторів будуть перетворені в проміжні моделі за допомогою перетворення QVTo. Проміжні моделі — це моделі DCA, які відповідають метамоделі DCA. DCA є центральним архітектурним шаблоном для систем програмного забезпечення. DCA називають трьома аспектами: дані, керування та алгоритм. Перетворення QVTo виконується інтерпретатором QVTo. Крім того, моделі DCA перетворюються на код C++ за допомогою перетворення Xtend, яке виконується віртуальною машиною Java. Вимоги високого рівня часто неофіційно записують у тексті архітектори/інструментальники ASML. Тестер вручну створює тестові моделі та моделі генераторів відповідно до неофіційних вимог. Немає жодних вимог низького рівня чи формальних специфікацій, спільних для тестувальників.

Тестувальники використовують функції oracle для перевірки результатів. Функції оракула повертають результати порівняння фактичного результату з очікуваним результатом. Результати є двійковими відповідями, які вказують, чи прийнятний фактичний результат. Існує два типи функцій оракула. Один полягає в тому, щоб порівняти код моделі/згенерованого коду рядок за рядком. Інший полягає в оцінці результатів виконання згенерованого коду C++ за допомогою тестових випадків, створених тестувальниками. Спочатку очікуваних моделей DCA немає. Тестер повинен вручну перевірити правильність фактичного результату. Після того, як фактичний результат вважається правильним, вони будуть встановлені як очікуваний результат. Коли реалізація трансформації змінюється, тестер надасть ті самі вхідні дані та знову вручну порівнює фактичний вихід із очікуваним. Немає офіційної специфікації для тестування та немає автоматичного створення тестової моделі.

Є звіти про збої, включаючи типи та випадки. Однак немає жодного звіту про помилки в перетвореннях, які викликають збої, і немає аналізу того, чому ці помилки не були виявлені під час тестування. Більшість помилок виникли в трансформації M2T. Немає вимірювання якості тесту. Під час тестування були виявлені повторювані тестові моделі.

Підсумовуючи, проблеми тестування трансформації моделі є наступними:

- Складність створення кваліфікованого набору тестів. Вимоги високого рівня містяться в офіційних описах очікуваних функцій для згенерованого коду C++. Крім того, перетворення не дає правильного результату для нової функції вхідної моделі, оскільки нова функція не використовувалася жодною вхідною моделлю під час тестування. Не існує покриття метамоделі чи вимірювання покриття трансформації. Таким чином, немає індикатора якості набору тестів, який би допомагав тестувальникам завершити набір тестів. Крім того, ручна генерація тестових моделей неефективна.

- Складність створення ефективного набору тестів. Ефективний набір тестів має бути різноманітним, щоб запобігти надмірному тестуванню. Часто буває так, що тестувальники створюють дублікати або структурно подібні тестові моделі без усвідомлення. Для тестувальників є складним завданням переконатися, що повторюваних або структурно подібних тестових моделей уникають шляхом ідентифікації та розрізнення вхідних моделей вручну.

1.4. Огляд літератури щодо вирішення проблем, пов’язаних із тестуванням трансформації моделі

На рисунку 1.3 наведено методику проведення огляду літератури. Було два раунди перевірки та визначення пріоритетів.

Для першого раунду перевірки ми шукали дослідження, спрямовані на вирішення проблем, пов’язаних із тестуванням трансформації моделі.

Результат скринінгу першого туру – 36 досліджень. Є три основні стратегії пошуку, прийняті під час першого перегляду.

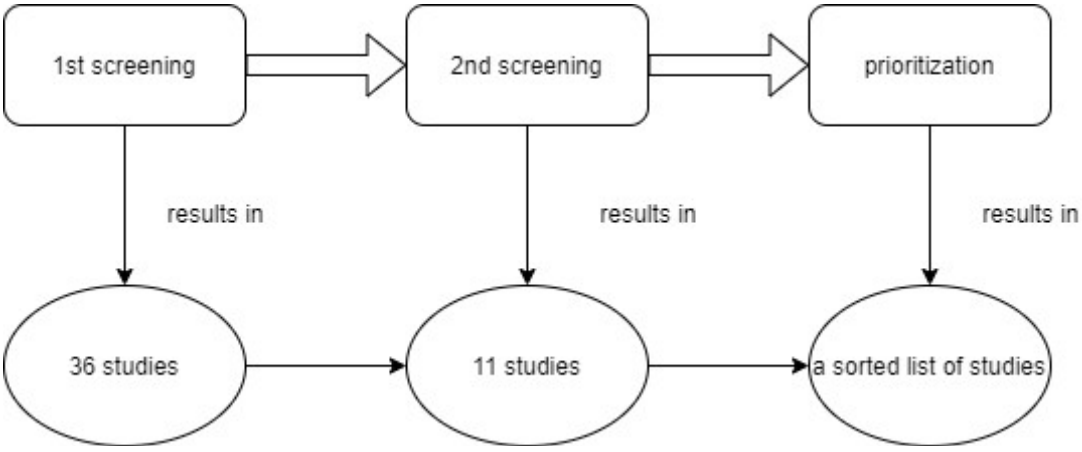

Рис. 1.3. Методологія огляду літератури по темі дослідження

Спочатку ми використали ключові слова для пошуку досліджень (написаних англійською мовою) у пошуковій системі Google Scholar, а саме "трансформація моделі", "модель у модель", "модель у текст", "тестування".

Також ми коротко ознайомилися зі змістом усіх досліджень після першого скринінгу та виключили ті, які не обговорювали (M2M і M2T) тестування трансформації моделі.

Ті, кого цитують дослідження тестування трансформації моделі, ймовірно, також говорять про тестування трансформації моделі. Тому ці дослідження з правильним змістом також були розглянуті.

Ми уникали дублювання первинних досліджень. Це означає, що якщо кілька досліджень проводяться як еволюція того самого підходу до тестування, то лише останнє дослідження буде включено до результатів першого скринінгу. Далі інформація про дослідження за результатами першого скринінгу узагальнена та представлена в таблицях. Ці таблиці будуть використані для аналізу здійсненності запропонованих підходів у заданому контексті.

Автоматизована генерація кваліфікованого та ефективного набору тестів. Більшість оглядів літератури [3, 4, 13, 31] про тестування трансформації моделі вказують на дві основні проблеми. Перша – це автоматизована генерація кваліфікованого та ефективного набору тестів. Хороша генерація набору тестів має бути автоматичною, оскільки генерація моделі введення вручну обмежить продуктивність тестувальників. Хороший підхід до створення набору тестів також має бути кваліфікованим, щоб набір тестів міг гарантувати, що перетворення моделі відповідають вимогам. Хороший набір тестів також має бути ефективним, а це означає, що дублікати або схожі тести можна виявити та уникнути.

Одним із найбільш дискутованих викликів у процесі тестування трансформацій моделей є наявність відповідного оракула. Оракул є другою сутністю, яка визначає очікуваний результат трансформації. У багатьох випадках тестувальники не мають чіткого розуміння того, як повинен виглядати правильний результат трансформації. Для вирішення цієї проблеми деякі дослідження пропонують альтернативні підходи до визначення оракулів, які можуть бути застосовані навіть у ситуаціях, коли традиційні методи не є ефективними.

При визначенні критеріїв тестування [22] вказали на необхідність визначення порогових значень для показників якості набору тестів. Пороги також можна назвати критеріями. Визначення критеріїв є важливими, оскільки вони повинні скеровувати тестувальників припинити тестування в потрібний час.

У літературі бракує обґрунтування вибору метрики якості набору тестів. Наприклад, в [31] запропонували використовувати критерії покриття специфікації трансформації замість критеріїв покриття метамоделі. Відповідно до [31], метою тестування трансформації моделі є перевірка наміру трансформації. Тому критерії охоплення специфікації є більш придатними, ніж критерії охоплення метамоделі. Однак це твердження не

було детально розроблено в цьому дослідженні, і немає достатнього дослідження щодо порівняння запропонованих критеріїв охоплення.

1.5. Порівняння підходів до трансформації моделей

1.5.1. Тестування трансформації M2M

У літературі запропоновано різні класифікації досліджень тестування трансформації M2M. Існує класифікація техніки на 9 категорій відповідно до методів: алгоритми пошуку графів, випадкове тестування, еволюційне тестування, розв'язання обмежень, перевірка моделі, статичний аналіз, абстрактна інтерпретація, тестування розділів і нарізка. В [4] класифікували дослідження відповідно до трьох фаз тестування трансформації моделі: створення тестових моделей, визначення критеріїв покриття та розробка функцій оракула. В [3, 44] об'єднали генерацію тестових моделей і визначення критеріїв охоплення в одну класифікацію, що генерує тестові моделі. Перша класифікація не підходить для порівняння, оскільки незрозуміло, який підхід покращує яку частину процесу тестування. Остання класифікація більш обґрунтована, оскільки підходи згруповані за призначенням. Крім того, оскільки критерії покриття використовуються як умова зупинки для фази генерації тестової моделі, ці дві дії належать до однієї фази тестування. Тому в цьому дослідженні ми класифікуємо дослідження на дві категорії: створення тестової моделі та розробка функцій оракула.

Існує значна кількість запропонованих підходів, які зосереджені на автоматичному створенні тестової моделі. Ці підходи до створення тестової моделі можна класифікувати за трьома категоріями: критерії охоплення тестом, аналіз мутацій і розподіл еквівалентності на основі різних оцінок якості тесту.

Таблиця 1.1 показує результати першого скринінгу досліджень, спрямованих на досягнення вибраних критеріїв охоплення тестом. Як видно з

таблиці 1.1, більшість тестового покриття є покриттям метамоделі та покриттям трансформації.

Таблиця 1.1.

Результат аналізу джерел. Створення тесту з критеріями покриття

Title	Test Criteria	Model Transformation Language	Used Technologies	Evaluation for the Approach
Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool[5]	Metamodel coverage	Not mentioned	MOF, OCL	Mutation analysis
Automatic model generation strategies for model transformation testing[6]	Metamodel coverage	Not mentioned	Alloy	Mutation analysis
Formal specification and testing of model transformations[3]	Not mentioned	ATL, QVT, RubyTL, JTL	ASSL language, USE	Correctness of output models checked by using USE
ATLTest: A White-Box Test Generation Approach for ATL Transformations[26]	Transformation coverage	ATL	OCL	Not mentioned
TETRABox - A Generic White-Box Testing Framework for Model Transformations[52]	Transformation coverage	Not mentioned	PAMOMO, QVT-Relations, OCL	Only observations
A Search Based Test Data Generation Approach for Model Transformations[38]	Transformation coverage	ATL	MOTTER	Not mentioned
Specification-driven model transformation testing[31]	Specification coverage	ATL/ETL	PAMOMO, Eclipse, Z3 solver, USE, OCL	Mutation analysis
Translating target to source constraints in model-to-model transformations[14]	Metamodel coverage	ATL, ETL, QVT, TGGs	OCL, anATLyzer	Measured solving time and total time, precision and recall metrics
On Analyzing Rule-Dependencies to Generate Test Cases for Model Transformations[46]	Rule-dependency coverage	TGG	OCL, USE framework and USE model validator	Rule dependencies

В [5] запропонували алгоритм для створення набору тестів, який може задовольнити покриття метамоделі за допомогою інструменту для створення тестів під назвою OMOGEN. В [15, 16] запропонували інструмент під назвою Cartier для створення тестових моделей для досягнення критеріїв охоплення метамоделями. Інструмент Cartier може використовувати обмеження OCL для метамоделей Ecore. Обмеження OCL для метамоделей Ecore використовуються для вираження інваріантів і моделювання попередніх умов перетворення. Потім Cartier викликає Alloy API , а потім запускає розв’язувач SAT для створення моделей. Однак обмеження OCL потрібно перетворити на факти Alloy вручну. У дослідженні [6] було прийнято дві стратегії на основі

розділу вхідної області як критерії охоплення метамоделі з [18] для керівництва створенням моделі, які називаються критеріями всіх діапазонів і критеріями всіх розділів. В [13] запропонували використовувати інваріанти OCL як специфікації перетворення та використовувати ASSL (A Snapshot Sequence Language) для створення тестових моделей. Тестові моделі можуть або задовольняти всім обмеженням, або порушувати принаймні одне обмеження. Але критерії покриття специфікацій не обговорювалися в [13]. Фреймворк TETRABox [52] також вимагає формалізованих вимог трансформації. TETRABox може створити набір тестів, який забезпечує певний рівень охоплення трансформації. В [38] запропонували підхід на основі OCL для генерації тестової моделі, який прийняв критерії покриття трансформації. В [30] представили візуальну декларативну мову під назвою PAMOMO для визначення поведінкових контрактів. Вони також представили PAMOMO Contract-Checker, який допомагає компілювати PAMOMO у QVT-Relations. Базуючись на мові PAMOMO, в [31] запропонували структуру під назвою тестування трансформації моделі на основі специфікації. У цій системі тестування вимоги формалізовані в конкретні параметри в PAMOMO. Специфікації PAMOMO можуть автоматично отримувати функції оракула та тестувати моделі. Згенеровані тестові моделі можуть охоплювати всі специфікації за допомогою методів вирішення SAT. У цьому дослідженні також запропоновано 7 рівнів критеріїв охоплення специфікації.

В [14] представили метод, який також можна використовувати для створення тестових моделей з урахуванням обмежень цільових моделей. Цей метод перетворює обмеження OCL у вихідну мета-модель за допомогою інструменту anATLyzer. В [32] доводять, що залежності правил безпосередньо впливають на властивості якості перетворення моделі для TGG. TGG — це формальна, декларативна, двонаправлена мова перетворення моделі [41, 33]. Ґрунтуючись на цьому доказі, в [46] запропонували метод для виявлення залежності правил TGG і автоматизації генерації тестової моделі на основі покриття залежностей правила.

Дослідження аналізу мутацій у тестуванні трансформації моделі зосереджувалися на автоматичному створенні кваліфікованих мутантів і зменшенні витрат на обчислення. Однак більшість підходів до аналізу мутацій залежать від мови, оскільки визначення оператора мутації вимагає знання мови. Тому техніка аналізу мутацій, яка використовується для тестування трансформації моделі, не настільки розвинена, як та, що використовується для C, C++ і Java. В [50] запропонували синтаксичні оператори, які можуть створювати, видаляти або оновлювати елементи метамоделі ATL. Однак оператори синтаксичної мутації не імітують реальні помилки розробника.

1.5.2 Тестування трансформації M2T

Таблиця 1.2 показує результат першого скринінгу досліджень тестування трансформації M2T. В [27] запропонували архітектуру систематичного тестування для генераторів коду. Інструмент ModeSSa генерує тестові моделі, націлені на оптимізацію генератора коду. В [28] запропонували структуру модульного тестування як для трансформації M2M, так і для M2T. В [20] запропонували структуру трансформації моделі для тестування веб-додатків на основі моделі (різновид трансформації M2T). Ця стаття присвячена вирішенню проблем із середовищем тестування, пов’язаних із цією структурою. В [28] запропонували визначення трьох критеріїв адекватності на основі конкретного синтаксису вхідних моделей. Дослідження [28] також класифікувало тести трансформації на три категорії: тести на відповідність, семантичні тести та текстові тести. В [10] запропонували автоматизований підхід до тестування під назвою CCUJ, щоб перевірити, чи відповідають реалізації Java їхнім моделям класу UML. В [19] запропонували структуру модульного тестування для трансформації U-OWL M2T. В дослідженні [21] запропонували інструмент налагодження під назвою HandyMOF для перетворення MOFScript. Хоча він зосереджений на

налагодженні, інструмент можна використовувати для вимірювання покриття трансформації, отриманого набором тестових моделей.

Таблиця 1.2.

Результат аналізу джерел. Тестування трансформації M2T

Title	Test Criteria	Model Transformation Language/Tool	Used Technologies	Evaluation for the Approach
Systematic Testing of Model-Based Code Generators[7]	Model coverage and code coverage	TargetLink	ModeSSa, UML, TargetLink	Not mentioned
Unit Testing Model Management Operations[8]	Not mentioned	EGL	Epsilon, EUnit	Not mentioned
Multi-level Testes for Model Driven Web Applications[20]	Not mentioned	Groovy	BPMN, WebML, Canoo	Not mentioned
A Method for Testing Model to Text Transformations	Conformance, textual, semantic	Acceleo	UML	Not mentioned
An Approach to Testing Java Implementation against Its UML Class-Model[10]	Branch-coverage	RSA	OCL, UML, RSA, Java	Compared with other approaches
Unit Testing of Model to Text Transformations	Not mentioned	U-OWL	MeDMoT, OWL	Not mentioned
Testing MOFScript Transformations with HandyMOF[21]	Transformation coverage	MOFScript	Eclipse,	Not mentioned
Back-To-Back Testing of Model-Based Code Generators[39]	Not mentioned	Genesys	Genesys	Not mentioned
Testing M2M/M2T/T2M Transformations[6]	Constraint/rule/relatedness of constraints and rules coverage	ATL	TractsTool, Matching Tables Builder, USE	Precision and recall

В [28] запропонували загальну метамодель для тексту для перетворення проблеми специфікації трансформації M2T/T2M у задачу специфікації трансформації M2M. Ґрунтуючись на цьому дослідженні, в [6, 8] розширили підхід трактів із тестування трансформації M2M і визначили механізм, заснований на відповідних таблицях. Таблиці відповідності вирівнюють реалізацію перетворення моделі з її трактами, які є характеристики цього перетворення моделі. Таблиці відповідності можуть як виявити, так і знайти помилку в перетворенні. В [39] запропонували підхід послідовного тестування для генератора коду Genesys. Основна ідея послідовного тестування полягає у виконанні вхідної моделі та згенерованого нею коду та перевірці, чи результати обох виконань однакові. Якщо виходи

однакові, це означає, що семантика вхідних моделей збережена. Це вимагає, щоб вхідні моделі могли бути виконані інструментом виконання моделі. Однак це не завжди доступно.

1.5.3. Визначення пріоритетів підходів

На цьому кроці ми визначаємо пріоритетність підходів з останнього розділу, враховуючи інтереси всіх зацікавлених сторін і здійсненність досліджень. Таблиця 1.3 показано пріоритетний список 11 досліджень. Через обмеження часу ми можемо детально реалізувати лише 2 підходи. Спочатку ми зосередимося на підходах 1 і 2, а інші підходи в списку будуть альтернативними варіантами.

Таблиця 1.3.

Представлення пріоритетності підходів

Order	Keywords	Year	Title	Used Technologies
1	tracts	2015	Testing M2M/M2T/T2M Transformations [6]	TractsTool, Matching Tables Builder, USE
2	classifying terms	2015	Employing classifying terms for testing model transformations [25]	ASSL, USE, OCL
		2017	Testing transformation models using classifying terms [7]	OCL, USE
		2018	Testing models and model transformations using classifying terms [6]	USE, OCL, ASSL, Kodkod
3	specification-driven testing	2015	Specification-driven model transformation testing [31]	ocl2smt, PAMOMO, Eclipse, Z3 solver, USE, OCL
4	source constraint analysis	2014	Test data generation for model transformations combining partition and constraint analysis [27]	OCL, OCLBBTesting, EMF2CSP
5	transformation coverage	2014	A Search Based Test Data Generation Approach for Model Transformations[38]	MOTTER
6		2013	TETRABox - A Generic White-Box Testing Framework for Model Transformations[52]	PAMOMO, QVT-Relations, OCL
7		2012	ATLTest: A White-Box Test Generation Approach for ATL Transformations[26]	OCL
8	metamodel coverage	2009	Automatic model generation strategies for model transformation testing[56]	Alloy
9		2006	Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool[5]	MOF, OCL

Підхід 1 спрямований на тестування перетворень за допомогою трактів. Трактати можуть вказувати, чого очікувати в цільовій моделі відповідно до вихідної моделі за допомогою OCL. Це відповідає потребі у підвищенні якості тестування.

Підхід 2 запропонував і використав ідею класифікації термінів для автоматичного створення набору моделей. Цей підхід вирішив проблему ручного моделювання. Крім того, це вирішує проблему зайвого тестування. Терміни класифікації визначають розділи згенерованих моделей. Кожен розділ матиме рівно одну задовільну модель.

Якщо підходи 1 і 2 виявляться абсолютно нездійсненними під час раннього впровадження (наприклад, інструмент пошкоджено або наданий приклад не працює), то можна застосувати наступний підхід у списку.

Висновки до розділу

У першому розділі проведено аналіз сучасних підходів до розробки програмного забезпечення (ПЗ) на основі моделей, зосереджуючись на процесах трансформації моделі (model transformation) та тестуванні цих процесів. У дослідженні було описано предметну область розробки ПЗ з використанням моделей, основні поняття, що стосуються трансформацій моделей, а також проведено огляд літератури, яка розглядає різні аспекти цих трансформацій.

Було виявлено, що трансформація моделей є ключовим етапом у моделі-орієнтованій розробці, яка дозволяє автоматизувати процеси генерації коду та забезпечити узгодженість між моделями різних рівнів абстракції. Тестування трансформацій виявляється критично важливим для забезпечення правильності та якості кінцевого ПЗ, тому особливу увагу приділено аналізу підходів до тестування як трансформацій «модель-модель» (M2M), так і трансформацій «модель-текст» (M2T).

Окремо розглянуто питання пріоритетизації підходів до трансформацій, що дозволяє ефективніше планувати процес розробки ПЗ на основі моделей. Аналіз літератури продемонстрував різноманіття методів тестування та їх еволюцію, що свідчить про активний розвиток цієї сфери.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ МЕТОДІВ ТА ІНСТРУМЕНТІВ ОЦІНКИ ТЕСТУВАННЯ ТРАНСФОРМАЦІЇ МОДЕЛЕЙ ІМПЛЕМЕНТАЦІЇ

2.1. Дослідження інструменту для формальної перевірки трансформацій моделей TractsTool

TractsTool - це інструмент, який перевіряє правильність реалізації трансформації M2M за допомогою трактатів. Трактати визначають, чого очікувати в цільовій моделі відповідно до вихідної моделі. Трактати задаються в OCL.

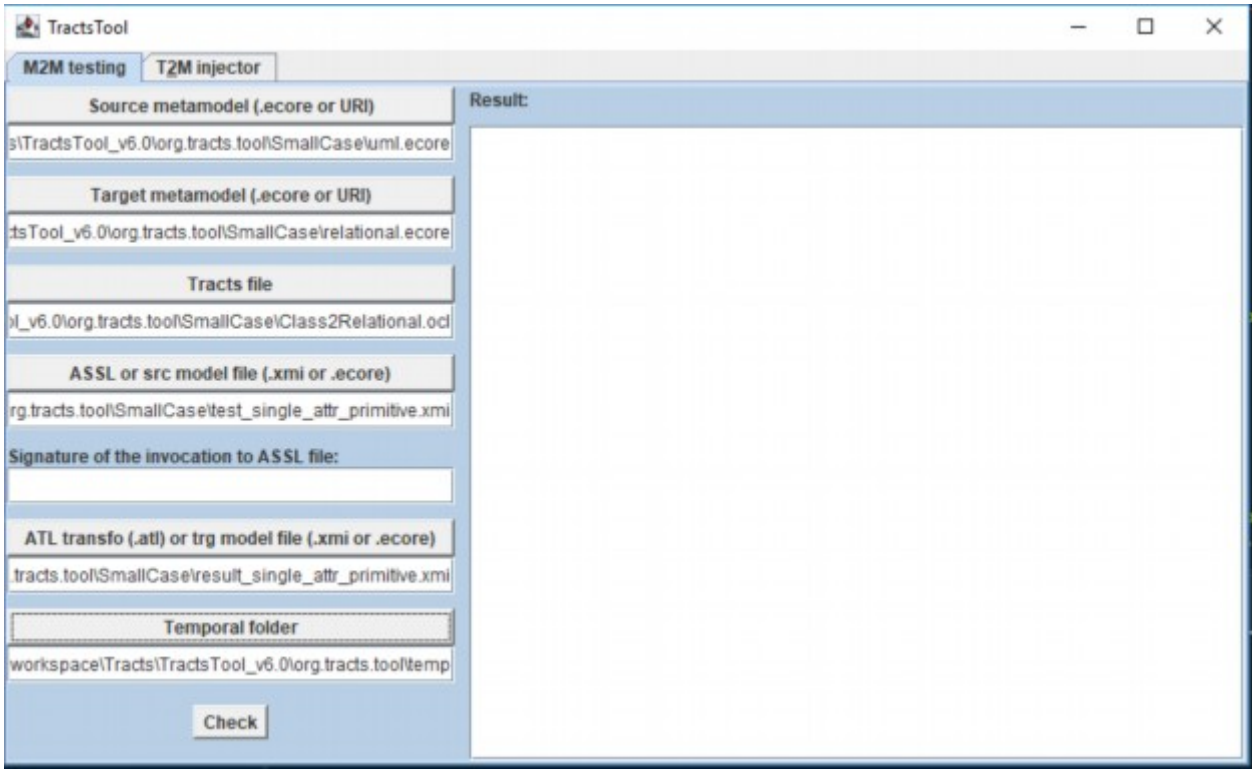

Рис. 2.1. Графічний інтерфейс інструменту TractsTool

Рисунок 2.1 показує графічний інтерфейс TractsTool. TractsTool є незалежним від мови. Щоб тестувати трансформації ATL, TractsTool потрібно приймати вихідні та цільові метамоделі, трактати, трансформацію ATL та сценарій ASSL як вхідні дані. Для трансформацій ATL TractsTool може генерувати вихідні моделі за допомогою сценарію ASSL. Для не-ATL

трансформацій TractsTool потрібно приймати вихідні та цільові метамоделі, вихідні та цільові моделі як вхідні дані. Основна ідея TractsTool полягає в тому, щоб перетворити вихідну метамодель і цільову метамодель в один єдиний файл специфікації USE (UML-based Specification Environment). TractsTool також потрібно генерувати вихідні та цільові моделі в USE. TractTool використовує інструмент USE як двигун для оцінки OCL-виразів у трактатах і перевіряє, чи моделі задовольняють трактати. TractsTool потім поверне результати оцінки.

TractsTool — це спеціалізований інструмент, розроблений для перевірки правильності реалізації трансформацій типу "модель-модель" (M2M). Цей інструмент орієнтований на моделювання та автоматичне тестування процесу перетворення моделей у межах розробки програмного забезпечення.

Наведемо основні характеристики TractsTool:

- Автоматична валідація трансформацій: TractsTool забезпечує автоматичну перевірку перетворення моделей для виявлення невідповідностей між вихідною моделлю та результатом трансформації. Це дозволяє знизити ризик помилок у програмних продуктах, що створюються на основі моделі-орієнтованої архітектури.

- Правильність трансформацій: Інструмент перевіряє, чи відповідає результат трансформації встановленим правилам та вимогам. Він також може порівнювати різні моделі з точки зору їх семантичної відповідності та структурних властивостей.

- Сценарії тестування: TractsTool дозволяє створювати та запускати різні сценарії тестування для трансформацій M2M. Це включає перевірку таких аспектів, як правильність синтаксичних перетворень, відповідність атрибутів та коректність відображення взаємозв'язків між елементами моделей.

- Відслідковування змін: Один з ключових аспектів роботи з TractsTool полягає у можливості відстежувати еволюцію моделей у часі та визначати, чи зміни в одній моделі правильно відображаються у трансформованій моделі.

- Гнучкість та розширюваність: TractsTool підтримує різні формати моделей та може бути інтегрований з іншими інструментами для моделювання. Це забезпечує його застосування в різних проєктних середовищах та підходах до розробки ПЗ.

Таким чином, TractsTool є потужним інструментом для забезпечення якості програмних систем, розроблених на основі моделей, шляхом автоматизації перевірок і гарантій правильності трансформацій M2M.

Далі буде представлено приклад застосування інструменту до невеликого перетворення під назвою UML2Relational, а також буде представлено оцінку придатності TractsTool.

2.1.1. Представлення випадку перетворення UML2Relational

Ми визначили одну вихідну метамодель під назвою UML, одну цільову метамодель під назвою Relational і перетворення QVTo. Цей невеликий випадок використовувався в невеликих прикладах чотирьох інструментів. Вихідна мова — це невеликий набір UML, а цільова — проста реляційна база даних.

Трансформація моделі UML у модель реляційної бази даних (РБД) за допомогою мови QVTo (QVT Operational) виконується на основі певних правил, що визначають, як елементи однієї моделі (UML) відображаються на елементи іншої моделі (РБД). Трансформація UML моделі в модель реляційної бази даних за допомогою QVTo є чітко регламентованим процесом, де кожен елемент UML відображається на відповідний елемент структури реляційної БД згідно з вищезазначеними правилами. Це дозволяє забезпечити правильність і узгодженість моделей, а також автоматизувати процес переходу між ними.

Трансформація QVTo може перетворити модель UML на модель реляційної бази даних. Правила перетворення визначені в перетворенні QVTo, показаному у фрагментів коду в лістингу 2.1.

Лістинг 2.1. Фрагмент коду перетворення QVTo для UML2Relational

```
1  modeltype UML uses 'http://www.example.org/uml';
2  modeltype RELATIONAL uses 'http://www.example.org/relational';
3
4  transformation class2relational(in uml : UML, out RELATIONAL);
5
6
7  main() {
8
9    uml.rootObjects()[UML::Model]->map model2schema();
10 }
11 mapping UML::Model::model2schema() : RELATIONAL::Schema {
12   type := self.type->map type2type();
13   table := self.classes->map class2table();
14   table += (self.classes.attribute
15     ->select(e | e.multivalued and e.classifier.oclIsKindOf(Classes)))
16     ->map multiClassAttribute2table()
17   ->union(self.classes.attribute
18     ->select(e | e.multivalued and e.classifier.oclIsKindOf(Type))
19     ->map multiPrimAttribute2table());
20 }
```

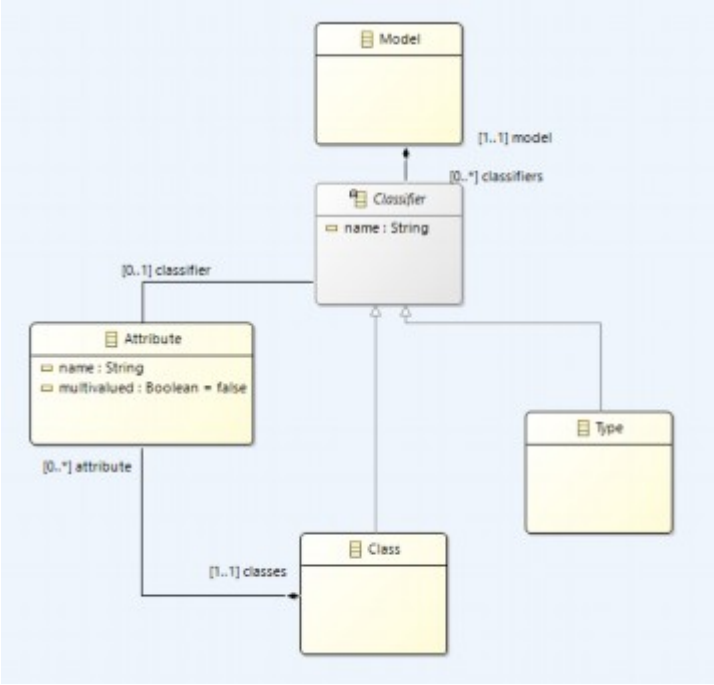

Рис. 2.2. Реалізація uml.ecore в Eclipse

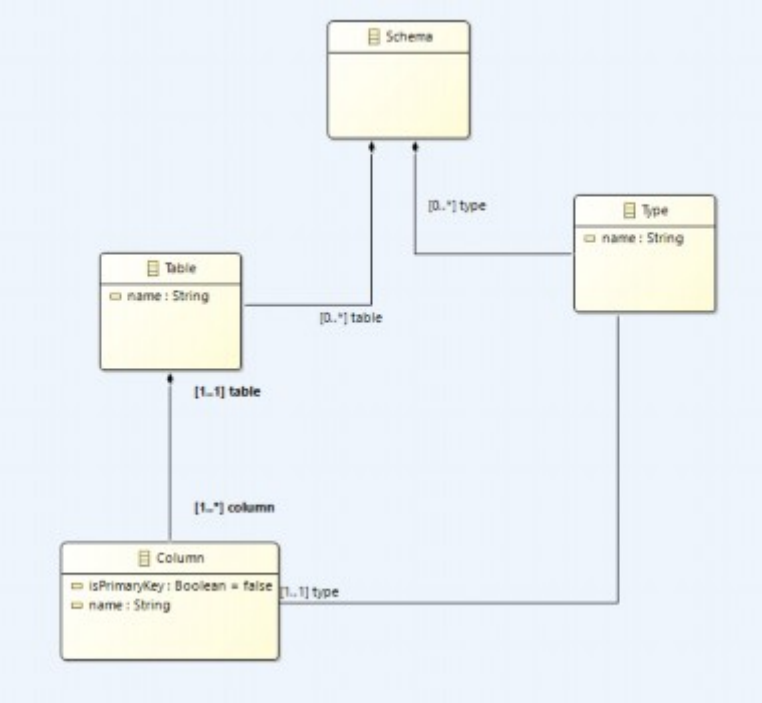

Рис. 2.3. Реалізація relational.ecore в Eclipse

2.1.2. Реалізація прикладу UML2Relational за допомогою TractsTool

Спочатку потрібно перевірити, чи може цей інструмент працювати без помилок, і виявити помилку в цьому перетворенні QVTo. Для цього потрібно написати трактати, щоб охопити логіку правил відображення, а потім додати дефекти в цільову модель. Засіб має дати негативний результат.

Щоб спростити проблему, правило під назвою type2type() у перетворенні QVTo береться як приклад (код наведено в лістингу 2.2). Це правило трансформації відображає кожен примітивний тип у вихідній моделі до примітивного типу в цільовій моделі.

Лістинг 2.2. Правило трансформації type2type в UML2Relational.qvto

```
1 mapping UML::Type::type2type() : RELATIONAL::Type{
2     name := self.name;
3 }
```

Таким чином, у файлі tracts (код наведено в лістингу 3.3) ми визначаємо, що всі екземпляри вихідних типів повинні знаходити принаймні

один тип із таким самим ім'ям у цільовій моделі. Назви елементів у тракті відрізняються від правила відображення QVTo. Щоб відрізнити вихідну метамодель від цільової, TractsTool додасть прексиси до імен елементів в обох метамоделях. Наприклад, TractsTool перейменує Type у вихідній метамоделі на src Type, а Type у цільовій метамоделі на trg Type. Тракт для type2type() показаний у лістингу 2.3.

Лістинг 2.3. Тракт для type2type()

```
1 context src_Type inv Type2Type:
2 src_Type.allInstances->forAll(t_src|trg_Type.allInstances->exists(t_trg|
3 t_trg.name = t_src.name))
```

Рисунок 2.4 показує вихідну модель і цільову модель. Як видно на рис. 2.5, типи в цільовій моделі (relational.xmi) відрізняються від типів у вихідній моделі (uml.xmi). Ми навмисно додали цей недолік і очікували, що TractsTool зможе виявити його.

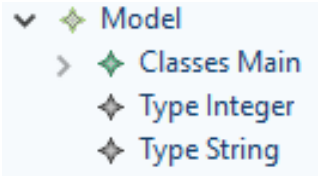

Рис. 2.4. Реалізація uml_model.xmi в Eclipse

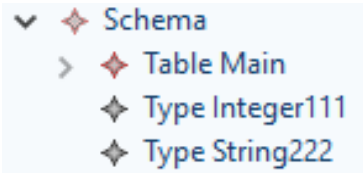

Рис. 2.5. Реалізація relational_model.xmi shown в Eclipse

Результат на рисунку 2.6 показує, що TractsTool не вдалося виявити помилку. В лістингу 2.4 показано згенеровані команди. Ці команди призначені для перетворення вихідної моделі та цільової моделі в моделі в

USE. Проаналізувавши команди, створені інструментом, ми виявили, що інструмент викликав USE та використовував команди для створення 2 типів джерела та 2 типів цілей. Два типи джерел мають назви ”Type_102” і ”Type_103”. Два цільових типу називаються ”Type_108” і ”Type_109”.

Лістинг 2.4. Згенеровані команди

```
!new src_Model ('Model_99')
!new src_Classes ('Classes_100')
!new src_Attribute ('Attribute_101')
!new src_Type ('Type_102')
!new src_Type ('Type_103')
!insert (@Model_99,@Classes_100) into src_classes_Classes_Model
!insert (@Model_99,@Type_102) into src_type_Model_Type
!insert (@Model_99,@Type_103) into src_type_Model_Type
!@Classes_100.name := null
!insert (@Classes_100,@Attribute_101) into src_attribute_classes_Attribute_Classes
!@Attribute_101.name := null
!@Attribute_101.multivalued := null
!insert (@Attribute_101,@Type_103) into src_classifier_Attribute_Classifier
!@Type_102.name := null
!@Type_103.name := null
!new trg_Schema ('Schema_104')
!new trg_Table ('Table_105')
!new trg_Column ('Column_106')
!new trg_Column ('Column_107')
!new trg_Type ('Type_108')
!new trg_Type ('Type_109')
!insert (@Schema_104,@Table_105) into trg_table_Schema_Table
!insert (@Schema_104,@Type_108) into trg_type_Schema_Type
!insert (@Schema_104,@Type_109) into trg_type_Schema_Type
!@Table_105.name := null
!insert (@Table_105,@Column_106) into trg_column_table_Column_Table
!insert (@Table_105,@Column_107) into trg_column_table_Column_Table
!@Column_106.isPrimaryKey := null
!@Column_106.name := null
!insert (@Column_106,@Type_108) into trg_type_Column_Type
!@Column_107.isPrimaryKey := null
!@Column_107.name := null
!insert (@Column_107,@Type_109) into trg_type_Column_Type
!@Type_108.name := null
!@Type_109.name := null
check -d
```

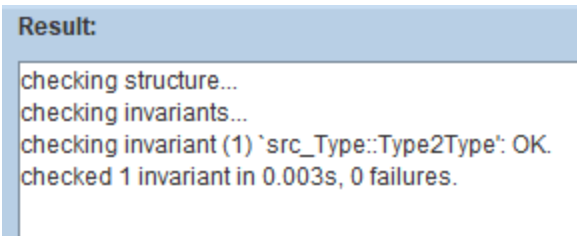

Рис. 2.6. Результат роботи TractsTool

Проте всі назви вихідного та цільового типів є «нульовими». Ми відстежили команди, створені TractsTool. Ми виявили, що команди присвоюють значення null усім атрибутам. Це причина, чому тип джерела та цільовий тип вважалися однаковими, і інструмент не зміг виявити помилку трансформації. Це також означає, що TractsTool не може також виявляти інші види помилок, оскільки він встановлює кожне значення атрибута як «null».

2.1.3. Опис результатів роботи інструменту TractsTool

Розглянемо основні функції описаного інструменту.

- Здатність TractsTool виявляти помилки залежить від якості трактів, наданих тестувальниками. Проблема, з якою стикаються тестувальники в галузі, полягає в тому, що під час тестування трансформації моделі не виявляється неприйнятна кількість помилок трансформації. Це означає, що люди-тестери не в змозі створити комплексний і кваліфікований набір тестів, щоб можна було виявити всі недоліки. Проте здатність TractsTool виявляти помилки повністю залежить від якості (наприклад, повноти та правильності) трактатів, наданих тестувальниками. Якщо користувач створює простий файл тракту, TractsTool поверне позитивні результати. Але це не обов’язково означає, що тестована трансформація вільна від помилок. Таким чином, здатність TractsTool виявляти помилки залежить від якості трактатів, наданих тестувальниками.

- Непослідовне найменування у файлі тракту. Існує проблема невідповідності імен у файлі тракту. Фрагмент згенерованого файлу USE для випадку UML2Relational показано в лістингу 3.5.

Лістинг 2.5. Фрагмент згенерованого файлу USE для випадку UML2Relational

```
1  model uml
2
3  class src_Model
4  attributes
5  end
6
7  class src_Classes < src_Classifier
8  attributes
9  end
10
11 class src_Attribute
12 attributes
13  name : String
14  multivalued : Boolean
15 end
16
17 class src_Type < src_Classifier
18 attributes
19 end
20
21 class src_Classifier
22 attributes
23  name : String
24 end
25
26 class trg_Table
```

Він додає префікс до назви кожного елемента в метамоделі Ecore, щоб розрізняти вихідну метамодель і цільову метамодель. Тому тестувальники повинні бути обережними щодо цієї різниці в іменуванні між створеними специфікаціями USE та метамоделями Ecore під час написання трактату. Це проблема з практичної точки зору, але її можна вирішити шляхом подальшого розвитку.

TractsTool все ще містить дефекти. TractsTool створить командний файл для створення екземплярів середовища виконання в USE відповідно до

вихідних і цільових моделей. Однак він призначив значення null для всіх атрибутів. Таким чином, поточна версія TractsTool все ще містить дефекти.

TractsTool може вирішити проблему оракула в M2M-перетвореннях, функція Oracle порівнює згенеровану цільову модель з очікуваною моделлю. TractsTool може бути корисним, коли немає доступної очікуваної моделі або доступна очікувана модель ненадійна. Очікувана модель перевіряється вручну під час тестування. Ця модель буде використана як очікуваний результат трансформації. Але перевіряється тільки вручну. Однак, якщо тестувальник помиляється і приймає неправильну модель як очікувану, весь процес тестування матиме проблеми, а помилки в перетворенні можуть залишитися непоміченими. Таким чином, TractsTool може відігравати роль у офіційній перевірці моделей, щоб тестування було менш схильним до помилок.

2.2. Дослідження функціональності інструменту тестування трансформації моделі

2.2.1. Опис особливостей інструменту Matching Table Builder

Matching Table Builder (MTB) — це інструмент, який використовується для тестування трансформацій моделі, що залежить від ATL (Atlas Transformation Language) — однієї з найпопулярніших мов для моделі-орієнтованих трансформацій (Model-to-Model, M2M). Основною метою MTB є перевірка коректності та узгодженості результатів трансформацій між вихідною і цільовою моделями.

Основні характеристики Matching Table Builder (MTB):

- Залежність від ATL: MTB призначений для використання в контексті трансформацій, створених за допомогою ATL. Він інтегрується з ATL-трансформаціями, забезпечуючи автоматичне тестування цих процесів і створення таблиць відповідностей між елементами вихідної та цільової моделей.

- Таблиці відповідностей (Matching Tables): Основна функція MTB полягає у створенні таблиць відповідностей, які фіксують відповідність між елементами вихідної та цільової моделей. Це дозволяє визначити, які елементи вихідної моделі перетворюються в які елементи цільової моделі після застосування трансформації.

- Автоматичне зіставлення елементів: MTB автоматично аналізує вихідну та цільову моделі, зіставляючи елементи відповідно до правил трансформації, визначених у ATL. Це дозволяє відслідковувати всі перетворення і виявляти помилки або невідповідності.

- Тестування коректності трансформацій: Інструмент перевіряє, чи правильно відображаються всі елементи вихідної моделі в цільовій моделі згідно з правилами трансформації. Це дозволяє підтвердити, що трансформація була виконана правильно, і жоден важливий елемент не було пропущено.

- Підтримка складних трансформацій: MTB може працювати з трансформаціями різного рівня складності, від простих M2M перетворень до складних багатоступеневих трансформацій, що включають множинні правила та взаємозв'язки між елементами.

- Логування та звітність: Після завершення процесу тестування MTB генерує звіт, який містить інформацію про результати трансформації, успішність зіставлень і можливі помилки. Це допомагає розробникам і тестувальникам легко ідентифікувати проблеми та коригувати їх.

- Підтримка інтеграцій з іншими інструментами: MTB можна інтегрувати з іншими інструментами для моделі-орієнтованої розробки, такими як Eclipse Modeling Framework (EMF) або інші засоби для побудови моделей. Це підвищує його гнучкість і зручність у великих проєктах.

Переваги Matching Table Builder:

- Швидке тестування: автоматизує процес перевірки трансформацій, значно зменшуючи час на ручне тестування.

- Надійність: забезпечує точну відповідність між вихідною та цільовою моделями, допомагаючи уникати помилок під час розробки трансформацій.

- Гнучкість: підтримує різні типи трансформацій, що дозволяє використовувати його в різних проєктних середовищах.

Matching Table Builder (MTB) — це потужний інструмент для тестування трансформацій моделей, побудованих на основі ATL. Він дозволяє автоматизувати процес перевірки відповідностей між елементами вихідної та цільової моделей, підвищуючи точність і ефективність тестування. Завдяки своїй інтеграції з ATL і підтримці складних трансформацій, MTB є важливим інструментом для забезпечення якості в моделі-орієнтованій розробці.

Як і TractsTool, метою MTB також є перевірка правильності трансформацій. Логіка перетворень також повинна бути виражена в обмеженнях OCL. Ці обмеження подібні до трактів у TractTool. Від TractsTool відрізняється те, що він не створює моделі в USE. Замість цього він вимагає, щоб користувач спочатку витягнув інформацію про вихідну та цільову метамоделі, які називаються типами, за допомогою екстрактора типів перетворень ATL (показано на рисунку 2.7).

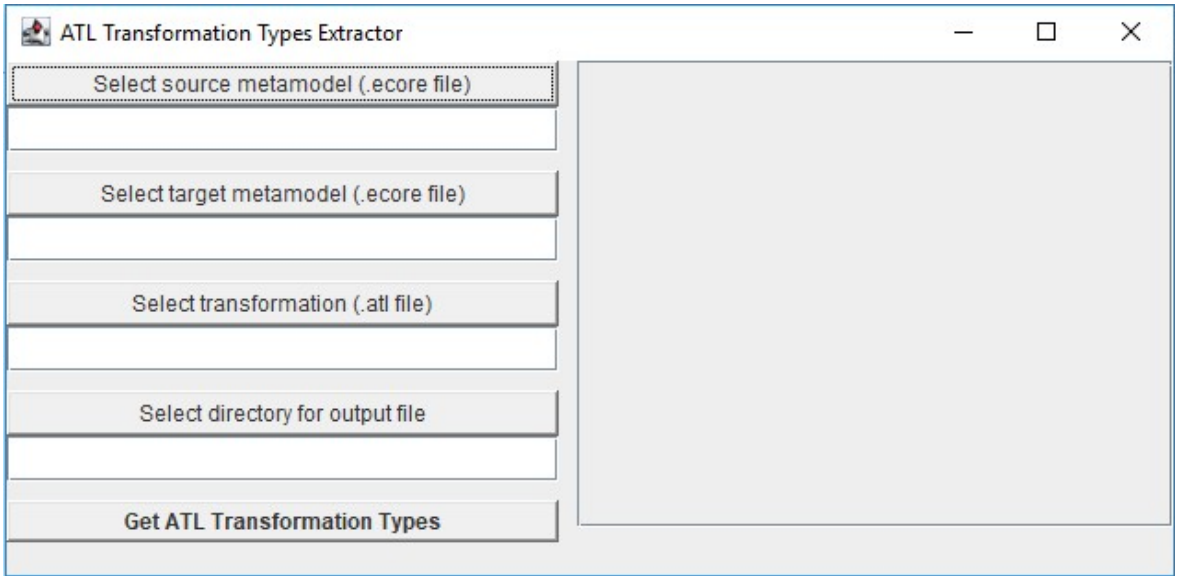

Рис. 2.7. Графічний інтерфейс екстрактора типів трансформації ATL

Оскільки екстрактор типів перетворень ATL може працювати лише для перетворень ATL, ми не можемо безпосередньо застосувати його до наших перетворень QVTo. Тоді інформацію про вихідну та цільову метамоделі слід прийняти як вхідні дані MTB. Графічний інтерфейс користувача показаний на рисунку 2.8. MTB перевірить правила ATL відповідно до обмежень OCL.

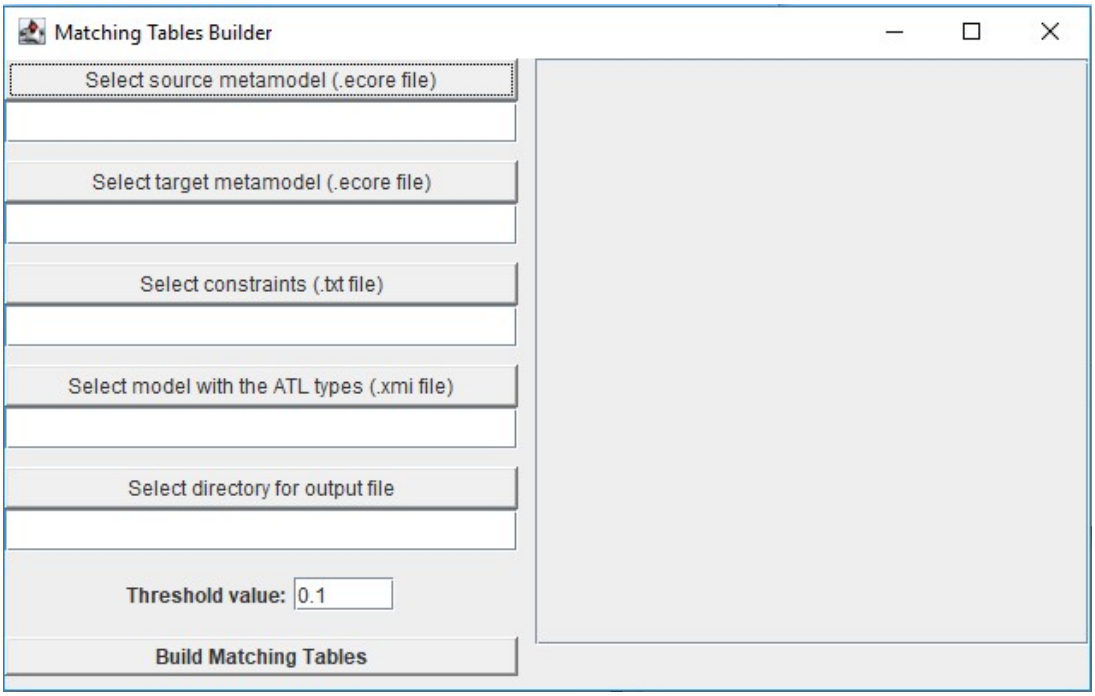

Рис. 2.8. Графічний інтерфейс Matching Table Builder

2.2.2. Представлення можливостей інструменту

Розглянемо основні функціональні можливості Matching Table Builder:

- MTB працює з перетвореннями ATL, але немає концептуальних і технічних перешкод для його адаптації до QVTo. В інструменті ASOME мовою перетворення M2M є QVTo. Тому ми не можемо провести ні просте тематичне дослідження, ні промислове дослідження. MTB не може бути безпосередньо застосований до не-ATL перетворень, але в принципі може бути адаптований до QVTo, оскільки ми маємо сліди, створені в результаті виконання перетворення. З цих слідів ми можемо отримати типи трансформації QVTo.

- Здатність MTB виявляти несправності залежить від якості обмежень OCL. У контексті тестування трансформації моделі метою тестування є виявлення помилок у трансформаціях моделі. Інструмент повинен допомогти тестувальнику виявити недоліки в трансформаціях, які важко виявити. Проте те, чи зможе MTB виявити помилки в перетворенні, повністю залежить від якості обмежень OCL. Якщо тестер пише просте обмеження OCL, MTB повернеться результати як позитивні, тоді як перетворення все ще містить помилки. MTB може перевірити, чи очікує перетворення тестер.

Подібно до TractsTool, MTB використовує обмеження OCL для офіційної перевірки перетворень. Це може бути рішенням проблем оракула, але потрібна адаптація інструменту.

Висновки до розділу

У другому розділі було проведено детальне дослідження методів та інструментів для оцінки якості тестування трансформацій моделей. У фокусі дослідження знаходилися два важливі інструменти — TractsTool та Matching Table Builder (MTB), кожен з яких пропонує різні підходи до формальної перевірки трансформацій моделей.

Дослідження інструменту TractsTool показало, що цей інструмент є ефективним для формальної перевірки трансформацій типу модель-модель (M2M). Він дозволяє не тільки перевіряти коректність трансформацій, але й створювати приклади реальних перетворень, таких як UML2Relational, для перевірки відповідності результатів встановленим правилам. Використання TractsTool допомагає виявляти та виправляти невідповідності у перетвореннях, що значно покращує якість процесу розробки програмного забезпечення.

Застосування Matching Table Builder (MTB) продемонструвало його спроможність перевіряти відповідності між елементами вихідної та цільової моделей, що було особливо корисним у контексті тестування трансформацій,

побудованих на основі ATL. Інструмент автоматизує процес зіставлення та генерує таблиці відповідностей, що дозволяє розробникам ефективно перевіряти результати трансформацій і знижувати кількість помилок у моделі-орієнтованих проєктах.

Таким чином, дослідження цих інструментів підтвердило їхню важливість у забезпеченні якості трансформацій моделей. TractsTool і MTB є ключовими компонентами в системі оцінки та тестування трансформацій, допомагаючи автоматизувати перевірки, підвищувати точність трансформацій та знижувати ризики, пов’язані з некоректними перетвореннями моделей у процесі розробки програмного забезпечення.

РОЗДІЛ 3. ЗАСТОСУВАННЯ МЕТОДІВ ТА ІНСТРУМЕНТІВ ОЦІНКИ ТЕСТУВАННЯ ГЕНЕРАЦІЇ І ТРАНСФОРМАЦІЇ МОДЕЛЕЙ ІМПЛЕМЕНТАЦІЇ

В цьому розділі виконаємо дослідження та оцінку двох інструментів, які реалізують автоматичну генерацію моделі для розділення еквівалентності. Спочатку буде представлено USE, а потім Efinder, що розширює USE та робить його більш придатним для Eclipse Modeling Framework.

3.1. Середовище специфікацій на основі UML

Середовище специфікацій на основі UML (USE) — це інструмент, призначений для формальної верифікації та аналізу моделей UML (Unified Modeling Language) та специфікацій, визначених у вигляді OCL (Object Constraint Language). USE забезпечує платформу для моделювання та перевірки коректності специфікацій, дозволяючи користувачам створювати моделі UML та перевіряти їх за допомогою правил, обмежень і операцій, визначених мовою OCL.

Основні характеристики USE:

- Формальна верифікація специфікацій: USE дозволяє користувачам створювати UML-моделі та перевіряти їх відповідно до формальних специфікацій, визначених у OCL. Це забезпечує надійний спосіб перевірки коректності моделей та виявлення потенційних проблем або невідповідностей у ранніх етапах розробки.

- Підтримка мови OCL: OCL є мовою для визначення обмежень та правил поведінки елементів UML-моделі. USE надає користувачам можливість написання OCL-виразів для верифікації таких аспектів, як інваріанти класів, перед- та постумови операцій, а також інших специфікацій.

- Інтерактивна перевірка моделей: Інструмент дозволяє взаємодіяти з моделями у реальному часі. Користувачі можуть створювати об’єкти UML-

класів, встановлювати атрибути, встановлювати зв’язки між об’єктами та перевіряти, чи дотримуються всі обмеження, визначені в OCL.

- Тестування сценаріїв: USE підтримує можливість тестування моделей за допомогою різних сценаріїв. Це включає створення тестових випадків, перевірку їх виконання та аналіз відповідності результатів очікуваним. Це полегшує виявлення помилок у специфікаціях і моделях UML.

- Візуалізація моделей: USE надає базові можливості для візуалізації UML-моделей, що полегшує розуміння структури та поведінки системи. Використовуючи діаграми класів та об’єктів, розробники можуть легко відстежувати взаємозв’язки між елементами моделі.

- Підтримка різних видів обмежень: Окрім інваріантів і постумов, USE підтримує перевірку таких аспектів, як:

- Обмеження цілісності: визначаються для перевірки того, чи всі об’єкти й зв’язки у моделі відповідають заданим обмеженням.

- Кількісні обмеження: включають обмеження на кількість об’єктів або зв’язків між ними.

- Логічні обмеження: можуть бути використані для визначення більш складних умов поведінки.

- Перевірка моделей на прикладах: Інструмент дозволяє створювати та перевіряти конкретні моделі, де можна наочно побачити, як правила та обмеження працюють в реальних прикладах. Це особливо корисно для моделювання поведінки системи в різних станах.

- Підтримка командного рядка: USE також має інтерфейс командного рядка, що дозволяє автоматизувати деякі процеси перевірки специфікацій і інтегрувати його у більші конвеєри розробки та тестування.

Розглянемо приклад використання. У процесі моделювання можна визначити UML-клас, наприклад, "Клієнт" з атрибутами та операціями, і додати до нього OCL-обмеження. Наприклад, можна додати інваріант, що кожен клієнт повинен мати унікальний ідентифікатор.

Приклад OCL-виразу:

```
context Client
inv UniqueID: Client.allInstances()->forAll(c1, c2 | c1 <>
c2 implies c1.id <> c2.id)
```

Цей вираз визначає, що всі об'єкти класу "Клієнт" повинні мати унікальні ідентифікатори. Інструмент USE допоможе перевірити, чи не порушується це обмеження під час створення моделей та сценаріїв тестування.

Середовище специфікацій на основі UML (USE) може генерувати моделі шляхом автоматичного вирішення обмежень OCL. USE базується на підмножині уніфікованої мови моделювання (UML). Основні особливості використання є:

• Користувач може створювати моделі UML у графічному інтерфейсі за допомогою перетягування, за допомогою команд або сценарію ASSL.

• USE може оцінювати вирази OCL для моделі та повертати значення виразів OCL.

• USE має плагін під назвою «валідатор моделі». Незважаючи на те, що назва цього плагіна — «перевірник моделі», його основна функція — генерувати моделі екземплярів відповідно до обмежень OCL, указаних користувачем.

Завдяки цим функціям USE може генерувати та розділяти моделі за допомогою класифікуючих термінів. Концепція класифікації термінів представлена в наступному розділі.

3.1.1. Основні терміни класифікації

Терміни класифікації — це вирази OCL, які описують особливості моделей. Терміни класифікації зі значеннями є обмеженнями OCL. Ці значення утворюють характерне значення. Значення характеристик використовуються для ідентифікації розділів. Інваріанти OCL і класифікуючі терміни з характерними значеннями є обмеженнями OCL.

Кількість партицій становить 2^N, де N – загальна кількість класифікуючих термінів. У наступному тестовому випадку N дорівнює 3. Кількість розділів дорівнює 8. Дослідження [6, 7, 25] запропонували ідею класифікації термінів на основі USE. Терміни класифікації використовуються для побудови характеристичного значення, яке ідентифікує розділ еквівалентності. Для кожного розділу USE може створити модель екземпляра.

3.1.2. Приклад застосування інструменту USE

Оскільки ми зацікавлені в валідаторі моделі USE, ми провели простий приклад, щоб перевірити, чи може валідатор моделі створити набір екземплярів моделей із класифікуючими термінами. Враховуючи метамоделі Ecore, ми використали інструмент для генерації моделей екземплярів, щоб кожен розділ мав одну модель. Ми використали попередньо визначену метамодель джерела з випадку UML2Relational (див. рис. 2.2).

Етапи реалізації:

- Переведіть метамоделі Ecore у формат специфікацій USE. Повний файл специфікації USE включає інформацію про метамоделі, інваріанти OCL, спільні для всіх моделей екземплярів, та класифікуючі терміни з характеристичними значеннями.

- Визначте інваріанти OCL метамоделі у файлі специфікації USE. Ці інваріанти OCL є спільними для всіх згенерованих моделей і не є класифікуючими термінами.

- Налаштуйте властивості моделі у графічному інтерфейсі Model Validator.

- Запустіть процес генерації та отримайте модель екземпляра у вигляді діаграми об'єктів.

Інструмент USE приймає виключно специфікації USE як вхідні дані. Тому вихідна метамодель та обмеження OCL були переведені у формат специфікації USE в єдиному файлі. Інструмент USE не має вбудованої

функції для такого перекладу, тому ця операція була виконана за допомогою інструменту TractsTool.

Термін класифікації. У цьому випадку ми визначаємо три логічні терміни класифікації: Multivalued, PrimAttr і ClassAttr. Комбінація булевих значень створює характеристичне значення для розділу еквівалентності. Якщо MultiValued має значення true, це означає, що існує принаймні один багатозначний атрибут. Якщо PrimAttr має значення true, це означає, що існує принаймні один атрибут, тип якого є примітивним типом. Якщо ClassAttr має значення true, це означає, що існує принаймні один атрибут, тип якого є типом класу. Ми визначили інформаційну панель термінів класифікації в специфікації USE.

Лістинг 3.1. Дашборд термінів класифікації

```
1  class CTDashboard
2  attributes
3    MultiValued: Boolean
4    PrimAttr: Boolean
5    ClassAttr: Boolean
6
7  operations
8    MultiValued_OP(): Boolean = Attribute.allInstances->exists(a | a.multivalued =
         true)
9    PrimAttr_OP(): Boolean = Attribute.allInstances->exists(a | a.classifier.
         oclIsTypeOf( Type))
10   ClassAttr_OP(): Boolean =  Attribute.allInstances->exists(a | a.classifier.
         oclIsTypeOf( Classes))
11 end
12
13 context CTDashboard inv CT_Operation :
14   MultiValued = MultiValued_OP() and PrimAttr = PrimAttr_OP() and ClassAttr =
         ClassAttr_OP()
```

Цей дашборд (інформаційна панель) є просто індикатором характерного значення та вказуватиме, до якого еквівалентного розділу належить модель. Рисунок 3.1 показує приклад того, як інформаційна панель

представлена в GUI. У цьому прикладі характерним значенням є 110 (двійкове).

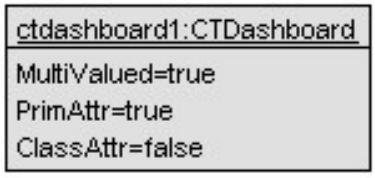

Рис. 3.1. Дашборд термінів класифікації, показаний в USE GUI

3.1.3. Використання інваріантів у специфікації

Інваріанти OCL метамоделі також повинні бути включені до специфікацій USE. Ми вказали деякі основні обмеження. Пояснення наведені нижче, деякі з інваріантів є частиною визначення мови (наприклад, назви типів є унікальними), тоді як деякі призначені лише для експерименту (наприклад, має бути принаймні 2 типи).

• Згенерований екземпляр повинен мати рівно одну модель.

Лістинг 3.2. Обмеження для Model

```
1 context Model inv Model_Size:
2   Model.allInstances->size() = 1
```

• Має бути принаймні 2 типи.

• Не повинно бути типів з однаковою назвою.

Лістинг 3.3. Обмеження для Type

```
1 context Type
2   inv Type_Size:
3     Type.allInstances()->size() >= 2
4   inv Type_Name:
5     Type.allInstances()->exists( a, b | (a.name = b.name) and a <> b) = false
```

• Повинен бути принаймні один клас.

• Не повинно бути класів з однаковою назвою.

• Не повинно бути жодного типу з тим самим іменем, що й будь-який клас.

• Не повинно бути атрибутів з такими ж іменами, як будь-які класи.

Лістинг 3.4. Обмеження для Classes

```
context Classes
inv Classes_Size:
Classes.allInstances()->size() >= 1
inv Unique_ClassesName_AmongClasses:
Classes.allInstances()->exists( a, b| (a.name = b.name) and a <> b) = false
inv Unique_ClassesName_AmongType:
Classes.allInstances()->exists( a| Type.allInstances()->exists(b|a.name = b.
    name)) = false
inv Unique_ClassesName_AmongAttribute:
Classes.allInstances()->exists( a| Attribute.allInstances()->exists(b|a.name =
     b.name)) = false
```

• Кількість атрибутів має бути не меншою за кількість класів.

• Не повинно бути атрибутів з однаковими іменами.

Лістинг 3.5. Обмеження для Attribute

```
context Attribute
    inv Attribute_Size:
      Attribute.allInstances->size() >= Classes.allInstances->size()
    inv Unique_AttrName_AmongAttr:
        Attribute.allInstances()->exists( a, b| (a.name = b.name) and a <> b)
            = false
```

3.1.4. Налаштування конфігурації в UML-based Specification Environment

USE може генерувати моделі екземплярів на основі заданих обмежень і кордонів. USE надає користувачеві графічний інтерфейс для налаштування генерації всіх елементів. Якщо очікуваним значенням Classes є «Main», як показано на рис. 3.2, тоді USE згенерує класи з назвою «Main». У

конфігурації можна вказати кілька очікуваних значень. Якщо поле залишити порожнім, USE генеруватиме випадкові рядки.

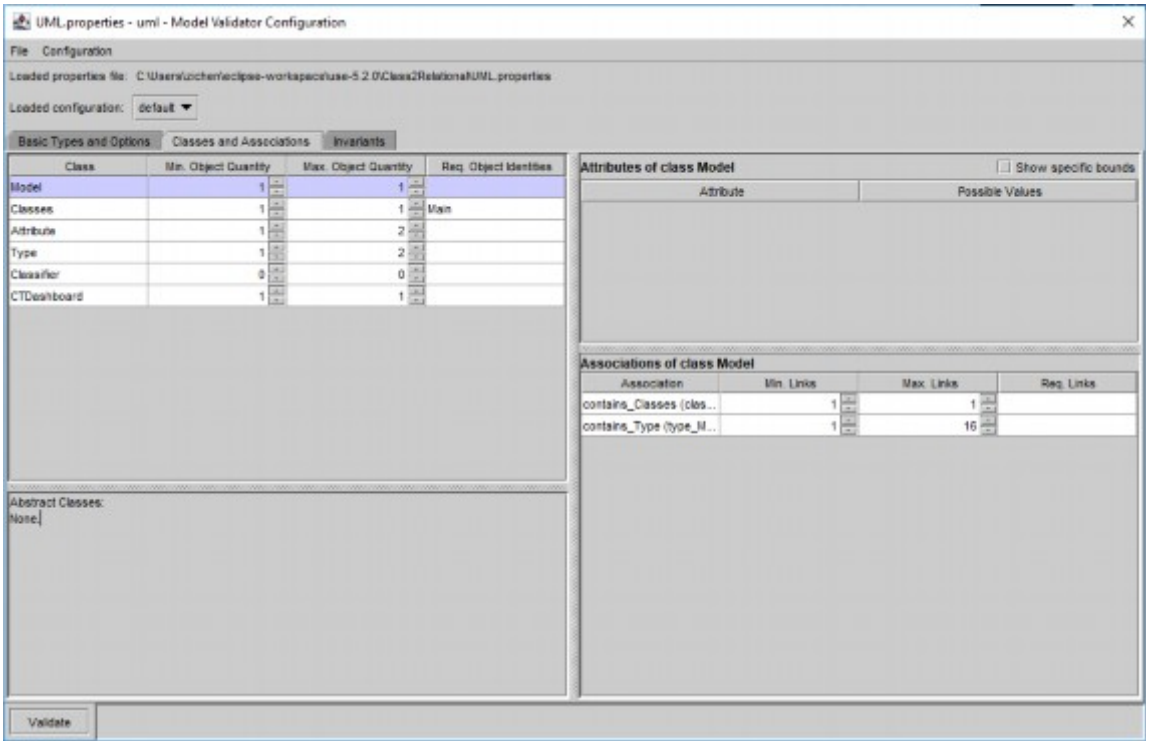

Рис. 3.2. Конфігурація

Отже, USE згенерував 8 рішень, показаних на рис. 3.3. Кожне рішення належить до розділу. Характеристики кожного розділу наведені в табл. 3.1. Цей випадок показує, що USE може генерувати моделі екземплярів відповідно до обмежень OCL. Однак цей інструмент може експортувати створені моделі лише як PDF-файли. Тому ми не проводили практичний кейс з USE.

Таблиця 3.1.

Характеристики розділів

MultiValued	PrimAttr	ClassAttr
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

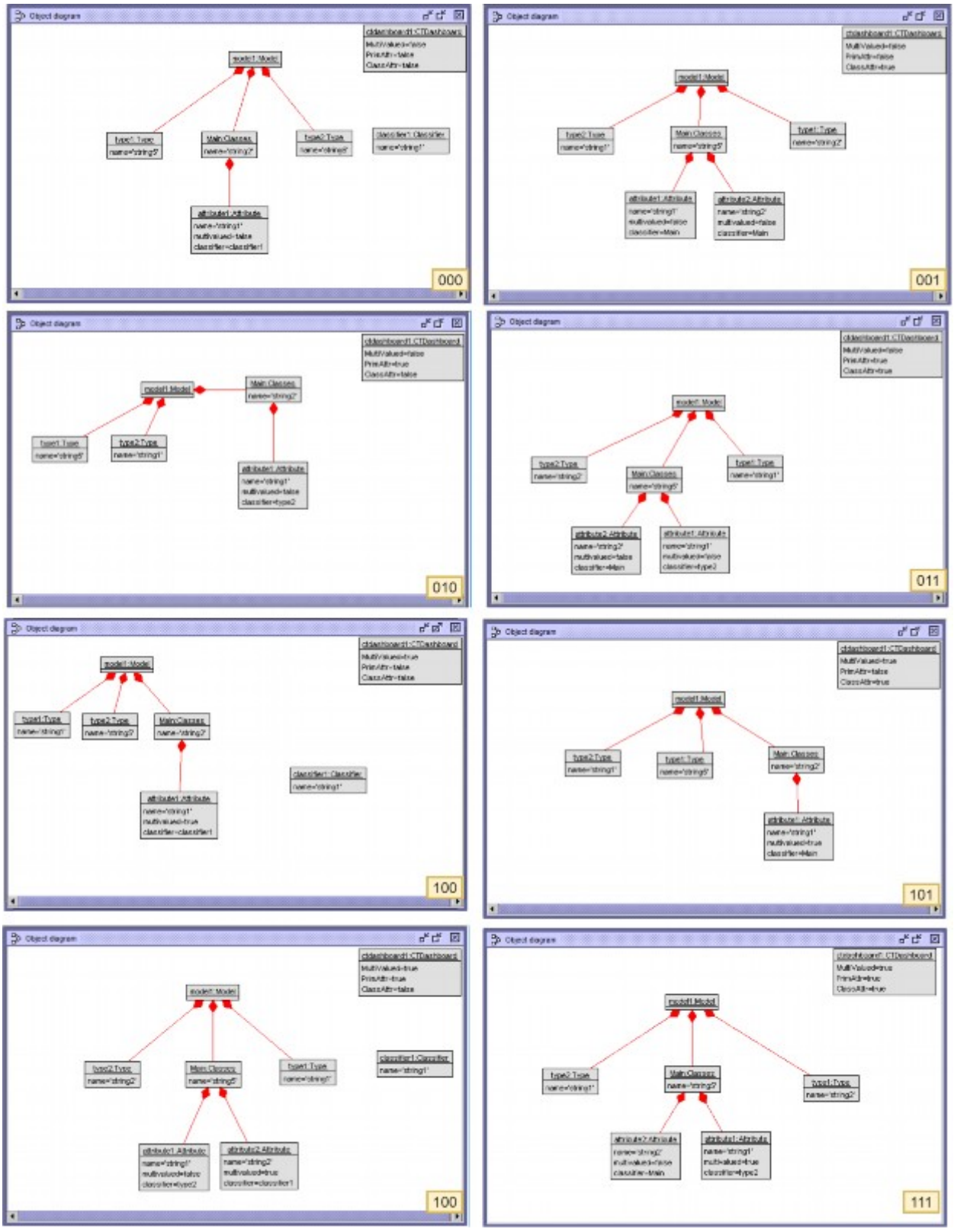

Рис. 3.3. Згенеровані моделі

3.1.5. Результати роботи середовища специфікацій

Отже, USE може генерувати моделі екземплярів відповідно до обмежень OCL. Найважливішою особливістю є те, що USE може надавати

рішення для набору обмежень OCL і генерувати екземпляри під час виконання. Ця функція корисна, коли тестувальникам потрібно створити тестові моделі з комбінаціями різних функцій.

Обмеження USE. Метамодель спочатку потрібно перевести в специфікації USE, але USE не має цієї функції. Крім того, USE може зберігати моделі екземплярів лише як PDF-файл, тому немає вбудованого механізму експорту до EMF.

Таблиця 3.2 показує узагальнені результати оцінювання USE. Підсумовуючи, USE має потенціал для покращення поточного процесу тестування. Ми не зазнали серйозних помилок. Але через дві проблеми, згадані вище, і той факт, що інструмент під назвою Efinder [12] був нещодавно розроблений, ми вирішили не проводити подальше прикладне дослідження з USE. Натомість ми вирішили дослідити придатність Efinder, оскільки він пом'якшив обмеження USE, пропонуючи кращу інтеграцію з EMF.

Таблиця 3.2.

Результати оцінювання USE

No.	Criteria	Results (Yes or No)	Reasons
1	The tool used in the proposed approach(s) can run without errors in the small case studies.	Yes	According to the small case study, USE can perform without errors.
2	The tool can handle industrial-size model transformations	No	USE does not accept Ecore metamodels and can not export models as Ecore models.
3	The total time required by the proposed approaches should be less than the time required by the current practice of model transformation testing at Altran.	NA	We can not make a judgment on this criterion because we can not apply it to the industrial case study.
4	The proposed approach has the potential to improve the current practice of model transformation testing.	Yes	The classifying term approach using USE model validator is to reduce the workload of manual modeling so that it can be easier for testers to generate a qualified test suite. The idea of classifying terms can help avoid structurally similar test models.

3.2. Використання інструменту Efinder для побудови UML моделей

3.2.1. Особливості та призначення інструменту побудови моделей

USE — це інструмент, який можна використовувати для вирішення обмежень і створення моделей UML. Як показано в розділі 3.1 , USE не може експортувати моделі як файли XMI і не приймає метамоделі Ecore. Efinder [12] вирішив ці дві проблеми USE.

Efinder — це інструмент для генерації прикладів моделей, який використовується для тестування трансформацій моделей, перевірки правил та обмежень у специфікаціях моделей, а також для валідації моделей на основі визначених умов. Основне призначення Efinder полягає у генеруванні коректних моделей на основі специфікацій, які задані мовою OCL (Object Constraint Language). Цей інструмент допомагає автоматизувати процес створення прикладів моделей, які відповідають заданим правилам і обмеженням.

Наведемо основні характеристики інструменту Efinder:

- Генерація моделей на основі OCL: Efinder дозволяє створювати приклади моделей, що відповідають обмеженням, визначеним у UML-специфікаціях за допомогою мови OCL. Це дає можливість автоматично отримувати набори моделей, які підходять для тестування трансформацій, перевірки коректності специфікацій та інших випадків використання.

- Перевірка специфікацій: Інструмент використовує OCL-вирази для перевірки того, чи відповідають згенеровані моделі всім встановленим інваріантам, умовам та обмеженням. Це дозволяє виявити помилки у специфікаціях або обмеженнях, на основі яких були побудовані UML-моделі.

- Пошук помилок у моделюванні: Efinder може бути використаний для знаходження помилкових моделей, тобто таких, які не відповідають специфікаціям. Інструмент генерує можливі моделі і вказує на ті, що порушують правила, що є корисним для аналізу та усунення недоліків у моделі.

- Валідація трансформацій моделей: Efinder також використовується для тестування трансформацій моделей, перевіряючи, чи можуть вони коректно працювати на різних прикладах моделей. Генерація прикладів дозволяє побачити, як трансформації працюють у різних ситуаціях та з різними конфігураціями даних.

- Автоматизація тестування: Інструмент підтримує автоматичне генерування великої кількості моделей для масового тестування трансформацій та перевірки обмежень. Це суттєво підвищує ефективність тестування і знижує кількість ручної роботи, пов'язаної з створенням тестових прикладів.

- Інтеграція з іншими інструментами: Efinder можна інтегрувати з іншими інструментами для моделювання і тестування, що дозволяє використовувати його як частину комплексного середовища для моделі-орієнтованої розробки. Наприклад, його можна поєднувати з інструментами для UML-моделювання або тестування трансформацій.

- Різноманітність сценаріїв використання: Генерація моделей може бути корисною для різних цілей:

- Тестування трансформацій: перевірка коректності перетворень моделей.

- Валідація правил: перевірка, чи відповідають моделі встановленим правилам і обмеженням.

- Аналіз специфікацій: тестування специфікацій для виявлення можливих помилок у проєкті системи.

Переваги використання Efinder:

- Швидка генерація моделей: автоматизація процесу створення моделей значно скорочує час, необхідний для підготовки тестових прикладів.

- Масштабованість: можливість генерування великої кількості моделей для масового тестування трансформацій та правил.

- Гнучкість: підтримка широкого спектра UML та OCL специфікацій дозволяє застосовувати інструмент до різних проєктів і завдань.

Efinder — це інструмент, призначений для автоматизації процесу генерування прикладів моделей на основі специфікацій UML та OCL. Його використання сприяє ефективній перевірці трансформацій моделей, тестуванню правил та обмежень, а також аналізу коректності моделей. Efinder підвищує ефективність тестування і забезпечує надійну валідацію програмних систем на різних етапах розробки.

Efinder — це інструмент на основі USE. Він має дві нові функції. Перша полягає в тому, що він може трансформувати метамоделі Ecore та обмеження OCL у специфікації USE. Фрагмент перекладених специфікацій USE наведено в лістингу 3.6.

Лістинг 3.6. Фрагмент перекладених специфікацій засобами Efinder

```
model uml

class Model
operations
  oclContainer() : OclAny = oclUndefined(OclVoid)
  oclContents() : Set(OclAny) = self.classifiers->asSet()
end

class Class < Classifier
operations
  oclContainer() : OclAny = oclUndefined(OclVoid)
  oclContents() : Set(OclAny) = self.attribute->asSet()
end

class Attribute
attributes
  name : String
  multivalued : Boolean
operations
  oclContainer() : OclAny = Class.allInstances()->select(o|o.attribute->includes(
      self))->any(true)
  oclContents() : Set(OclAny) = Set { }
end

class Type < Classifier
operations
  oclContainer() : OclAny = oclUndefined(OclVoid)
  oclContents() : Set(OclAny) = Set { }
end
```

По-друге, Efinder може зберігати моделі екземплярів як дані XMI (XML Metadata Interchange).

3.2.2. Практичний приклад: генерація моделей кількох екземплярів

Метою цього прикладу є створення набору моделей за допомогою класифікуючих термінів. Ми використали метамодель UML із випадку UML2Relational, який ми визначили раніше (показано на рисунку 2.2). Кожна модель має належати до розділу, ідентифікованого значенням характеристики.

Загалом реалізація цього прикладу складається з двох етапів:

• Генерування набору файлів OCL за допомогою сценарію.

• Виконання Efinder з кожним файлом OCL і генерування моделі екземпляра для розділу, якщо це дозволено.

Для кожного виконання Efinder приймає файл OCL як вхідні дані. Потім він автоматично трансформує відповідні метамоделі Ecore у специфікації USE та створить рівно одну модель. Тому, щоб створити кілька моделей, нам потрібно підготувати кілька файлів OCL. Кожен файл OCL відповідає розділу.

Кількість розділів дорівнює кількості файлів OCL. Кількість файлів OCL дорівнює кількості розділів. У цьому тестовому випадку кількість файлів OCL, які потрібно підготувати, становить 8. Файли OCL мають однакові інваріанти OCL. Різні OCL-файли мають однакові терміни класифікації, але з різними характерними значеннями.

Щоб автоматизувати цю генерацію, був розроблений сценарій, що поданий у лістингу 3.7 для генерації обмежень OCL з використанням термінів класифікації логічного типу. Існують різні типи термінів класифікації. Якщо термін класифікації не має логічного значення, ми здійснюємо формулювання його таким чином, щоб він закінчувався логічними значеннями.

Лістинг 3.7. Код для створення OCL-файлів із класифікуючими термінами

```
1  public class ClassifyingTerms {
2    int size = 0;//num of classifying terms (CT)
3    int value = 0;// the maximum characteristic value of the classifying terms, e.g.,
         for 3 CTs, the maximum characteristic value = 111, which is 7 in decimal
4    HashMap<String, List<String>> mapOfCTs = new LinkedHashMap<String, List<String
         >>();
5
6    public ClassifyingTerms(LinkedHashMap<String, String> mapOfPaths) throws
         IOException {
7      for(Map.Entry<String, String> m : mapOfPaths.entrySet()) {
8
9        String context = m.getKey();
10       String path = m.getValue();
11       File fileCT = new File(path);
12       List<String> CTs = new LinkedList<String>();
13       BufferedReader readerCT = new BufferedReader(new FileReader(fileCT));
14       String ct = null;
15       while ((ct = readerCT.readLine()) != null) {
16         CTs.add(ct);
17         System.out.println(ct);
18       }
19       this.size += CTs.size();
20       this.mapOfCTs.put(context, CTs);
21       readerCT.close();
22     }
23     //calculate the maximum characteristic value
24     int pow = this.size - 1;
25     while(pow != -1) {
26       this.value += Math.pow(2, pow--);
27     }
28   }
29 }
```

Наприклад, в лістингу 3.8 показано термін класифікації цілого типу. Якщо нам потрібно згенерувати розділи для моделей, розміри їхніх об’єктів – від розміру об’єкта 1 до розміру об’єкта 1–3.

Тим часом існують інші типи термінів класифікації. Щоб просто створити OCL-файли, нам потрібно написати терміни класифікації цілого типу, такі як терміни класифікації булевого типу.

Лістинг 3.8. Класифікаційний термін цілого типу

```
1 Context Entity
2  inv Entity_Size:
3     Entity.allInstances()->size() = 1
```

У лістингу 3.9 показано два терміни класифікації логічного типу, Entity Size1 з логічним значенням true і Entity Size2 з логічним значенням false. Значення характеристики (двійкове) дорівнює 01. Сценарій у лістингу 3.7 призначить цим двом термінам класифікації характерні значення від 00 до 11. Якщо значення характеристики дорівнює 11, генерація моделі не вдасться, оскільки кількість об'єктів не може бути істинною одночасно. У цьому випадку Ender пропустить генерацію. Будуть створені лише задовільні моделі.

Лістинг 3.9. Термін класифікації цілого типу переписується на два терміни класифікації булевого типу

```
1 Context Entity
2  inv Entity_Size1:
3     Entity.allInstances()->size() = 1 = true
4  inv Entity_Size2:
5     Entity.allInstances()->size() = 2 = false
```

Усі терміни класифікації отримають логічне значення під час генерації файлу OCL. Усі терміни класифікації в одному контексті потрібно вказувати в одному текстовому файлі рядок за рядком. Наприклад, в лістингу 3.10 показано зміст класифікаційних термінів. Ці терміни класифікації отримають логічне значення та будуть вставлені у файл OCL за допомогою сценарію.

Лістинг 3.10. Три терміни класифікації

```
1 Attribute.allInstances()->exists(a | a.multivalued = true)
2 Attribute.allInstances()->exists(a | a.classifier.oclIsTypeOf( Type))
3 Attribute.allInstances()->exists(a | a.classifier.oclIsTypeOf( Class))
```

Лістинг 3.11 показує приклад повного файлу OCL. Він визначає шлях до файлу метамоделі Ecore, інваріантів і термінів класифікації з характерним значенням. Характеристичне значення 000 (двійковий) у файлі. ”- -” є символом коментаря в OCL.

Лістинг 3.11. Повний файл OCL із значенням характеристики 000

```
import 'uml.ecore'
package uml
context Model inv Model_Size:
  Model.allInstances()->size() = 1

context Type
  inv Type_Size:
    Type.allInstances()->size() >= 2
  inv Type_Name:
    Type.allInstances()->exists( a, b | (a.name = b.name) and a <> b) = false --
        there should be no types with the same name

context Class
  inv Class_Size:
    Class.allInstances()->size() >= 1
  inv Unique_ClassName_AmongClass:
    Class.allInstances()->exists( a, b| (a.name = b.name) and a <> b) = false --
        there should be no Class with the same name.
  inv Unique_ClassName_AmongType:
    Class.allInstances()->exists( a| Type.allInstances()->exists(b|a.name = b.name)
        ) = false -- there should be no Type with the same name as any Class.
  inv Unique_ClassName_AmongAttribute:
    Class.allInstances()->exists( a| Attribute.allInstances()->exists(b|a.name = b.
        name)) = false -- there should be no Attribute with the same name as any
        Class.

context Attribute
  inv CT1:
  Attribute.allInstances()->exists(a | a.multivalued = true) = false
  inv CT2:
  Attribute.allInstances()->exists(a | a.classifier.oclIsTypeOf( Type)) = false
  inv CT3:
  Attribute.allInstances()->exists(a | a.classifier.oclIsTypeOf( Class)) = false
  inv Attribute_Size:
    Attribute.allInstances()->size() >= Class.allInstances()->size()
  inv Unique_AttrName_AmongAttr:
      Attribute.allInstances()->exists( a, b| (a.name = b.name) and a <> b) = false
          -- there should be no Class with the same name.

endpackage
```

3.2.3. Налаштування і виконання процесу генерування моделей

Основний тестовий код показано в лістингу 3.12, а конфігурації моделі розташовані з рядка 26 по рядок 30. Користувачеві необхідно налаштувати очікувані мінімальні значення та максимальні значення елементів.

Лістинг 3.12. Тестовий код

```
1
2  public class ClassTest extends AbstractEmfOclTest{
3
4      public void registerMetamodel(String pathOfEcoreModel) {
5      Resource.Factory.Registry.INSTANCE.getExtensionToFactoryMap().put(
6            "ecore", new EcoreResourceFactoryImpl());
7      ResourceSet rs = new ResourceSetImpl();
8      // enable extended metadata
9      final ExtendedMetaData extendedMetaData = new BasicExtendedMetaData(rs.
           getPackageRegistry());
10     rs.getLoadOptions().put(XMLResource.OPTION_EXTENDED_META_DATA,
11         extendedMetaData);
12     URI uriOfYourModel = URI.createURI(pathOfEcoreModel);
13     Resource r = rs.getResource(uriOfYourModel, true);
14     EObject eObject = r.getContents().get(0);
15     if (eObject instanceof EPackage) {
16         EPackage p = (EPackage)eObject;
17         EPackage.Registry.INSTANCE.put(p.getNsURI(), p);
18     }
19   }
20
21   public void configureAndRunTest(String pathOfOCLFile, String pathOfOutputXMI)
         throws FileNotFoundException, IOException {
22     CompleteOCLStandaloneSetup.doSetup();
23     Model pivot = loadOclDocumentFromURI(pathOfOCLFile);
24
25     // Assume that in all tests there is a root class called Model
26     TestBoundsProvider boundsProvider = new TestBoundsProvider()
27         .withInterval("Model", 1, 1)
28         .withInterval("Class", 1, 1)
29         .withInterval("Type", 1, 2)
30         .withInterval("Attribute", 1, 2);
31
32     UseMvFinder finder = new UseMvFinder()
33         .withBoundsProvider(boundsProvider);
34
35     EFinderRunner runner = EFinderRunner.
36         withOclModel(pivot).
37         withFinder(finder);
```

Efinder виконувався 8 разів і створював модель для кожного з 8 розділів. На рисунку 3.4 показано 8 згенерованих моделей екземплярів та їхні властивості. Числа першого ряду є характерними значеннями. Вони використовуються для ідентифікації різних розділів. Цей тест показує, що Efinder може генерувати набір моделей екземплярів, кожна з яких належить до розділу. Ми можемо зробити висновок, що ми можемо використовувати Efinder для створення набору моделей за допомогою класифікуючих термінів.

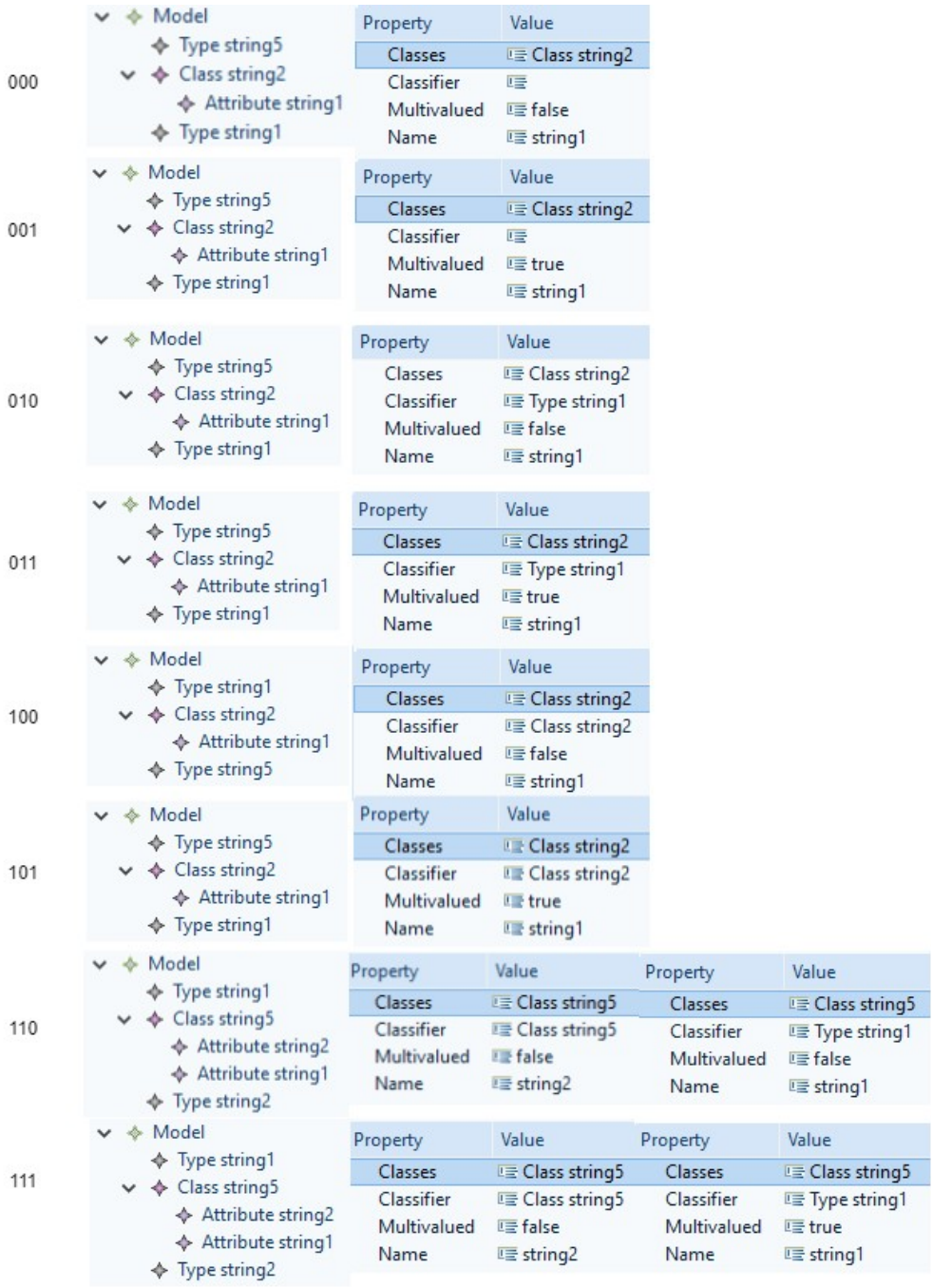

Рис. 3.4. Згенеровані моделі

3.2.4. Опис результатів роботи

Можна константувати факт, що Efinder має потенціал для підвищення загальної продуктивності в тестуванні трансформації моделі. Обидва випадки показали, що Efinder має дві корисні функції порівняно з USE. Він може трансформуватися.

Метамодель Ecocore, інваріанти OCL та класифікація значень термінів у специфікації USE. Efinder також може зберігати згенеровані екземпляри як моделі Ecore. Efinder може генерувати набір моделей екземплярів відповідно до визначених умов класифікації. Він створить модель екземпляра для кожного розділу, визначеного термінами класифікації. Ця функція зменшує зусилля при ручному створенні моделей екземплярів.

Це також дозволяє уникнути повторюваних або структурно схожих тестових моделей, створених тестувальниками під час створення моделей. Під час генерації тестових моделей вручну тестер може створювати моделі, що належать до того самого розділу еквівалентності. Efinder може уникнути проблеми, оскільки він генеруватиме різні моделі, ідентифіковані за значенням характеристики.

Якість тестування залежить від класифікаційних умов, наданих тестувальниками. У поточній версії Efinder користувачеві потрібно не лише написати терміни класифікації, а й вказати інші інваріанти, щоб рішення було задовільним. Якість тестування залежить від умов класифікації, які задають тестувальники.

Щодо недоліків Efinder, то тут треба розглянути декілька проблем. По-перше, є деякі операції OCL, які досі не підтримуються, зокрема "matchs", ”size”, ”at”, ”toUpper”, ”toUpperCase”, ”toLower”, "підрядок" і "oclType". Це обмежує обмеження, які може вказати користувач. Відповідно потрібно ігнорувати оригінальні інваріанти OCL метамоделей, щоб Efinder міг скомпілювати.

По-друге, операція, визначена розробниками проектів не підтримується Efinder. Це допоміжна операція. В OCL усі невдачі інваріантів є помилками.

Розробники представили ці помічники для розрізнення помилок і попереджень. Efinder не може їх скомпілювати, однак ці операції незначні і ними можна знехтувати. Якщо в моделі є помилка, результат виразу OCL інваріанта буде оцінено як нульовий, а потім буде перетворено на нульовий. Це спричинить проблему для Efinder. Тоді Efinder може створити недійсні моделі.

По-третє, Efinder не підтримує взаємодію між обмеженнями OCL і написаними вручну операціями Java у метамоделях. Якщо тестувальники в хочуть отримати користь від Efinder, їм потрібно використовувати OCL для обмежень замість Java.

По-четверте, з прикладу ми виявили, що повідомлення про незадовільне підтвердження не є зрозумілим для користувача.

По-п'яте, Efinder не підтримує створення пари підключених моделей. Очікується, що інструмент може надати цю функцію для користувача, оскільки ланцюжок перетворення приймає дві моделі як вхідні дані. Ми вирішили це, написавши програму для поділу монолітної моделі на одну модель і одну модель генератора.

Хоча плагін термінів класифікації ще не розроблено, ми маємо певні очікування щодо нього. Плагін для класифікації термінів повинен запитувати у користувача інформацію про класифікацію термінів відповідно до типу класифікованого терміна. Він повинен мати зручний інженерний інтерфейс. Він повинен автоматично запустити Efinder, щоб створити модель для кожного розділу відповідно до характерних значень. Наприклад, для цілочисельного типу терміна класифікації, ціле значення є характерним значенням, яке ідентифікує розділ. Користувач може очікувати, що це ціле значення буде варіюватися від 1 до 100. Тоді цей плагін повинен запитати діапазон, який розділ, і автоматично генерувати файли OCL.

Отже, Efinder вирішив дві основні проблеми USE. Efinder робить ідею класифікації термінів більш придатною для поточного тестування трансформації моделі в галузі. Таблиця 3.3 показує результати оцінювання

Efinder. Незважаючи на те, що Efinder все ще має деякі проблеми, згадані вище, Efinder з ідеєю класифікації термінів має потенціал для покращення поточної практики оцінки тестування трансформації моделей імплементації.

Таблиця 3.3.

Результати оцінювання Efinder

No.	Criteria	Results (Yes or No)	Reasons
1	The tool used in the proposed approach(s) can run without errors in the small case studies.	Yes	According to the small case study, Efinder can perform without errors.
2	The tool can handle industrial-size model transformations	No	In the industrial case studies, the compilation of OCL files and Ecore models failed. Some OCL operations and Java operations are not supported.
3	The total time required by the proposed approaches should be less than the time required by the current practice of model transformation testing at Altran.	NA	We can not measure it since the current version of the tool still has problems in the industrial case studies.
4	The proposed approach has the potential to improve the current practice of model transformation testing.	Yes	Efinder extends USE. Therefore, Efinder also can implement classifying terms and bring the benefits of classifying terms to model transformation testing.

Висновки до розділу

Отже, в цьому розділі представлено проведено детальне дослідження та практичне застосування методів і інструментів для оцінки тестування генерації та трансформації моделей. Розглянуто середовище специфікацій на основі UML (USE). Досліджено застосування середовища USE для формальної перевірки моделей UML за допомогою OCL (Object Constraint Language) і показано, як за допомогою інваріантів та інших обмежень можна гарантувати коректність моделей. Розглянуто налаштування конфігурацій середовища для ефективного тестування специфікацій та проведено аналіз результатів роботи середовища. USE дозволило виявити помилки та недоліки в моделях на ранніх етапах їх розробки.

Досліджено інструмент Efinder для побудови UML моделей. Проаналізовано функціональність Efinder як інструменту для генерації моделей UML, що відповідають заданим специфікаціям і обмеженням. Показано, що Efinder значно спрощує процес автоматизованого створення тестових моделей, що дозволяє прискорити валідацію трансформацій та перевірку правил. Детально описано процес налаштування і виконання генерації моделей, а також результати роботи інструменту, які підтвердили його ефективність у тестуванні різних сценаріїв.

Отже, розділ демонструє важливість використання спеціалізованих інструментів, таких як USE і Efinder, для тестування трансформацій і генерації моделей. Ці інструменти забезпечують можливість автоматизованого, систематичного підходу до перевірки моделей і специфікацій, що значно підвищує надійність та якість програмних систем. Завдяки їх використанню, процес моделі-орієнтованої розробки стає більш ефективним та структурованим, зменшуючи ризики виникнення помилок на різних етапах проєктування та реалізації програмного забезпечення.

ВИСНОВКИ

У магістерській роботі проведено всебічний аналіз методів і інструментів для оцінки тестування трансформації моделей імплементації. Для розуміння поточного стану та проблематики тестування трансформацій у промислових умовах було здійснено глибоке дослідження наявних підходів до тестування, а також вивчено потреби тестувальників. У рамках дослідження проведено огляд наукової літератури, що охоплює основні підходи до вирішення питань тестування трансформацій моделей. Огляд дозволив визначити два основні типи підходів: перевірку коректності трансформацій моделей за допомогою формальних специфікацій і автоматичну генерацію моделей з подальшим розділенням еквівалентних класів.

Для реалізації першого підходу були обрані два інструменти: TractsTool та Matching Table Builder, які базуються на формальних специфікаціях і використовують концепцію трактатів. Трактати виступають як контракти трансформації, що описують логіку перетворення між вихідними та цільовими моделями. Другий підхід, пов’язаний із автоматичною генерацією моделей, було досліджено на прикладі інструментів USE та Efinder.

Оцінка ефективності зазначених інструментів здійснювалась за допомогою чотирьох критеріїв, на основі яких було проведено серію малих досліджень на конкретних випадках використання. У ході цього процесу було виявлено низку проблем і обмежень у використанні TractsTool, Matching Table Builder та USE для промислових потреб, що ставить під сумнів їх придатність у великих проєктах. Єдиним інструментом, який показав позитивні результати під час малих досліджень, був Efinder, що дало змогу провести додаткові промислові дослідження з його використанням.

У роботі детально обговорено результати оцінки придатності чотирьох інструментів на основі запропонованих критеріїв, а також проаналізовано

можливі напрями їх удосконалення для подальшого застосування в промислових умовах. Зокрема, обговорюються шляхи покращення функціональних можливостей інструментів TractsTool, Matching Table Builder, USE та Efinder з метою підвищення їх практичної цінності.

Таким чином, основним висновком роботи є те, що хоча Efinder продемонстрував найбільшу ефективність для генерації моделей у промислових умовах, інші інструменти, такі як TractsTool, Matching Table Builder та USE, потребують додаткових досліджень та вдосконалень для їх повноцінного застосування у складних промислових проєктах.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Vincent Aranega, Jean-Marie Mottu, Anne Etien, Thomas Degueule, Benoit Baudry, and Jean-Luc Dekeyser. Towards an automation of the mutation analysis dedicated to model transformation. Software Testing, Verication and Reliability, 25(5-7):653{683, 2015.
2. Ellen Francine Barbosa, Jos e Carlos Maldonado, and Auri Marcelo Rizzo Vincenzi. Toward the determination of sucient mutant operators for c. Software Testing, Verication and Reliability, 11(2):113{136, 2001.
3. Benoit Baudry, Trung Dinh-Trong, Jean-Marie Mottu, Devon Simmonds, Robert France, Sudipto Ghosh, Franck Fleurey, and Yves Le Traon. Model transformation testing challenges. 2006.
4. Benoit Baudry, Sudipto Ghosh, Franck Fleurey, Robert France, Yves Le Traon, and Jean-Marie Mottu. Barriers to systematic model transformation testing. Communications of the ACM, 53(6):139{143, 2010.
5. Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. Metamodelbased test generation for model transformations: an algorithm and a tool. In 2006 17th International Symposium on Software Reliability Engineering, pages 85{94. IEEE, 2006.
6. [6] Loli Burgueno. Testing m2m/m2t/t2m transformations. In SRC@ MoDELS, pages 7{12, 2015.
7. Loli Burgue~no, Frank Hilken, Antonio Vallecillo, and Martin Gogolla. Testing transformation models using classifying terms. In International Conference on Theory and Practice of Model Transformations, pages 69{85. Springer, 2017
8. Loli Burgue~no, Javier Troya, Manuel Wimmer, and Antonio Vallecillo. Static fault localization in model transformations. IEEE Transactions on Software Engineering, 41(5):490{506, 2014.
9. Eric Cariou, Rapha el Marvie, Lionel Seinturier, and Laurence Duchien. Ocl for the specication of model transformation contracts. In OCL and

Model Driven Engineering, UML 2004 Conference Workshop, volume 12, pages 69{83, 2004.

10. Hector M Chavez, Wuwei Shen, Robert B France, and Benjamin A Mechling. An approach to testing java implementation against its uml class model. In International Conference on Model Driven Engineering Languages and Systems, pages 220{236. Springer, 2013.
11. Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. Metamorphic testing: a new approach for generating next test cases. Technical report, Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong . . . , 1998.
12. Jesus Sanchez Cuadrado and Martin Gogolla. Modelnding in the emf ecosystem. Journal of Object Technology, 19(2), 2020.
13. Jesus Sanchez Cuadrado, Esther Guerra, and Juan de Lara. Static analysis of model transformations. IEEE Transactions on Software Engineering, 43(9):868{897, 2016.
14. Jesus Sanchez Cuadrado, Esther Guerra, Juan de Lara, Robert Claris o, and Jordi Cabot. Translating target to source constraints in model-to-model transformations. In 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), pages 12{22. IEEE, 2017.
15. Jean-Marie Favre. Foundations of meta-pyramids: Languages vs. metamodels{episode ii: Story of thotus the baboon1. In Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum f ur Informatik, 2005. 5
16. Olivier Finot, Jean-Marie Mottu, Gerson Suny e, and Christian Attiogb e. Partial test oracle in model transformation testing. In International Conference on Theory and Practice of Model Transformations, pages 189{204. Springer, 2013.
17. Olivier Finot, Jean-Marie Mottu, Gerson Suny e, and Thomas Degueule. Using meta-model coverage to qualify test oracles. 2013.

18. Franck Fleurey, Benoit Baudry, Pierre-Alain Muller, and Yves Le Traon. Towards dependable model transformations: Qualifying input test data. 2007.
19. Franck Fleurey, Benoit Baudry, Pierre-Alain Muller, and Yves Le Traon. Qualifying input test data for model transformations. Software & Systems Modeling, 8(2):185{203, 2009.
20. Piero Fraternali and Massimo Tisi. Multi-level tests for model driven web applications. In International Conference on Web Engineering, pages 158{172. Springer, 2010.
21. Jokin Garca, Maider Azanza, Arantza Irastorza, and Oscar Daz. Testing mofscript transformations with handymof. In International Conference on Theory and Practice of Model Transformations, pages 42{56. Springer, 2014. 18, 19
22. Christine M Gerpheide, Ramon RH Schi elers, and Alexander Serebrenik. Assessing and improving quality of qvto model transformations. Software Quality Journal, 24(3):797{834, 2016.
23. Martin Gogolla, Fabian B uttner, and Mark Richters. Use: A uml-based specication environment for validating uml and ocl. Science of Computer Programming, 69(1-3):27{34, 2007.
24. Martin Gogolla and Antonio Vallecillo. Tractable model transformation testing. In European Conference on Modelling Foundations and Applications, pages 221{235. Springer, 2011.
25. Martin Gogolla, Antonio Vallecillo, Loli Burgueno, and Frank Hilken. Employing classifying terms for testing model transformations. In 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), pages 312{321. IEEE, 2015.
26. Carlos A Gonz alez and Jordi Cabot. Atltest: a white-box test generation approach for atl transformations. In International Conference on Model Driven Engineering Languages and Systems, pages 449{464. Springer, 2012.

27. Carlos A Gonz alez and Jordi Cabot. Test data generation for model transformations combining partition and constraint analysis. In International Conference on Theory and Practice of Model Transformations, pages 25{41. Springer, 2014.
28. Esther Guerra, Jes us S anchez Cuadrado, and Juan de Lara. Towards e ective mutation testing for atl. In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), pages 78{88. IEEE, 2019.
29. Esther Guerra, Juan De Lara, Dimitris Kolovos, and Richard Paige. A visual specication language for model-to-model transformations. In 2010 IEEE symposium on visual languages and human-centric computing, pages 119{126. IEEE, 2010.
30. Esther Guerra, Juan de Lara, Manuel Wimmer, Gerti Kappel, Angelika Kusel, Werner Retschitzegger, Johannes Sch onb ock, and Wieland Schwinger. Automated verication of model transformations based on visual contracts. Automated Software Engineering, 20(1):5{46, 2013
31. Esther Guerra and Mathias Soeken. Specication-driven model transformation testing. Software & Systems Modeling, 14(2):623{644, 2015.
32. Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. Formal analysis of model transformations based on triple graph grammars. Mathematical Structures in Computer Science, 24(4), 2014.
33. Stephan Hildebrandt, Leen Lambers, Holger Giese, Jan Rieke, Joel Greenyer, Wilhelm Schafer, Marius Lauder, Anthony Anjorin, and Andy Sch urr. A survey of triple graph grammar tools. Electronic Communications of the EASST, 57, 2013. 15
34. Bézivin, J., & Gerbé, O. (2001). "Towards a precise definition of the OMG/MDA framework." Automated Software Engineering, 8(2), 165-181.
35. Mens, T., & Van Gorp, P. (2006). "A taxonomy of model transformation." Electronic Notes in Theoretical Computer Science, 152, 125-142.

36. Amrani, M., Cordy, J. R., Dingel, J., & Selim, G. M. (2012). "Model transformation testing: the state of the art." International Journal on Software Tools for Technology Transfer (STTT), 16(4), 377-391.
37. Kuang, J., Lin, J., & Yang, J. (2018). "Model transformation testing: A systematic mapping study." Information and Software Technology, 95, 202-219.
38. Fleurey, F., & Steel, J. (2006). "Model transformation testing: From theory to practice." In International Workshop on Model-Driven Engineering, Verification and Validation, 1-10.
39. Boronat, A., & Gómez, J. (2010). "Test-driven development in model transformation." In Model-Driven Engineering Languages and Systems, 748-763.
40. Selim, G. M., Cordy, J. R., & Dingel, J. (2013). "Model transformation testing: A survey." In Software Engineering Research, Management and Applications, 149-164.
41. Aichernig, B. K., et al. (2013). "Test models and coverage criteria for model transformations." In Formal Methods in Software Engineering, 1-12.
42. Kolovos, D. S., et al. (2010). "A framework for model transformation testing." In International Conference on Model-Driven Engineering Languages and Systems, 219-223.
43. Arendt, T., Taentzer, G., & Hermann, F. (2015). "Ensuring well-formedness of in-place model transformations using graph transformation." In Software & Systems Modeling, 14(3), 1203-1225.
44. Mottu, J., Baudry, B., & Le Traon, Y. (2006). "Mutation analysis testing for model transformations." In International Conference on Model Driven Engineering Languages and Systems, 376-390.
45. Lin, J., & Yang, J. (2015). "A formal framework for verifying model transformations in software development." Journal of Systems and Software, 110, 120-134.

46. Kessentini, M., et al. (2013). "Search-based model transformation by example." Software & Systems Modeling, 12(3), 543-560.
47. Boronat, A. (2016). "Metamodel-driven testing of model transformations." In Model-Driven Engineering and Software Development, 227-243.
48. Oakes, M., & Thomo, A. (2019). "Assessing quality in model transformation testing using metrics." Software Quality Journal, 27(1), 145-164.
49. Selim, G. M., Cordy, J. R., & Dingel, J. (2014). "Towards model transformation testing coverage criteria." In 2nd Workshop on the Analysis of Model Transformations, 1-10.
50. Gogolla, M., et al. (2018). "Validation and verification of model transformations." In Model-Driven Engineering Languages and Systems, 110-121.
51. Wimmer, M., et al. (2010). "Model transformation reuse across metamodels." In Theory and Practice of Model Transformations, 31-46.
52. Büttner, F., et al. (2012). "On defining and validating transformations." Journal of Object Technology, 11(1), 2-25.
53. Barbier, F., et al. (2011). "A tool-supported approach to model transformation testing." In 10th Workshop on Model-Based Development, 95-106.
54. Guerra, E., et al. (2013). "A formal approach to ensuring model transformation correctness." In International Conference on Fundamental Approaches to Software Engineering, 116-131.
55. Mottu, J., Baudry, B., & Le Traon, Y. (2008). "Model transformation testing using data perturbation." In Proceedings of the ACM SIGSOFT Symposium on Software Testing and Analysis, 219-229.