МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

магістр

на тему “Використання принципів аналізу формальних понять для валідації цілісності доменної моделі”

Виконав: студент 2 курсу, групи МФ-61

спеціальність 122 «Комп’ютерні науки»

освітньо-наукова програма

«Інформатика»

Голубничий В.О.

(прізвище та ініціали)

Керівник Волков І.В.

(прізвище та ініціали)

Рецензент Стрілець В.Є.

(прізвище та ініціали)

Харків – 2023 року

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

MERODE – Model Driven, Existence Dependency Relation, Object Oriented Development.

CRUD – Create Read Update Delete.

SOLID – Single responsibility, Open–closed principle, Liskov substitution principle, Interface segregation, Dependency inversion.

OOSSADM – Object–Oriented Structured Systems Analysis and Design Methodology.

UML – Unified Modeling Language.

Валідація – процес перевірки того, чи відповідає програмне забезпечення специфікаціям та вимогам користувача.

Верифікація – процес перевірки того, чи відповідає програмне забезпечення певним вимогам, специфікаціям та стандартам, які визначені для цього типу програмного забезпечення.

ВСТУП

В розробці програмного забезпечення існує проблема валідації та верифікації [1], яка полягає в тому, що навіть найбільш талановиті програмісти можуть допустити помилки при написанні програмного коду. Ці помилки можуть бути критичними і призвести до серйозних проблем, таких як виток конфіденційної інформації, втрата даних або навіть аварії системи. Верифікація програмного забезпечення – це процес перевірки того, чи відповідає програмне забезпечення певним вимогам, специфікаціям та стандартам, які визначені для цього типу програмного забезпечення. Основною метою верифікації є перевірка того, що програмне забезпечення виконує свої функції та задовольняє вимогам, які були поставлені перед ним, а також виявлення можливих помилок у процесі розробки та впровадження. В той же валідація є перевіркою того, що програмне забезпечення виконує очікувані функції та задовольняє потребам користувачів. У реальному світі методи верифікації використовуються переважно в критичних галузях, таких як авіація, медицина, або фінанси, де навіть найменша помилка може мати серйозні наслідки, оскільки верифікація є затратним процесом як з точки зору часу, так і з точки зору фінансів. Таким чином, дана дипломна робота фокусується саме на валідації як на більш доступній та менш складній альтернативі. Таким чином, після огляду цих двох важливих понять у світі програмної інженерії слід розглянути процес створення доменних моделей та етапи, що його складають.

Одним із перших кроків у процесі розробки програмного забезпечення є створення концептуальної доменної моделі, на основі якої в подальшому буде базуватися програма [2].

Під час проектування моделі можливо допустити помилки [3], які можуть негативно вплинути як на саму розробку, так і на процес використання програми користувачем, оскільки високоякісна доменна модель

є однією із ключових вимог успішності проекту [4]. До таких помилок слід віднести наступне:

- пропущені об’єкти в доменній моделі, що може привести до повторного використання коду, роблячи його заплутаним та складним для розуміння;
- наявність об’єктів в доменній моделі з властивостями, які не є їм властивими, що може вплинути як на користувача, так і на безпеку застосунку, якщо елементами даної доменної моделі є типи користувачів в системі авторизації програми;
- заплутаність доменної моделі після певного періоду неконтрольованого росту, що також робить модель складною для використання;
- наявність об’єктів в моделі, які більше не використовуються, що робить модель більшою ніж вона є насправді.

Складність вирішення даних помилок також збільшується фактом того, що у випадку доменної моделі із великою кількістю об’єктів та властивостей, перевірка моделі без використання деяких механізмів або правил може не виявити проблеми. На допомогу в таких випадках приходять техніки, що базуються на різноманітних математичних принципах.

Наприклад, останнім часом є наявним певний інтерес до використання формального аналізу концептів для вирішення проблем, які виникають під час розробки доменної моделі та застосунку в цілому:

- реорганізація існуючої ієрархії класів та покращення якості наявного коду [5; 6];
- ідентифікації класів кандидатів у спадковому коді [7];
- ідентифікації класів для написання діаграм використання [8; 9];
- узгодження документації написаної різними розробниками, використовуючи контрольований словник та набір граматики [10; 11];
- дизайну індивідуальних класів та методів [12].

Однак слід зазначити, що формальний аналіз концептів майже не використовувався [13] на початкових етапах моделювання доменної моделі та розробки програмного забезпечення. Тому з'являється необхідність у проведенні дослідження, яке могло би показати можливість використання математичного підходу на ранніх етапах розробки, або спростувати її.

Таким чином, **об'єктом** даного дослідження є використання математичних принципів для валідації цілісності доменної моделі під час розробки програмного забезпечення. Оскільки, як було зазначено вище, існує певна кількість робот, що фокусуються саме на використання формального аналізу концептів для роботи із програмним забезпеченням, то ця робота концентрується на поглибленні існуючих знань, а тому **предмет** роботи можливо сформулювати як ефективність та доцільність використання формального аналізу концептів для аналізу цілісності доменної моделі програмного забезпечення. Наразі існує значна кількість підводних каменів та можливих проблем під час роботи з доменними моделями, тому **метою** даної роботи є спрощення процесу перевірки цілісності даних моделей, що повинно позитивно вплинути як на саму швидкість роботи, так і на задоволення від користування програмним забезпеченням у користувачів.

Результати цього дослідження можна використати як основу для подальших теоретичних та практичних досліджень у даній сфері, а також як підґрунтя для розробки окремої комерційної системи аналізу доменних моделей та валідації їх цілісності. Також варто зазначити, що після використання ідей відображених в даній роботі, допомогло знайти декілька проблем в складній доменній моделі існуючого проекту. Для досягнення вказаної мети було використано як емпіричні (експериментування з наявною доменною моделлю), так і теоретичні (аналіз існуючої проблеми та наявних можливих рішень).

РОЗДІЛ 1
ОГЛЯД ПРИНЦИПІВ ФОМАЛЬНОГО АНАЛІЗУ КОНЦЕПТІВ ТА СУМІЖНИХ ТЕХНОЛОГІЙ

1.1 Етапи розробки доменної моделі

Доменна модель – це концептуальна модель, яка описує ключові об'єкти, концепції та зв'язки в певній діловій галузі або домені. Вона слугує основою для розробки програмного забезпечення, допомагаючи розробникам і бізнес–аналітикам зрозуміти проблеми, що потребують вирішення і способи їх вирішення. Доменна модель містить ключові сутності (наприклад, людей, місця, продукти або послуги), атрибути цих сутностей (наприклад, імена, адреси, розміри або вартості) та взаємодії між ними (наприклад, замовлення, оплати, доставки тощо). Створення доменної моделі складається з багатьох етапів [14]:

- ознайомлення з бізнес–доменом: бізнес аналітик та розробники, які йому допомагають, повинні мати глибоке розуміння бізнес–домену, з яким пов'язана програма. Вони повинні вивчити документацію, спілкуватися з експертами, зрозуміти ключові терміни, які використовуються в галузі;
- визначення ключових понять: команда повинна ідентифікувати ключові об'єкти, з якими пов'язана домена. Наприклад, якщо це інтернет–магазин, ключові об'єкти можуть включати товари, замовлення, клієнти тощо;
- створення сутностей та атрибутів: команда повинна створити список сутностей (наприклад, продукти, замовлення) та атрибутів, які пов'язані з кожною сутністю (наприклад, для продуктів – назва, опис, ціна, вага тощо);

- визначення зв'язків: команда повинна визначити, які зв'язки існують між сутностями. Наприклад, зв'язок "один до багатьох" може вказувати, що один клієнт може мати багато замовлень, а зв'язок "багато до багатьох" може вказувати на те, що кожен продукт може мати багато замовлень, а кожне замовлення може містити багато продуктів;
- створення діаграми класів: команда повинна створити діаграму класів, яка відображає сутності, їх атрибути та зв'язки між ними. Ця діаграма може бути використана для комунікації з іншими розробниками, бізнес–аналітиками та експертами;
- валідація доменної моделі: команда повинна перевірити доменну модель на відповідність бізнес–домену та на наявність протиріччя. Для цього можна провести ревізію з експертами та бізнес–аналітиками, або використати автоматичні засоби валідації моделі;
- підтримка та розширення моделі: доменна модель може змінюватися відповідно до змін в бізнес–домені. Розробник повинен підтримувати та розширювати модель з часом, щоб забезпечити відповідність програмного забезпечення бізнес–домену.

Узагальнюючи, доменна модель це — важлива складова при розробці програмного забезпечення, яка допомагає розробникам та бізнес–аналітикам зрозуміти бізнес–домен, визначити ключові об'єкти та зв'язки між ними та побудувати програмне забезпечення, яке відповідає вимогам бізнесу.

Ручна валідація доменної моделі є затратним процесом, оскільки вимагає значних зусиль та часу, але ручна валідація також є критично важливою для успішної розробки програмного забезпечення, оскільки від неї залежать інші етапи розробки, такі як проектування бази даних, розробка архітектури програми та написання коду. Недостатньо провалідована доменна модель може призвести до помилок та неправильних рішень на різних етапах розробки, що зумовлює додаткові витрати та затримки. Тому

важливо виділити достатньо часу та ресурсів для валідації доменної моделі, а також використовувати автоматичні засоби валідації моделі для зниження витрат та покращення якості моделі.

Автоматичні засоби для валідації доменної моделі — це існуючі інструменти, які можуть використовуватися для автоматизації процесу валідації. Наприклад, інструменти для верифікації моделей на підтримку статичного аналізу коду, інструменти для автоматичної перевірки моделей на відповідність принципам SOLID або іншим правилам проектування програмного забезпечення, а також спеціалізовані програми для валідації концептуальних моделей даних. Щодо основних автоматичних засобів валідації, до них слід віднести наступне:

- CodeSonar [15]: інструмент для аналізу статичного коду, який допомагає виявити помилки в проекті засобами використання технології "аналізу діаграм викликів";
- SonarQube [16]: інструмент для статичного аналізу коду, який допомагає виявляти помилки в коді та забезпечує високу якість продукту;
- Visual Paradigm [17]: інструмент для розробки концептуальних моделей даних, який допомагає визначити основні сутності та їх відношення;
- Modelio [18]: інструмент для розробки концептуальних моделей даних та для валідації відповідності моделі принципам SOLID;
- Enterprise Architect [19]: інструмент для проектування та валідації доменних моделей, що дозволяє визначити бізнес–правила та взаємозв'язки між сутностями.

Зазначені інструменти є лише деякими з можливих засобів для автоматичної валідації доменної моделі, і краще обрати той, що найбільш відповідає потребам та вимогам конкретного проекту.

Формальний аналіз концептів є потужним інструментом, який може допомогти вирішити проблеми, пов'язані з ручною та автоматичною

валідацією доменної моделі. Використання цього методу дозволяє виявляти та виправляти помилки у доменній моделі на ранніх етапах розробки, що зменшує час та витрати на її подальшу розробку та тестування. Однією з проблем ручної валідації доменної моделі є складність її аналізу та визначення правильності розміщення концептів та їх зв'язків. Формальний аналіз концептів дозволяє використовувати математичні методи для опису властивостей та залежностей між концептами у доменній моделі, що забезпечує більш точну та повну валідацію. Автоматична валідація доменної моделі також може мати свої обмеження, зокрема, залежності між концептами можуть бути дуже складними та неочевидними для програми, що здійснює валідацію. Формальний аналіз дозволяє здійснювати аналіз на більш високому рівні абстракції, що допомагає виявляти та виправляти проблеми, які можуть бути пропущені під час автоматичної валідації. Перед тим як проглянути формальний аналіз концептів, слід оглянути методології роботи з моделями, що були впроваджені раніше.

1.2 Матричні техніки в об’єктно орієнтованому моделюванні

В загальному існує три матричних техніки для роботи з об’єктно орієнтованим концептуальним доменним моделюванням:

- Create, Read, Update, Delete (CRUD) матриця була початково запропонована в роботах, пов’язаних з інженерією інформації [20]. Ціллю даної матриці є ілюстрація відносин між об’єктами та процесами в яких вони беруть участь. Процес може створити, читати, оновлювати або видаляти об’єкт;
- Object–Oriented Structured Systems Analysis and Design Methodology (OOSSADM), що можливо перекласти як методологія дизайну та аналізу об’єктно орієнтованих структурованих систем, базується на підходах побудов складних систем. В даному підході сутності накладають послідовні обмеження на бізнес–властивості шляхом

використання діаграм послідовностей. Так як властивості можуть з’являтися в діаграмах багатьох сутностей, було запропоновано поняття матриці сутностей–властивостей [21] для того, що прослідити за тим, які сутності керуються якими властивостями;

- Model Driven, Existence Dependency Relation, Object Oriented Development (MERODE) – це об’єктно орієнтована методологія дизайну та аналізу [22; 23], яка є доповненням до UML в тому, що вона є точною та повною методологією. MERODE подає інформаційну систему завдяки визначенню конкретних властивостей, їхніх ефектів на бізнес–об’єкти та відповідні бізнес–правила. Подібно до OOSSADM, властивості ідентифікуються як незалежні концепти, що подається в таблиці властивостей–об’єктів, яка показує які властивості належать яким об’єктам. Кожен тип об’єктів має методи для кожної властивості. Дані методи реалізують створення, видалення об’єктів та зміну їхніх станів як послідовність подій (властивостей) відповідних типів.

Властивості в OOSSADM та MERODE можливо розглядати як елементарні процеси, що мають якийсь ефект (створенні, видалення, модифікація) на одному або декількох типах об’єктів. Таким чином, у найпростішій формі, дані три таблиці можуть показувати які об’єкти беруть участь у яких елементарних процесах. Одним зручним способом представлення даних таблиць є перехресні таблиці, де в прямокутній таблиці рядки вказують типи об’єктів, а колонки вказують на відповідні властивості. Якщо наявний хрестик на пересіченні певного рядка та колонки, то це значить, що дана властивість належить даному об’єкту.

Однією з головних різниць між даними трьома таблиці є те, як вони заповнюються. На початкових етапах таблиці заповнюються за допомогою базового класичного аналізу: проводяться інтерв’ю сесії із ключовими користувачами та використовується загальна логіка. Після цього, таблиці повинні бути перевіреними на основі деяких критерій якості. Зазвичай дані

критерії будуть намагатися ідентифікувати пропущені рядки або колонки за допомогою загальних та інтуїтивних правил таких як:

- для кожного типу об’єктів повинен бути по крайній мірі один процес, що створює об’єкти даного типу;
- кожен процес повинен читати деякі дані та інші правила;
- кожна властивість повинна належати як мінімум одному типові;
- кожен тип об’єктів повинен мати як мінімум дві властивості.

Серед всіх трьох підходів лише MERODE містить формальні критерії повноти та сформованості таблиці. Відповідні таблиці та матриці можливо порівняти до концепту із формального аналізу концептів, де типи сутностей можуть бути співвіднесеними з об’єктами у формальному аналізі концептів, а події можуть бути розглянуті як певні властивості.

1.3 Основні принципи формального аналізу концептів

Формальний аналіз концептів це новий математичний підхід, що може бути використаним як метод кластеризації без вчителя. Об’єкти, які беруть участь в одній групі подій, групуються в концепти. Перша точка аналізу це таблиця, яка складається із рядків M (об’єктів), колонок F (властивостей) та пересічень $T \subseteq M \times F$ (відношень між об’єктами та властивостями). Математична структура, яка посилається на таку таблицю називається формальним контекстом *(M, F, T)*. Приклад такого формального контексту відображено у табл. 1.1.

Таблиця 1.1

Приклад формального контексту

	enter	leave	acquire	classify	borrow	renew	return	sell	lose
Member	X	X			X	X	X		X
Book			X	X	X	X	X	X	X
Loan					X	X	X		X

У таблиці можливо помітити, що об’єкти відносяться до певної кількості властивостей (указано хрестиками); таким чином об’єкт відноситься до властивості лише тоді, якщо він таку властивість має. Маючи формальний контекст, формальний аналіз концептів визначає всі концепти із даного контексту і організує їх згідно з відношенням субконцепт–суперконцепт. Результатом цього є побудована лінійна діаграма (решітка).

Поняття концепту є центральним для формального аналізу концептів. Те, як формальний аналіз концептів розглядає концепт, співпадає з міжнародним стандартом ISO 704, який надає наступне визначення: концепт визначається як одиниця думки, що має інтенсіонал та екстенсіонал [8, 9]. До екстенсіоналу входять всі об’єкти, які належать до даного контексту, в той час як інтенсіонал (сенс концепту) містить всі властивості, які є спільними для його об’єктів. Зазвичай, в даному контексті розглядаються інформаційні атрибути, але тут також можливо розглянути поведінкові атрибути також (реакція на подію або участь в деяких процесах). Таким чином, проілюструємо нотацію формального контексту використовуючи дані з табл. 1.1. Для множини об’єктів $O \subseteq M$, властивості, що є спільними для всіх об’єктів o в множині O визначається наступним чином та позначається як $\sigma(O)$:

$$A = \sigma(O) = \{f \in F \mid \forall o \in O : (o, f) \in T\}. \quad (1.1)$$

Візьмемо для прикладу множину $O \subseteq M$, яка складається з об’єктів Member, Book та Loan. Ця множина O є сильно пов’язаною з другою множиною A, яка містить властивості “borrow”, “renew”, “return”, “lose”, так як вона містить спільні атрибути об’єктів з множини O. Таким чином, $\sigma(\{Member,\ Book,\ Loan\}) = \{borrow,\ return,\ renew,\ lose\}$. Також слід зазначити, що можливо сформувати зворотне відношення, знайшовши для множини властивостей A множину об’єктів, які ці властивості мають:

$$O = \tau(A) = \{i \in M \mid \forall f \in A : (i, f) \in T\}. \quad (1.2)$$

Повертаючись до даних в табл. 1.1. та взявши набір властивостей, що належать Loan, ми отримаємо множину властивостей {borrow, renew, return, lose}, якій відповідає множина $O \subseteq M$, що складається з об'єктів Member, Book та Loan. Таким чином $\tau(\{borrow,\ return,\ renew,\ lose\}) = \{Member,\ Book,\ Loan\}$.

Як можливо помітити, існує натуральне відношення між O як множиною об'єктів, які поділяють всі атрибути з множини A, та A як множиною всіх властивостей, що описують усі об'єкти, які містяться в множині O. Кожна така пара *(O, A)* називається формальним концептом у наданому контексті. Множина $A = \sigma(O)$ називається інтенсіал, в той час як множина $O = \tau(A)$ називається екстенсіал. Потрібно зазначити, що концепти є завжди максимальними в сенсі того, що множина O містить всі об'єкти, які мають властивості з множини A. Так само множина A містить всі спільні атрибути об'єктів із множини O.

Існує натуральний ієрархічний порядок між концептами у заданому контексті, який називається відношенням субконцепту–суперконцепту:

$$(O_1, A_1) \subseteq (O_2, A_2) \leftrightarrow (O_1 \subseteq O_2, A_2 \subseteq A_1). \quad (1.3)$$

Концепт $d = (O_1, A_1)$ називається субконцептом концепту $e = (O_2, A_2)$, або еквівалентно e називається суперконцептом d), якщо екстенсіал d є підмножиною екстенсіалу e, або якщо інтенсіал d є надмножиною інтенсіалу e. Наприклад концепт із інтенсіалом "enter", "leave", "lose", "return", "renew" та "borrow" є субконцептом концепту з інтенсіалом "lose", "return", "renew" та "borrow". Якщо знову скористатися даними з попередніх прикладів, то екстенсіал першого концепту складається з Loan, Member та Book, в той час як екстенсіал другого концепту складається лише з Member.

Множина усіх концептів всередині одного формального контексту об’єднана з визначеним відношенням субконцепту–суперконцепту для даних концептів задає повну решітку, яка називається концептуальна решітка $\beta(M, F, T)$ даного формального контексту. Решітку можливо представити для розгляду у вигляді розміченої лінійної діаграми. Лінійна діаграма на рис. 1.1 є компактним представленням концептуальної решітки формального контексту відображеного в табл. 1.1. Круги або вузли на діаграмі відображають формальні концепти, що є об’єктами і таким чином є частиною концептуальної решітки. Затінені прямокутники, які є під’єднаними до концептів, відображають властивості, які концепт має. Незатінені прямокутники, під’єднані до концептів, відображають набір елементів, що входять до даного концепту. Інформація, яка міститься в табл. 1.1, може бути пояснена за допомогою рис. 1.1, використовуючи наступне правило: об’єкт g описується властивістю m, якщо існує висхідний шлях від g до m. Наприклад, Member може бути описаним властивостями “enter”, “leave”, “lose”, “return”, “renew”, та “borrow.

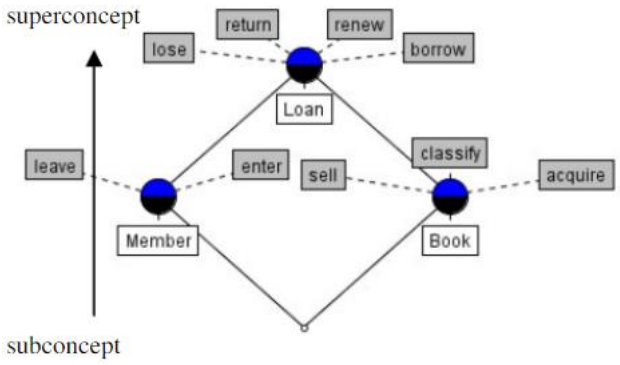

Рис. 1.1 – Решітка на основі табл. 1.1

Для того, щоб отримати екстенсіал формального концепту на основі лінійної діаграми слід зібрати всі об’єкти, йдучи низхідними шляхами від відповідного вузла. Наприклад, екстенсіалом найвищого концепту на рис. 1.1 є {Loan, Book, Member}. Для того, щоб отримати інтенсіал формального концепту, потрібно прослідкувати за всіма висхідними шляхами від

потрібного вузла та зібрати всі властивості. Наприклад, другий концепт в рядку має властивості “sell”, “classify”, “acquire”, “lose”, “renew”, “return”, та “borrow”. Найвищий та найнижчий концепти в решітці є особливими. Найвищий концепт містить всі об’єкти у своєму екстенсіалі. Найнижчий концепт містить всі властивості у своєму інтенсіалі, які містяться у решітці. Концепт є субконцептом всіх концептів, які можливо досягнути шляхами вверх. Наприклад перший концепт в рядку з екстінсіалом {Member} є субконцептом верхнього вузла з екстенсіалом {Loan, Member, Book}. В середні формального аналізу концептів концепт згенерований об’єктом P визначається наступним чином:

$$\gamma(P) = \Big(\tau\big(\sigma(P)\big), \sigma(P)\Big). \tag{1.4}$$

Паралельно, концепт, який створюється властивістю b визначається як:

$$\lambda(b) = \Big(\tau(b), \sigma\big(\tau(b)\big)\Big). \tag{1.5}$$

На лінійній діаграмі концепти позначені об’єктами, що створюють відповідний концепт C. Дані об’єкти називаються власними об’єктами концепту C.

1.4 Основні принципи методології MERODE

Ключовим поняттям методології MERODE є поняття залежність існування, яка накладається на структуру решітки (не слід путати із ієрархіями наслідування). Концепт залежності існування базується на понятті життя об’єкта. Життя об’єкта це проміжок між точкою в часі де об’єкт був створений та точкою в часі де він припинив своє існування. Залежність існування базується на двох рівнях:

- на рівні типів об’єктів та класів;
- на рівні появ об’єктів.

Залежність існування це – часткове упорядкування на об’єктах та їх типах і визначається наступним чином: нехай P та Q будуть типами об’єктів, де P екзистенційно–залежний на Q (тобто $P \leftarrow Q$), тоді і тільки тоді, коли кожна поява p типу P включена в існування лише одного і того самого об’єкта q типу Q. Таким чином, p називається залежним об’єктом, в той час як P називається залежим типом об’єктів. Об’єкт p є екзистенційно–залежним від об’єкта q, який називається об’єктом володарем. Тип об’єктів Q відповідно називається типом володарем.

Результатом цього є те, що існування залежного об’єкта не може початися до того, як буде створений об’єкт володар. Варто зазначити, що залежний об’єкт так само закінчує своє існування найпізніше в той же самий час як закінчує своє існування об’єкт володар.

Поняття залежності існування є схожим до поняття слабкої сутності та поняття об’єкта володаря, що було запропоноване в OOSSADM. В понятті відношення сутностей [24] ми можемо використати поняття слабкої сутності для відображення екзистенційно–залежних типів об’єктів, оскільки існування слабкої сутності залежить від існування інших об’єктів шляхом слабкої залежності. Залежність існування є еквівалентною поняттю поняттю слабкого відношення, яке є обов’язковим для кожної слабкої сутності. MERODE вимагає те, що об’єкти в концептуальній моделі повинні мати лише залежність існування. Таким чином, діаграма класів може бути представлена як граф залежності існування. Приклад такої діаграми класів, яку також можливо розглядати як повну решітку зображено на рис. 1.2.

Граф залежності існування визначається наступним чином: нехай M буде множиною типів об’єктів у концептуальній схемі. Граф залежності існування відношення $\leftarrow$, яке є мультимножиною[2] поверх $M \times M$, таке, що $\leftarrow$ задовольняє наступні обмеження:

1. Тип об’єктів ніколи не екзистенційно–залежний сам на себе:

$$\forall P \in M: (P, P) \notin \leftarrow \tag{1.6}$$

2. Залежність існування є ациклічною Це значить, що:

$$\forall n \in N, n \geq 2, \forall P_1, P_2, \dots, P_n \in M: \tag{1.7}$$
$$(P_1, P_2), (P_2, P_3), \dots, (P_{n-1}, P_n) \in \leftarrow \Rightarrow (P_n, P_1) \notin \leftarrow .$$

$\Leftarrow$ є нерефлексивним транзитивним замиканням $\leftarrow$, де $\Leftarrow \subseteq M \times M$ та виконуються наступні умови:

$$\forall P, Q \in M: (P, Q) \in \leftarrow \Rightarrow (P, Q) \in \Leftarrow \tag{1.8}$$

$$\forall P, Q, R \in M: (P, Q) \in \leftarrow \text{ та } (Q, R) \in \Leftarrow \Rightarrow (P, R) \in \Leftarrow. \tag{1.9}$$

На практиці MERODE вимагає, щоб граф залежності існування був повністю зв’язаним. Далі потрібно ввести поняття таблиці об’єктів–властивостей. Таблиця об’єктів властивостей це – таблиця, де рядок відповідає певному типу об’єктів, а колонка відповідає певній властивості. Кожна клітинка таблиці має або пропуск, або символ “X”, що значить те, що об’єкт має дану властивість. Нехай T $\subseteq M \times A \times \{" ", "X"\}$ таке, що:

$$\begin{aligned} &\forall P \in M, \forall a \in A: \\ &(P, a, " ") \in \text{Для } (P, a, \text{"X"}) \in T \\ &\forall P \in M: x(P) = \{a \in A \mid (P, a, "X") \in \text{T}\} \\ &A = \cup \{x(P) \mid P \in M\}. \end{aligned} \tag{1.10}$$

Таблиця показується як матриця, що містить один рядок для одного типу об’єктів та одну колонку для кожної властивості. “X” на пересіченні

указує, що дана властивість є елементом $x(P)$ (алфавіт P), де P тип, об’єктів із заданого рядка.

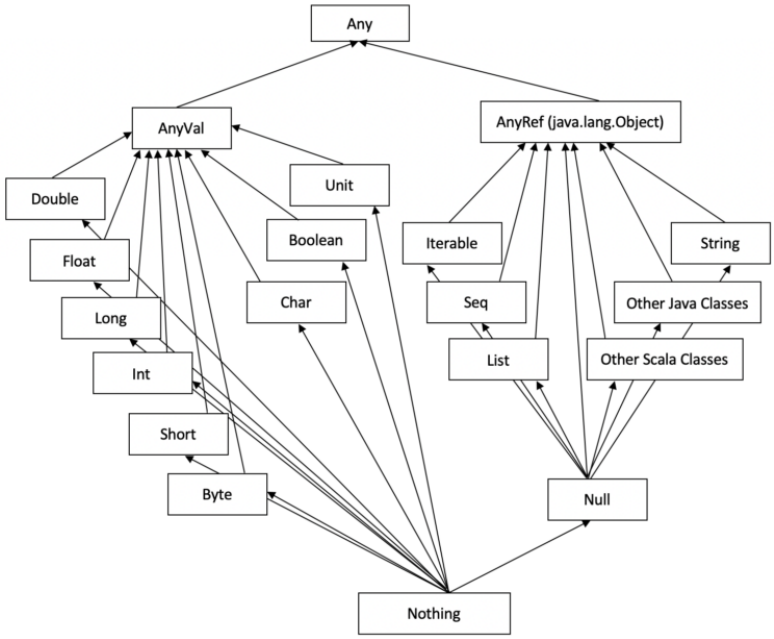

Рис. 1.2 – Приклад повної решітки

На додачу до самої таблиці, MERODE містить набір правил, що дозволяють перевірити правильність сформованої таблиці. Ці правила були сформовані на основі логіки якості моделі (повнота та узгодженість), життєвому циклі об’єктів та формалізації шляхом використання алгебри процесів. MERODE пропонує чотири правили:

1. корисний період існування доменного типу об’єкту має певну тривалість, яка може бути визначена двома подіями: момент коли об’єкт входить у домен, в якому ми зацікавлені та коли даний об’єкт покидає домен. Іншими словами, кожен тип об’єктів повинен брати участь у покрайній мірі у двох подіях: створення та знищення об’єкта. Для таблиці це значить, об’єкт повинен мати по крайній мірі два “X”, які не обов’язково повинні відповідати створенню та знищенню;

2. кожна властивість повинна бути використана по крайній мірі одним типом об’єктів. Для таблиці це значить те, що для кожної колонки є по крайній мірі один рядок з “X”;
3. дане правило можливо охарактеризувати як правило розповсюдження. Об’єктний тип володар має всі властивості, які мають його залежні типи. Наприклад, зміна стану боргу (повернення книги на місце) автоматично тягне за собою зміну відповідної книги та членства: книга повернута на полицю та членство має одну копію менше. Таким чином, якщо P екзистенційно–залежний від Q, алфавіт P повинен бути підмножиною алфавіту Q. Таке відношення називається правилом розповсюдження: $P \leftarrow Q \Rightarrow x(P) \subseteq x(Q)$;
4. четверте правило називається контрактним правилом. Воно говорить, що коли два типи об’єктів поділяють дві або більше властивості, то спільні властивості повинні бути у алфавіті одного або більше екзистенційно–залежних типів об’єктів:

$$\forall P, Q \in M: \big(x(P) \cap x(Q)\big) \geq 2 \wedge \neg\big(x(P) \subseteq x(Q) \vee x(Q) \subseteq x(P)\big) \Longrightarrow \exists R_1, \dots, R_n \in M: \forall i \in \{1, \dots, n\}: R_i \leftarrow P, Q \wedge x(R_1) \cup \dots \cup x(R_n) = x(P) \cap x(Q). \quad (1.11)$$

Наслідком цього є наступне: $x(P) \subseteq x(Q) \Longrightarrow P \leftarrow Q$. Варто помітити, що в MERODE, контрактне правило можливо використати лише в тому випадку, якщо наявності двох та більше спільних властивостей і одне з них повинно породжувати залежні типи, в той час як друге повинне їх нищити. За наявності лише однієї спільної властивості MERODE не вимагає наявності додаткового типу об’єктів, оскільки поняття типу об’єктів вимагає наявність по крайній мірі двох властивостей. Необхідність саме

двох властивостей була породжена фактом того, життєвий цикл вимагає породження та закінчення об’єкту.

1.5 Формальний аналіз концептів та матриці об’єктів–властивостей

Як було описано раніше, правила коректності були сформовані з використанням семантики залежності існування, загального глузду та алгебри процесів. Таким чином слід реорганізувати привали MERODE використовуючи терміни формального аналізу концептів:

- перше та друге правило можливо пояснити наступним чином: об’єкти, які не містять властивості, та властивості, які не належать жодному об’єктові, не мають жодної практичної користі в концептуальній моделі;
- третє правило говорить те, що принцип розповсюдження по шляхам залежності існування диктує, що решітка формального аналізу концептів та графа об’єктів–властивостей є певною мірою ізоморфними. Іншими словами, накладаючи правила коректності MERODE, ми отримуємо решітку в концептуальному об’єктно–орієнтованому аналізі. Зворотне є також вірним: відношення субконцепту–суперконцепту у формальному аналізі концептів є ізоморфним до відношення залежності існування;
- останнє четверте правило, нажаль, не має можливості перевірки, хоча й може бути описане за допомогою формального аналізу концептів.

1.5.1 Формальний аналіз концептів та матриці об’єктів–властивостей

Давайте розглянемо таблицю об’єктів–подій відображену на табл. 1.2. Решітка, яка відповідає заданій таблиці зображена на рис. 1.3. Задана таблиця

містить пустий рядок, що говорить про те, що є певний тип об’єктів *R*, який не має жодної властивості взагалі (або не бере участь у жодній події). У формальному аналізі концептів це значить, що тип об’єктів R є частиною екстенсіоналу верхнього концепту відповідної решітки (top). Таким чином, заданий концепт має пустий інтенсіонал. Описуючи інакше, є тип об’єктів, який немає нічого та не бере участь у жодній події.

Дана таблиця також маж пусту колонку, яка значить що жоден із типів об’єктів не маж властивість *e*, або не бере участь у події *e*. У формальному аналізі концептів це значить, що властивість або подія e є частиною інтенсіоналу нижнього концепту відповідної решітки (*bottom*). Даний концепт не містить типів об’єктів що значить, що є атрибут, який не є прив’язаним до якогось об’єкта.

MERODE розглядає дані два випадки як аномалії моделювання. Перше правило вимагає, що кожен тип об’єкту має як мінімум дві властивості, або бере участь в двох подіях, які відповідають за створення та видалення об’єктів. Таким чином, пусті рядки не є дозволеними. Дане правило MERODE може бути переписаним наступним чином: нехай P буде типом об’єктів, L буде концептом, а m та n властивостями:

$$\begin{aligned}&\text{Дано}: (M, F, T)\\&\forall P \in M:\\&\exists L \in \beta(M, F, T): P \in ext(L) \wedge \exists m, n: m, n \in int(L) \wedge m \neq n. \quad (1.12)\end{aligned}$$

MERODE також вимагає те, що кожна властивість є релевантною як мінімум для одного типу об’єктів, таким чином пусті колонки не є дозволеними. Варто зазначити, що з практичної точки зору, можливість зустріти властивість типу об’єктів, яка не належить жодному типу є малоймовірною. Дане правило MERODE може бути сформоване, використовуючи терміни формального аналізу концептів, наступним чином:

$$\text{Дано}: (M, F, T)$$
$$\forall P \in M:$$
$$\forall a \in F: \exists L \in \beta(M, F, T): ext(L) \neq \emptyset \wedge a \in int(L). \quad (1.13)$$

Таблиця 1.2

Приклад таблиці з пустими колонками та рядками

	a	b	c	d	e
P		X	X	X	
Q	X	X	X		
R					

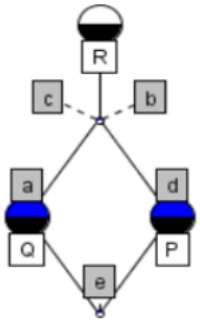

Рис. 1.3 – Решітка, яка відповідає табл. 1.2

1.5.2 Правило розповсюдження та відношення між концептами

Завдяки правилу розповсюдження, кожен об’єкт, що є частиною графа об’єктів–властивостей, породжує кортеж:

$$\varepsilon(P) = \left(P^*, x(P)\right), \quad (1.14)$$

де P^* це – множина всіх головних об’єктів P, а $x(P)$ це – алфавіт P. Таким чином $\varepsilon(P)$ співпадає із концептом з міткою P у лінійній діаграмі формального аналізу концептів. Крім того, якщо P екзистенційно–залежить

від Q, то тоді існує висхідний шлях від вузла з міткою Q до вузла з міткою P. P^* може бути перевизначене також наступним чином:

$$P^* = \{X \mid (P, X) \in \Leftarrow\} \cup \{P\}. \tag{1.15}$$

На основі цього можливо заявити, що $\varepsilon(P)$ є повноцінним концептом у формальному аналізі концептів. Щоб це показати потрібно довести, що $P^* = \tau\big(X(P)\big) = \{Q \in M \mid \forall e \ \in x(P) \mid e \ \in x(Q)\}$:

$$\begin{aligned} & Q \in P^* \\ & \Leftrightarrow (P, Q) \in \Leftarrow \\ & \Leftrightarrow x(P) \subseteq x(Q) \\ & \Leftrightarrow \forall e \in x(P) \colon e \in x(Q) \\ & \Leftrightarrow Q \in \tau\big(x(P)\big). \end{aligned} \tag{1.16}$$

Також на основі цього можливо встановити, що якщо $P \leftarrow Q$, то $\varepsilon(P)$ є суперконцептом $\varepsilon(Q)$. Доказом цього є наступне:

$$\begin{aligned} & \varepsilon(P) = \big(\{P\} \cup \{X | (P, X) \in \Leftarrow\}, x(P)\big) \\ & \varepsilon(Q) = \big(\{Q\} \cup \{Y | (Q, Y) \in \Leftarrow\}, x(Q)\big) \\ & (1): \\ & P \leftarrow Q \\ & \Rightarrow (Q \Leftarrow Y \Rightarrow P \Leftarrow Y) \\ & \Rightarrow \{X \mid (P, X) \in \Leftarrow\} \supseteq \{Y \mid (Q, Y) \in \Leftarrow\} \\ & \Rightarrow (\{P\} \cup \{X \mid (P, X) \in \Leftarrow\}) \supseteq (\{Q\} \cup \{Y \mid (Q, Y) \in \Leftarrow\}) \\ & (2): \\ & x(P) \subseteq x(Q) \\ & (1) + (2) \Longrightarrow \varepsilon(P) \text{ є суперконцептом } \varepsilon(Q). \end{aligned} \tag{1.17}$$

Далі можливо показати, що якщо $y(P)$ є суперконцептом $y(Q)$ у формальному аналізі концептів, то тоді $P \leftarrow Q$ в MERODE. Доводиться це наступним чином:

$$\begin{aligned} & y(P) = \Big(\tau\big(\sigma(P)\big), \sigma(P)\Big) \\ & y(Q) = \Big(\tau\big(\sigma(Q)\big), \sigma(Q)\Big) \\ & \Rightarrow int\big(y(P)\big) \subseteq int\big(y(Q)\big) \\ & \Rightarrow x(P) \subseteq x(Q) \\ & \Rightarrow P \leftarrow Q. \end{aligned} \tag{1.18}$$

Для того, щоб навести приклад слід повернутися до табл. 1.1. Множина подій, у якій бере участь Loan є підмножиною подій, у якій бере участь Member. В MERODE Loan є екзистенційно–залежним від Member. В той час як у формальному аналізі концептів, концепт створений об’єктом Loan є суперконцептом концепту створеним об’єктом Member. Таким чином, існує висхідний шлях від вузла з позначкою Member до вузла з позначкою Loan, як це зображено на рис. 1.1. Отже, можливо встановити відношення між термінами методології MERODE та термінами формального аналізу концептів і таким чином зв’язати разом відповідні об’єкти.

1.5.3 Формальний аналіз концептів та контрактне правило

Слід також обговорити те, як решітка об’єктів–властивостей може допомогти ідентифікувати потенційно пропущені типи об’єктів. Якщо решітка об’єктів властивостей містить концепт з двома або більше власними властивостями у своєму інтенсіалі та нуль типів об’єктів у своєму екстенсіалі, тоді ми маємо вузол з певними позначками властивостей, але без наявності відповідних позначок типів об’єктів. В MERODE це може нотифікувати нас про те, що тип об’єктів на один рівень нижче містить в своєму алфавіті

властивості, які не з'являються у властивостях одного або більше залежних типів об'єктів, що порушує контрактне правило. Тому, якщо решітка $B(O, A, I)$ містить концепт C такий, що $own\big(ext(C)\big) = \emptyset$ та $own\big(int(C)\big) \geq 2$, то тип об'єкту є відсутнім в таблиці об'єктів–властивостей згідно з контрактним правилом MERODE. Щоб це проілюструвати давайте уявимо, що також можливо зарезервувати книги, які не є на полицях. Якщо член бібліотеки змінює рішення, то повинна бути наявною можливість відмінити бронювання. Отже, події "reserve" та "cancel" були доданими до табл. 1.1. Результат цього відображений в табл. 1.3.

Таблиця 1.3

Розширена таблиця з новими властивостями

	enter	leave	acquire	classify	borrow	renew	return	sell	lose	reserve	cancel
Member	X	X			X	X	X		X	X	X
Book			X	X	X	X	X	X	X	X	X
Loan					X	X	X		X		

Затінена частина таблиці показує спільні властивості Member та Book. Деякі з властивостей є також частиною залежного типу Loan, хоча нові властивості не є частиною алфавіту спільного залежного типу. Якщо відобразити нову таблицю об'єктів–властивостей у вигляді решітки, то можливо помітити відсутність типу об'єктів, який відповідає за резервування (рис. 1.4).

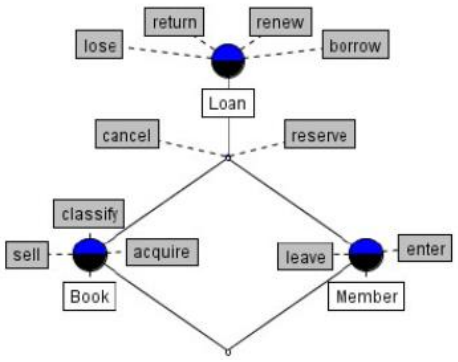

Рис 1.4 – Решітка на основі оновленої таблиці

Концепт із власними властивостями “reserve” та “cancel” не має жодного власного типу об’єктів у своєму екстенсіалі. Для того, щоб відповідати вимогам, які накладаються контрактним правилом MERODE, тип об’єктів, який має власні властивості, що містяться в інтенсіалі концепту, повинен бути доданий до доменної моделі додатку. Даний тип об’єктів повинен бути екзистенційно–залежним від типів об’єктів, що містяться в концептах одного рівня нижче в решітці. У даному випадку, згідно з контрактним правилом, дві властивості повинні бути або включеними в алфавіт Loan, або бути доданими до нового типу об’єктів, який буде названим Reservation, який буде залежати від обох Member та Copy. Згідно з основними практиками MERODE, останній підхід є більш рекомендованою ніж перший, оскільки борг може виникнути без резервування, а саме резервування може виникнути без будь–якого боргу. Змінена решітка відображена на рис. 1.5, в той час як сама оновлена таблиця об’єктів властивостей відображена на табл. 1.4.

Таблиця 1.4

Розширена таблиця з новим типом об’єктів

	enter	leave	acquire	classify	borrow	renew	return	sell	lose	reserve	cancel
Member	X	X			X	X	X		X	X	X
Book			X	X	X	X	X	X	X	X	X
Loan					X	X	X		X		
Reservation										X	X

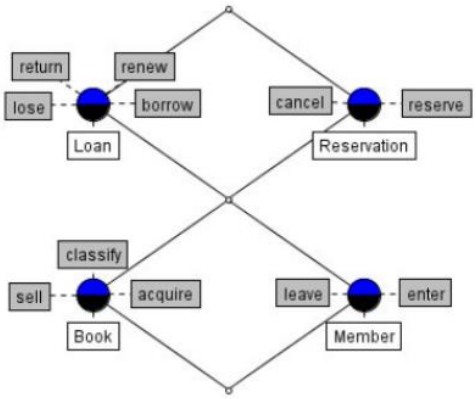

Рис 1.5 – Решітка після додавання нового типу об'єктів

Однак варто зазначити, що формальний аналіз концептів формально не має будь–якого підґрунтя для контрактного правила. Особливо, немає жодних вимог, що контрактне правило повинне працювати лише у випадку двох або більше спільних властивостей. Можливо створити новий тип об’єктів, якщо існує навіть один вузол з однією властивістю та без жодних власних типів об’єктів. Крім того, вузол зображений на рис. 1.3 вже має певний екстенсіал, тому не існує негайної необхідності створювати новий тип об’єктів у даній частині решітки. Як результат, формальний аналіз концептів не має формальних основ для даного правила. Навіть у MERODE дане правило було додане для перевірки наявності тупиків (deadlocks) [25], а не для того, щоб ідентифікувати пропущені типи об’єктів. Факт того, що формальний аналіз концептів не підтримує дане правило, може стати мотивацією для його перегляду.

Таким чином, на основі даних правил можливо сформувати новий метод валідації цілісності доменних моделей, який буде вимагати того, що решітка, яка буде побудована на основі даної, не повинна порушувати жодне з правил.

Висновки до розділу 1

У підрозділі 1.1 було розглянуто поняття доменної моделі та етапи її розробки. Було обговорено як ручні, так і автоматичні методи валідації доменної моделі, було показано, що дані методи мають як переваги, так і недоліки. На основі цього було згадано те, як формальний аналіз концептів може вирішити вказані недоліки.

У наступному підрозділі 1.2 були обговорені три матричні техніки, які можуть використовуватися під час роботи з доменними моделями. Було показано, що лише MERODE містить певні формальні правила, які можуть бути використані для валідації доменних моделей.

У підрозділі 1.3 було сформовано та показано основні принципи формального аналізу концептів. Було встановлено та показано можливі взаємозв’язки між концептами.

У передостанньому підрозділі 1.4 було обговорено основні принципи методології MERODE. Було сформовано та показано правила коректності таблиці об’єктів властивостей.

В останньому підрозділі 1.5 було обговорено те, як принципи та правила MERODE можливо поєднати з принципами формального аналізу концептів. Було показано, що комбінація MERODE та формального аналізу концептів здатна знаходити пропущені типи об’єктів, які слід розглянути як можливі кандидати для додавання.

РОЗДІЛ 2
ВИКОРИСТАННЯ ПРИНЦИПІВ ФОРМАЛЬНОГО АНАЛІЗУ КОНЦЕПТІВ ДЛЯ ВАЛІДАЦІЇ ЦІЛІСНОСТІ ДОМЕННОЇ МОДЕЛІ

2.1 Аналіз наявних засобів побудови решітки

Створення решітки для валідації моделі є важливим етапом у процесі розробки програмного забезпечення. Використання формального аналізу концептів для побудови решітки дозволяє отримати формальну специфікацію домену задачі і дозволяє провести глибинний аналіз взаємозв'язків між концептами. Основна мета побудови решітки – це створення множини концептів та їх взаємних відношень, що відображає домен задачі. Решітка може бути використана для валідації моделі, оскільки вона дозволяє візуалізувати структуру домену задачі та виявити можливі проблеми зі специфікацією моделі. Окрім цього, побудова решітки забезпечує зрозуміння взаємозв'язків між різними концептами в домені задачі. Це дозволяє розробникам отримати більш глибоке розуміння проблеми та допомагає виявити можливі прогалини в знаннях про домен. Отже, створення решітки за допомогою формального аналізу концептів є ефективним інструментом для валідації моделі та забезпечує більш глибокий аналіз домену задачі. Візуалізація взаємозв'язків між концептами дозволяє виявити можливі проблеми зі специфікацією моделі та забезпечує більш повне розуміння домену задачі, що є важливим для успішного розроблення програмного забезпечення.

Використання автоматичних засобів побудови решітки у формальному аналізі концептів має декілька переваг порівняно з побудовою решітки власноруч:

- швидкість: Ручна побудова решітки може бути дуже часовитратною та ресурсномісткою задачею, особливо коли маємо

справу з великим обсягом даних або складною моделлю. Автоматичні засоби можуть значно прискорити процес побудови решітки, зменшуючи час, необхідний для виконання даної задачі;

- точність: Ручна побудова решітки може бути вразливою до помилок та неточностей, особливо при великій кількості даних. Автоматичні засоби можуть знизити ризик помилок, забезпечуючи більш точні результати;
- масштабованість: Ручна побудова решітки може бути обмежена масштабом задачі. Автоматичні засоби можуть допомогти розширити масштаб задачі, що може бути особливо важливим для великих обсягів даних або складних моделей;
- інтеграція: Автоматичні засоби можуть бути легше інтегровані з іншими програмами та інструментами, такими як системи управління базами даних, звітність та аналітика. Це може забезпечити більш зручний та ефективний процес валідації моделі.

Таким чином, використання автоматичних засобів побудови решітки може забезпечити швидкий та точний процес валідації моделі у формальному аналізі концептів, збільшити масштабованість задачі та спростити її інтеграцію з іншими програмними засобами.

До основних засобів побудови решітки слід віднести:

- ConExp: програма на мові Java, яка включає в себе інтерфейс для введення даних та побудови решітки;
- ToscanaJ; інтерактивне середовище для побудови решіток, написане на мові Java;
- Galicia: програмний комплекс для побудови решіток та виконання інших операцій у формальному аналізі концептів;
- FCART: набір інструментів для формального аналізу концептів, який містить засіб для побудови решіток.

Слід оглянути переваги та недоліки кожного засобу для того, щоб визначити той, який найкраще підходить для вирішення поставленої задачі.

2.1.1 ConExp

ConExp [26] це – програма для побудови решіток у формальному аналізі концептів, розроблена у Інституті програмних систем Російської академії наук. Вона є вільним програмним забезпеченням, доступним на платформі Java, та може використовуватись для аналізу різноманітних даних.

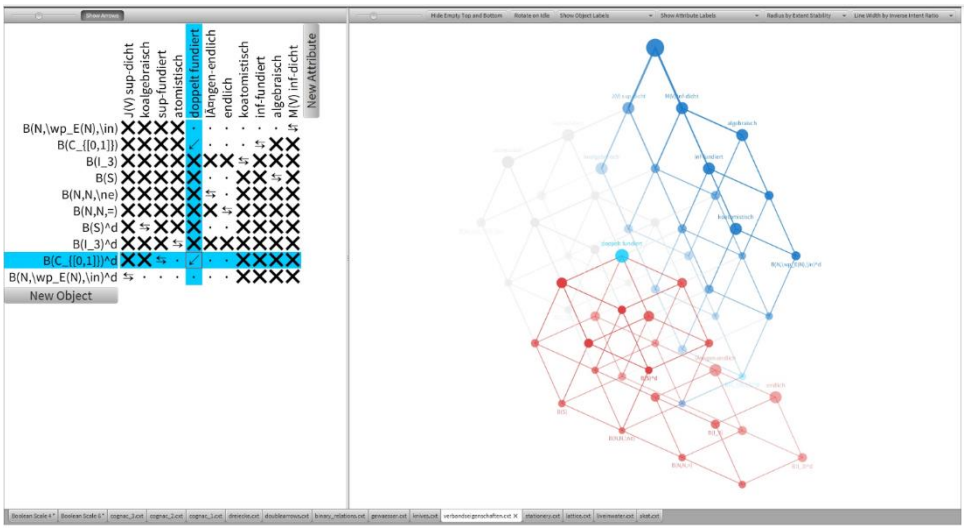

Рис. 2.1 – графічний інтерфейс ConExp FX

ConExp має наступні переваги:

- безкоштовний та відкритий код: ConExp є безкоштовним програмним забезпеченням з відкритим вихідним кодом, що означає, що його можна використовувати, змінювати та розповсюджувати безкоштовно;
- підтримує різні формати введення: ConExp підтримує різні формати введення, такі як CSV, XML, TXT, які дозволяють вам імпортувати дані з різних джерел;
- широкий набір функцій: ConExp має широкий набір функцій для роботи з концептуальними решітками, включаючи підтримку редагування, фільтрації, сортування, групування, агрегації та візуалізації даних;

- підтримка мови класів: ConExp підтримує мову класів, що дозволяє вам здійснювати більш точний формальний аналіз концептів;
- можливість створення власних функцій: ConExp дає можливість створювати власні функції на основі існуючих або з нуля, що дозволяє розширити можливості програми;
- платформонезалежний: ConExp підтримується на різних платформах, таких як Windows, Linux та Mac OS X, що дозволяє використовувати його на різних пристроях.

Нажаль, даний засіб має і певні недоліки:

- відсутність підтримки графічного інтерфейсу користувача: ConExp є консольною програмою, яка не має графічного інтерфейсу користувача. Це може зробити її використання не зручним для користувачів, які не знайомі з командним рядком;
- відсутність підтримки більш високих рівнів абстракції: ConExp не підтримує поняття метаданих, що може зробити складним або неможливим виконання аналізу для великих множин даних;
- відсутність механізму імпорту даних з різних форматів: ConExp не підтримує безпосереднє імпортування даних з різних джерел, що може зробити процес введення даних виснажливим і вимагати багато часу та зусиль;
- обмежені можливості генерації звітів: ConExp не надає багато можливостей для генерації звітів, що може ускладнити процес виведення результатів в зручному для аналізу форматі;
- старий рівень підтримки: останнє оновлення ConExp відбулося у 2005 році. Це може означати, що програма може мати певні проблеми з сучасними операційними системами або іншими програмними засобами;
- обмежена підтримка мов: ConExp не підтримує багато мов, що може стати проблемою для користувачів з різних країн, які не говорять англійською;

– відсутність підтримки роботи з великими даними: ConExp може мати проблеми з роботою з дуже великими масивами даних, що може вплинути на продуктивність та точність результатів.

Не дивлячись на відсутність інтерфейсу у базовій формі, існують певні модифікації такі як ConExp FX, які мають зручний інтерфейс користувача.

2.1.2 ToscanaJ

ToscanaJ [27] це – програмний засіб для побудови решіток та аналізу даних у формальному контексті. Він є продуктом наукового проекту, створеного в Італії в університеті Флоренції, та є вільним програмним забезпеченням, яке можна завантажити та використовувати безкоштовно. ToscanaJ є модульним програмним забезпеченням, яке додатково складається з програм Elba, Siena, Lucca.

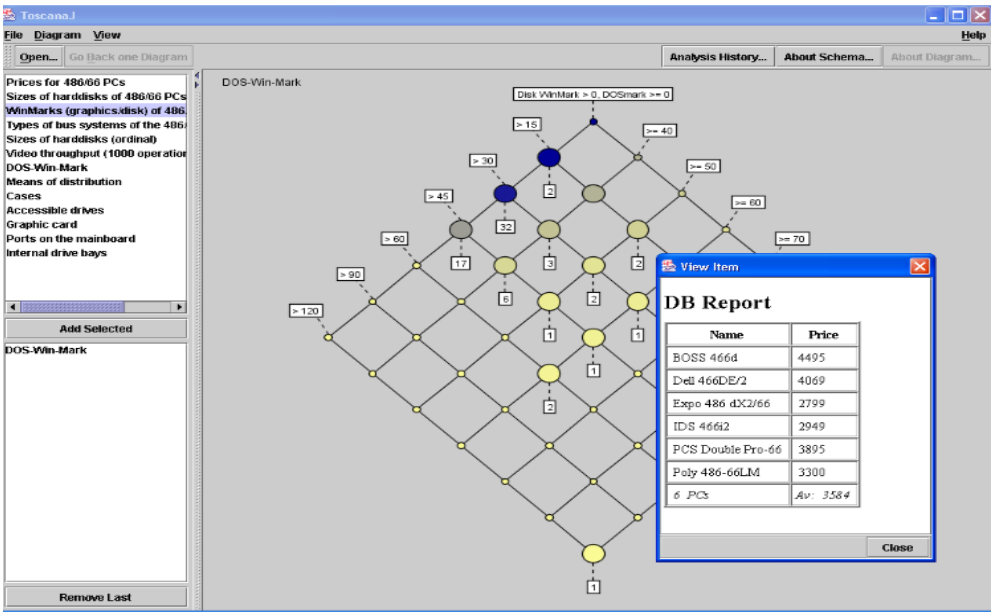

Рис. 2.2 – графічний інтерфейс ToscanaJ

До переваг ToscanaJ слід віднести наступне:

– візуалізація решіток: ToscanaJ надає гнучкі засоби для візуалізації решіток. Користувачі можуть вибирати різні кольори, розміри,

форми та інші параметри відображення решітки;

- підтримка різних форматів даних: ToscanaJ може працювати з різними форматами вхідних даних, включаючи XML, CSV, OAF та інші. Крім того, ToscanaJ підтримує імпорт та експорт даних у різних форматах;
- розширені засоби фільтрації: ToscanaJ має потужні засоби фільтрації, які дозволяють користувачам здійснювати різні операції з даними. зокрема, користувачі можуть фільтрувати дані за допомогою декількох критеріїв, використовувати логічні оператори для комбінування критеріїв та інші;
- автоматичне завантаження даних: ToscanaJ може автоматично завантажувати дані з баз даних, що дозволяє значно зменшити час, потрібний для введення даних вручну. Крім того, ToscanaJ підтримує підключення до різних типів баз даних;
- підтримка міжнародної локалізації: ToscanaJ підтримує міжнародні налаштування, що дозволяє користувачам працювати з даними у різних мовах та форматах. Крім того, ToscanaJ може автоматично перекладати тексти на інші мови;
- відкритість та зручність у використанні: ToscanaJ є відкритим програмним забезпеченням та має простий інтерфейс, що дозволяє користувачам легко вивчити основні функції.

ToscanaJ також містить свій власний набір недоліків:

- обмеженість функціоналу: ToscanaJ не має багато функцій порівняно з іншими програмами для формального аналізу концептів. Його основна функція – це побудова решітки з даних в форматі CSV. Інші функції, такі як ієрархічний кластерний аналіз та графічне відображення результуючої решітки, можуть бути недоступні або обмежені;
- обмежені можливості керування процесом побудови решітки: ToscanaJ не надає користувачам багато можливостей для керування

процесом побудови решітки. Це може бути проблемою для тих, хто хоче налаштувати параметри аналізу, наприклад, вибрати рівень значущості для усіх концептів;

- відсутність активної підтримки: Останнє оновлення ToscanaJ було випущене у 2013 році, що може свідчити про відсутність активної підтримки. Це може призвести до проблем зі сумісністю та безпекою даних в майбутньому;
- неповна документація: Документація ToscanaJ може бути неповною або недостатньою для новачків, що може призвести до складнощів у використанні програми;
- висока вимогливість до ресурсів: ToscanaJ може вимагати значних обчислювальних ресурсів для побудови решіток з великою кількістю об'єктів та атрибутів, що може бути обмеженням для користувачів з обмеженими обчислювальними ресурсами.

2.1.3 Galicia

Galicia [28] є інструментом для аналізу різних типів алгебраїчних структур, включаючи решітки. Він дозволяє будувати решітки, знаходити їхні морфізми, виконувати обчислення з концептами та досліджувати різні типи структур, такі як алгебри, групи та напівгрупи. Galicia базується на використанні мови програмування Python і може бути використаний як з інтерфейсом командного рядка, так і з графічним інтерфейсом користувача. До переваг Galicia слід віднести наступне:

- інтерактивний інтерфейс: Galicia має зручний та інтуїтивно зрозумілий інтерфейс, який дозволяє користувачеві взаємодіяти з програмою та виконувати різні операції з решітками;
- підтримка різних форматів: програма підтримує різні формати файлів, такі як CSV, TXT, XML тощо, що дозволяє зручно імпортувати та експортувати дані;

- різноманітність операцій: Galicia надає користувачеві широкий набір операцій, таких як зведення, декомпозиція, візуалізація тощо, що дозволяє виконувати різноманітні завдання з аналізу даних;
- швидкість: програма працює досить швидко, навіть з великими наборами даних, завдяки ефективному алгоритму обробки;
- підтримка різних мов: програма має підтримку різних мов, що дозволяє користувачам з різних країн працювати з нею на їх власній мові;
- відкритий код: Galicia є відкритою програмою з відкритим вихідним кодом, що дозволяє розробникам вносити зміни та доповнення до коду програми;
- можливість використання в інших дисциплінах: Galicia може бути корисною не тільки для формального аналізу концептів, а й для інших дисциплін, таких як соціологія, політична наука, економіка тощо;
- доступність: програма Galicia доступна безкоштовно та може бути встановлена на різні платформи, включаючи Windows, Mac OS та Linux.

Дана програма також не є позбавленою недоліків:

- обмежені можливості візуалізації: Galicia пропонує лише один тип візуалізації – граф. Це може бути недостатньо для показу складних взаємозв'язків між концептами;
- обмежена можливість редагування: Galicia дозволяє редагувати лише один елемент решітки за раз, що може бути не зручним для користувача, якщо він хоче змінити декілька елементів одночасно;
- відсутність автоматичного оновлення: під час додавання або видалення концептів з решітки, користувач повинен власноруч перебудувати решітку, щоб відобразити оновлені дані. Це може бути важким і часом затратним для користувача;

– відсутність можливості збереження проектів: Galicia не дозволяє зберігати створені проекти в файловій системі. Користувач повинен зберігати кожну решітку окремо, що може бути незручним і призводити до втрати даних.

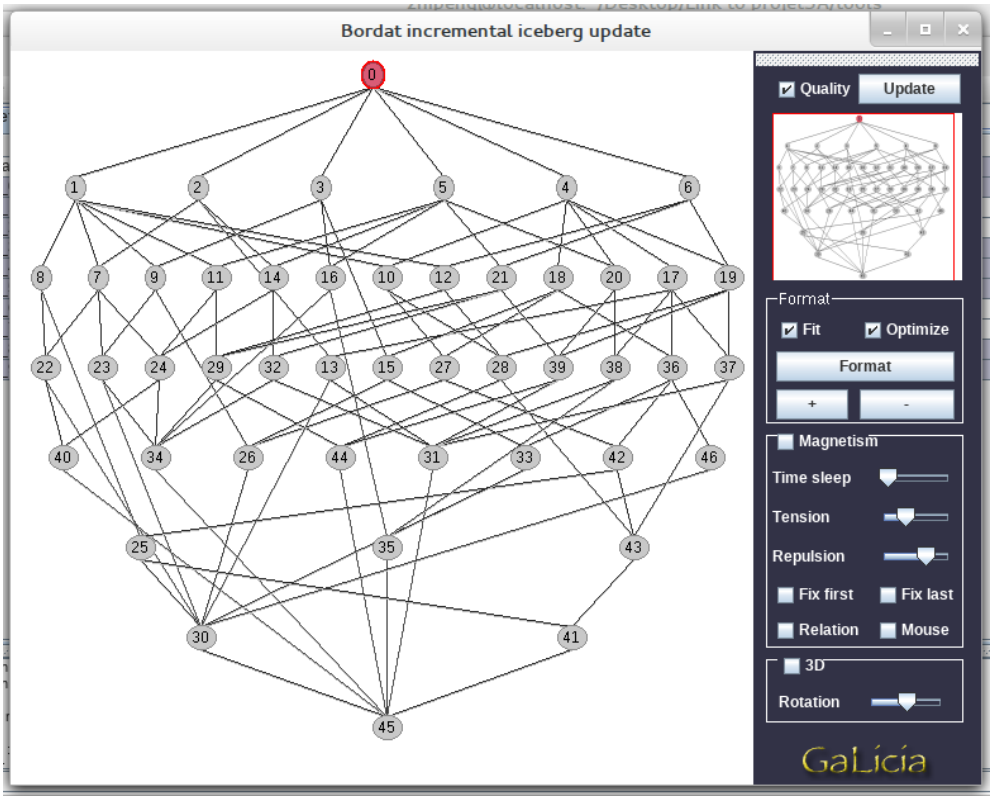

Рис. 2.3 – графічний інтерфейс Galicia

2.1.4 FCART

FCART [29] (Formal Concept Analysis Research Toolbox) це – програмне забезпечення для аналізу формальних концептів, що розроблене на мові програмування Java. Це відкрите програмне забезпечення, яке може бути використане для різних задач аналізу даних, зокрема для побудови решіток та візуалізації результатів аналізу даних у формі графів. Основною метою FCART є забезпечення дослідників та користувачів засобами для виконання аналізу формальних концептів та пов'язаних з ними завдань, таких як побудова решіток, пошук асоціативних правил, виявлення паттернів, тощо.

Завдяки широкому спектру функцій та модулів, FCART є потужним інструментом для виконання аналізу даних, який може бути використаний у різних дисциплінах, включаючи соціологію, психологію, лінгвістику, біологію та інші.

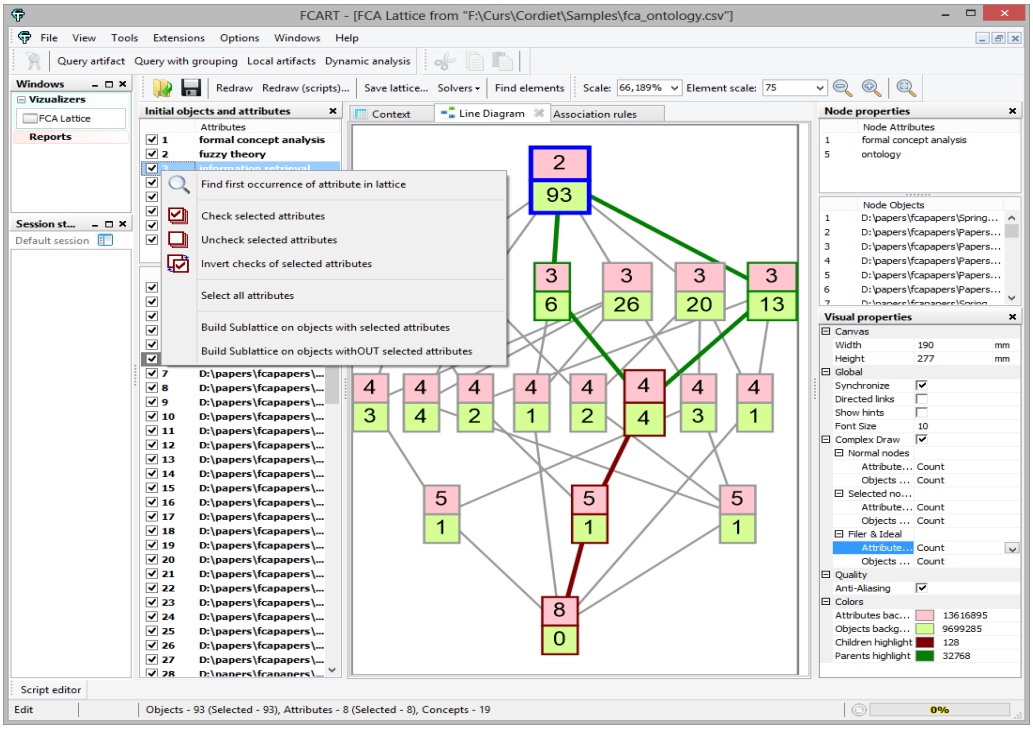

Рис. 2.4 – графічний інтерфейс FCART

Даний засіб має наступні переваги:

- відкритий код: FCART є безкоштовним та відкритим програмним забезпеченням, що дає змогу дослідникам вносити свої власні вклади у розвиток і використання цього засобу;
- різноманітність функцій: FCART містить багато корисних функцій, таких як відображення діаграм Фреймда або Галуа, пошук підмножин, що задовольняють задані критерії, та інші;
- можливість інтеграції з іншими інструментами: FCART може бути інтегрований з іншими інструментами, такими як R, для отримання більш глибокого аналізу даних;

- підтримка роботи з великими обсягами даних: FCART може обробляти великі обсяги даних, що дозволяє використовувати його для аналізу великих наборів даних.
- простота використання: FCART має дружній інтерфейс користувача, що дозволяє швидко навчитися його використовувати;
- підтримка різних алгоритмів побудови решіток: FCART підтримує різні алгоритми побудови решіток, що дозволяє дослідникам вибрати оптимальний алгоритм для своєї задачі;
- підтримка різних типів даних: FCART підтримує різні типи даних, включаючи числові, текстові та категоріальні дані;
- можливість редагування та збереження результатів: FCART дозволяє редагувати та зберігати результати аналізу в різних форматах, таких як CSV, HTML та інших.

Щодо недоліків даної програми, то вони є наступними:

- залежність від точності вхідних даних: точність результатів, отриманих з використанням FCART, залежить від якості вхідних даних. Якщо дані неповні або містять помилки, то це може привести до неточних результатів;
- високі вимоги до обчислювальних ресурсів: FCART вимагає великої кількості обчислювальних ресурсів для обробки великих масивів даних. Це може призвести до затримок у виконанні задач та необхідності використання потужніших комп'ютерів;
- відсутність можливості редагування решіток: після побудови решітки в FCART не передбачено можливості її редагування, тобто не можна додавати, видаляти або змінювати атрибути та їх відношення після побудови;
- обмежена підтримка форматів даних: FCART має обмежену підтримку форматів даних. Наприклад, він не підтримує роботу з великими текстовими файлами або базами даних;

- відсутність можливості мультиплатформенності: FCART був розроблений для роботи під операційними системами Windows, що обмежує його можливості використання на інших платформах.

Проаналізувавши чотири засоби побудови решітки, для виконання даної роботи було обрано саме ConExp, оскільки графічні модифікації ConExp [30] є зручними та інтуїтивними у використанні, не дивлячись на факт того, що оновлення базового засобу були припинені багато років тому.

2.2 Побудова доменної моделі

Після вибору засобу для будівництва слід обговорити та побудувати доменну модель, яка буде використовуватися для перевірки ідей, що було озвучені у першій частині даної роботи. Доменна модель запропонована у цій роботі є копію моделі системи аутентифікації та авторизації певної організації. Модель була очищена від будь–яких її елементів, що можуть виказувати на її походження для того, щоб не порушувати договір про нерозголошення. Слід зазначити, що не дивлячись на наявність більш сучасних засобів, дана модель та програма на основі неї використовуються, оскільки вони були створені велику кількість років тому, а для їхньої підтримки та оновлення було вже потрачено велику кількість фінансових та часових ресурсів. Беручи до уваги швидкість прийняття рішень в середині компанії розробника, час та вартість заміни сервісу аутентифікації, час та вартість навчання розробників й команди підтримки продукту, було прийняте рішення продовжувати використовувати стару модель та сервіс на її основі. Таким чином, будь–яке покращення існуючої моделі, яке можливо зробити за адекватний час та адекватні фінанси, є привабливою інвестицію для замовника, оскільки це дозволить покращити стабільність та ефективність моделі у довготривалій перспективі. У табл. 2.1 показано таблицю об’єктів–властивостей даної доменної моделі.

Таблиця 2.1

Доменна модель

	Full Name	Login	Hashed Password	Email	Registration Date	Last Login Date	Post Message
User	X	X	X	X	X	X	X
Moderator	X	X	X	X	X	X	X
Group Admin	X	X	X	X	X	X	X
Topic Admin	X	X	X	X	X	X	X
Super User	X	X	X	X	X	X	X

Продовження табл. 2.1

	Is Confirmed	Is Admin	Delete Message	Add Users	Remove Users	Change Group	Change Topic
User	X						
Moderator	X		X				
Group Admin	X	X	X	X	X	X	
Topic Admin	X	X	X	X	X		X
Super User	X	X	X	X	X	X	X

Продовження табл. 2.1

	Create Group	Create Topic	Delete Group	Delete Topic
User				
Moderator			X	X
Group Admin	X		X	
Topic Admin		X		X
Super User	X	X	X	X

Дана модель складається із наступних п’яти типів об’єктів:

- User: тип об’єктів, який відображає звичайних користувачів системи;
- Moderator: тип об’єктів, який відображає модераторів груп та тем у системі;
- Group Admin: тип об’єктів, які відображають адміністраторів груп у системі;
- Topic Admin: тип об’єктів, які відображають адміністраторів тем у системі;

- Super User: тип об’єктів, які відображають загального адміністратора всієї системи, здатного виконувати будь–які дії.

Крім самих об’єктів, дана модель також містить певний набір властивостей:

- Full Name: повне ім’я користувача;
- Login: логін користувача в системі;
- Hashed Password: хешований пароль користувача;
- Email: пошта користувача;
- Registration Date: дата реєстрації користувача;
- Last Login Date: дата останнього логіну в систему;
- Post Message: чи має користувач можливість відправити повідомлення у групу або тему;
- Is Confirmed: прапорець того, чи підтверджений даний користувач;
- Is Admin: прапорець того, чи є даний користувач адміністратором;
- Delete Message: прапорець того, чи може даний користувач видаляти повідомлення;
- Add Users: чи має даний користувач можливості додавати інших користувачів до групи, або теми;
- Remove Users: чи має даний користувач можливості видаляти інших користувачів з групи, або теми;
- Change Group: чи має даний користувач можливості змінювати групу;
- Change Topic: чи має даний користувач можливості змінювати тему;
- Create Group: чи має даний користувач можливості створювати групу;
- Create Topic: чи має даний користувач можливості створювати тему;
- Delete Group: чи має даний користувач можливості видаляти групу;
- Delete Topic: чи має даний користувач можливості видаляти тему.

Даній доменній моделі буде відповідати решітка зображена на рис. 2.5.

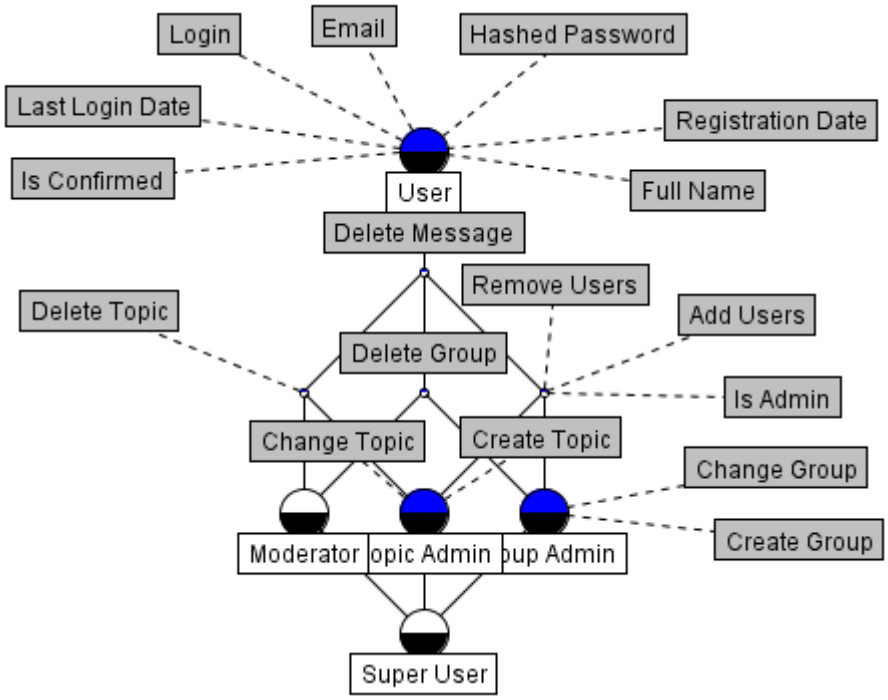

Рис. 2.5 – Решітка, яка відповідає табл. 2.1

2.3 Виправлення помилок у доменній моделі

Після аналізу решітки доменної моделі та використання правил MERODE адаптованих під формальний аналіз концептів були знайдені наступні помилки в доменній моделі, які можливо виправити:

- Moderator має властивості, які не є для нього необхідними: Delete Topic, Delete Group;
- пропущений тип об’єктів між Group Admin та Topic Admin, що призводить до дублікату коду у цих типах. Дана помилка була знайдена за допомогою контрактного правила MERODE.

Виправлення першої помилки дозволить досягнути кращої безпеки в самому додатку, не дивлячись на те, що вразливість є старою та прихованою за інтерфейсом користувача. Для того, щоб її виправити, слід забрати у типу Moderator властивості, які йому не належать, а саме властивості “Delete

Topic”, “Delete Group” Результат цих виправлень зображений на рис. 2.6 та табл. 2.2.

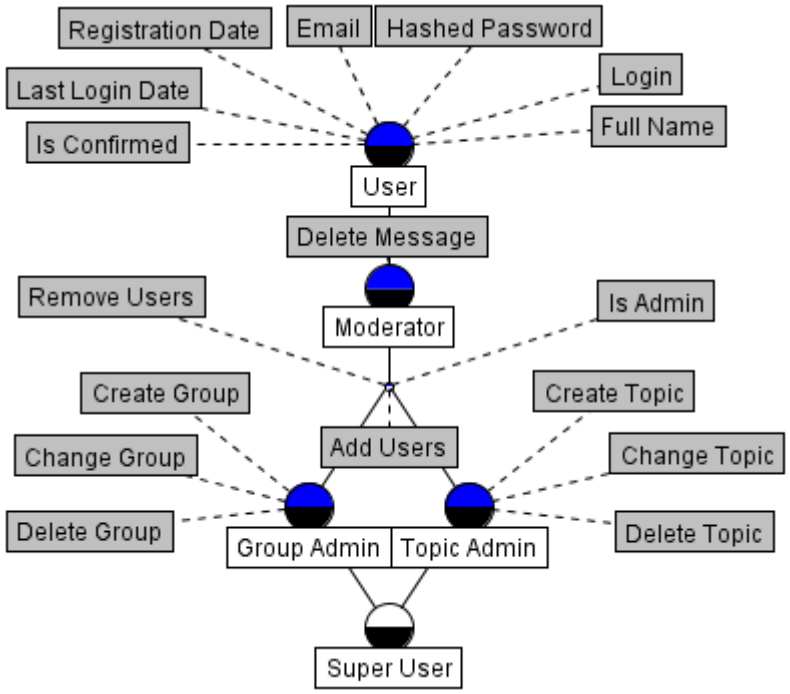

Рис. 2.6 – Решітка побудована після видалення зайвих властивостей

На зображеній решітці можливо помітити, що концепт Moderator став суперконцептом Group Admin та Topic Admin. Даний факт відображає те, що Moderator можливо реалізувати як предка класів Group Admin та Topic Admin для того, щоб збільшити можливість повторного використання коду, таким чином зменшуючи фактичну кількість коду, наявного в програмі.

Таблиця 2.2

Доменна модель після виправлення Moderator

	Full Name	Login	Hashed Password	Email	Registration Date	Last Login Date	Post Message
User	X	X	X	X	X	X	X
Moderator	X	X	X	X	X	X	X
Group Admin	X	X	X	X	X	X	X
Topic Admin	X	X	X	X	X	X	X
Super User	X	X	X	X	X	X	X

Продовження табл. 2.2

	Is Confirmed	Is Admin	Delete Message	Add Users	Remove Users	Change Group	Change Topic
User	X						
Moderator	X		X				
Group Admin	X	X	X	X	X	X	
Topic Admin	X	X	X	X	X		X
Super User	X	X	X	X	X	X	X

Продовження табл. 2.2

	Create Group	Create Topic	Delete Group	Delete Topic
User				
Moderator				
Group Admin	X		X	
Topic Admin		X		X
Super User	X	X	X	X

Наступним кроком є додавання спільного типу об’єктів, для Group Admin та Topic Admin, що дозволить перенести частини спільного коду у новий тип об’єктів, який буде предком типів об’єктів згаданих раніше. Дана зміна зменшить сумарну кількість коду, зробить його більш зрозумілим та зменшить необхідну кількість часу на додавання нових властивостей адміністраторам, якщо це буде необхідне у майбутньому, що дозволяє використати в реальності Open-Closed принцип SOLID, який говорить нам, що розширення наявного функціоналу повинно йти через наслідування. Результат даних виправлень зображений на рис. 2.7 та табл 2.3.

Таблиця 2.3

Доменна модель після додавання Admin

	Full Name	Login	Hashed Password	Email	Registration Date	Last Login Date	Post Message
User	X	X	X	X	X	X	X
Moderator	X	X	X	X	X	X	X
Admin	X	X	X	X	X	X	X
Group Admin	X	X	X	X	X	X	X
Topic Admin	X	X	X	X	X	X	X
Super User	X	X	X	X	X	X	X

Продовження табл. 2.3

	Is Confirmed	Is Admin	Delete Message	Add Users	Remove Users	Change Group	Change Topic
User	X						
Moderator	X		X				
Admin	X	X	X	X	X		
Group Admin	X	X	X	X	X	X	
Topic Admin	X	X	X	X	X		X
Super User	X	X	X	X	X	X	X

Продовження табл. 2.3

	Create Group	Create Topic	Delete Group	Delete Topic
User				
Moderator				
Admin				
Group Admin	X		X	
Topic Admin		X		X
Super User	X	X	X	X

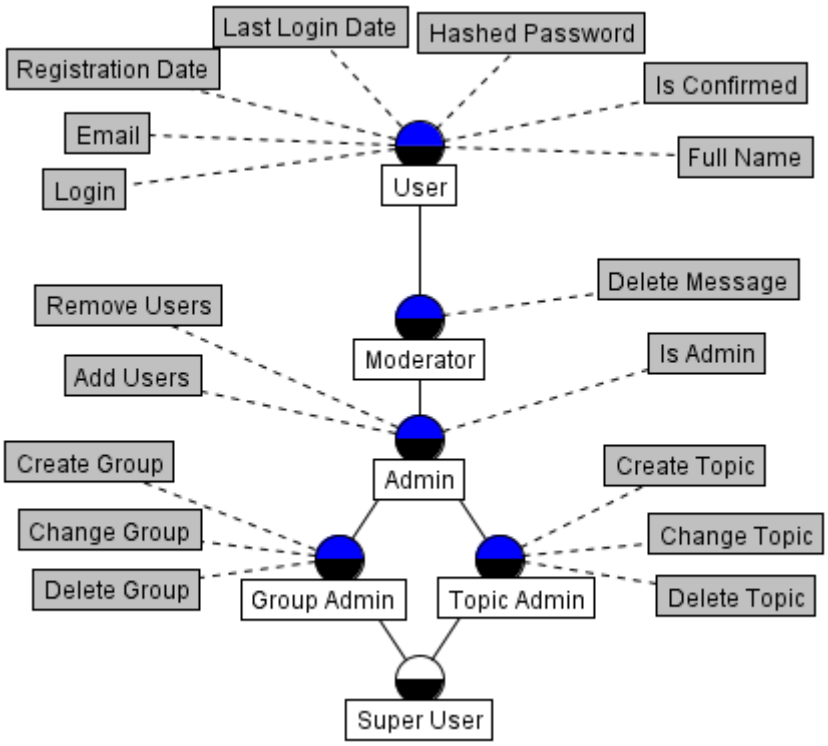

Рис. 2.7 – Решітка побудована після всіх виправлень

Як і під час минулого випадку, можливо помітити, що Admin став суперконцептом інших адміністраторів, дозволяючи нам зробити клас, який

його відображає, предком двох інших класів. Також використання принципів формального аналізу дозволило покращити доменну модель, збільшивши її безпечність та зменшивши її складність. Також важливо визнати, що побудова решітки виявилася корисною, оскільки вона допомогла зрозуміти те, що Moderator має властивості, які він мати не повинен. Додатково, всі виправлення доменної моделі зробили решітку більш легкою для розуміння, вирівнявши її структуру.

2.4 Аналіз результатів валідації доменної моделі

Важливо зазначити, що доменна модель, яка використовувалася для тестування ідей вказаних у даній роботі належить до старого та масивного проекту, де код був доволі складним для розуміння, оскільки проект значну частину часу розвивався без використання принципів дизайну програмного забезпечення. Цей факт зробив ручну перевірку доменної моделі складним та затратним процесом, що дозволило провести ефективну валідацію за допомогою принципів формального аналізу концептів. Таким чином, наступним кроком у розвитку правил валідації доменної моделі поданих у даній дипломній роботі є перевірка ефективності їх роботи використовуючи нові проекти, що слідували сучасним принципам дизайну та розробки програмного забезпечення.

Наступним кроком для покращення даних правил є пошук способів аналізу того, чи повинні дані типи об’єктів мати ті чи інші властивості за допомогою формальних правил. У даній роботі помилка у типі об’єктів Moderator була помічена власноруч після візуального аналізу решітки доменної моделі, що робить даний підхід неефективним за наявності ще більших доменних моделей. Однією із можливих ідей є перевірка типів об’єктів, що є сиблінгами на наявність надмірної кількості спільних властивостей, які не надходять від спільних предків. Для того, щоб підвищити ефективність використання даного методу можливо також створити

програмний застосунок здатний діставати доменні моделі із проектів, будувати формальний контекст на їхній основі та проводити відповідний аналіз. Даний програмний застосунок повинен мати єдиний формат введення для підтримуваних мов програмування, шо дозволить створювати окремі програми–модулі (адаптери), які вже в свою чергу надаватимуть можливість працювати з тією чи іншою мовою програмування.

Висновки до розділу 2

У підрозділі 2.1 було проаналізовано чому слід використовувати автоматичні засоби побудови решіток. Було розглянуто наступні засоби: ConExp, ToscanaJ, Galicia, FCART, серед яких було обрано саме ConExp.

У підрозділі 2.2 було оглянуто доменну модель, яка була буде використовуватися для перевірки можливості використання принципів формального аналізу концептів.

У підрозділі 2.3 було використано принципи описані у попередньому розділі для того, щоб виправити помилки в даній доменній моделі. Було показано що модель стала легшою для розуміння й використання та безпечнішою, оскільки один тип об’єктів має тепер лише притаманні йому властивості.

У останньому підрозділі 2.4 було проаналізовано ефективність використання принципів формального аналізу для валідації доменної моделі. Було встановлено, що ідеї подані в даній дипломній роботі слід додатково протестувати використовуючи доменні моделі з проектів, які слідували всім правилам розробки програмного забезпечення. Було зазначено що наступним можливим кроком є формалізація правил перевірки того, чи повинен даний тип об’єктів містити саме ті властивості, які йому були надані.

ВИСНОВКИ

Як результат виконання даної дипломної роботи було розроблено метод валідації доменних моделей із використанням принципів формального аналізу концептів, що показали свою ефективність на доменній моделі побудованій на основі моделі з реального проекту. Під час цього процесу було розглянуто інші приклади використання принципів формального аналізу для створення програмного забезпечення та його підтримки, що дозволило отримати декілька корисних ідей та уникнути можливих помилок.

Отриманий метод базується як на принципах матричної методології об’єктно орієнтованого дизайну та аналізу MERODE, які були об’єднані з поняттям концептів з формального аналізу концептів. Як результат, даний метод здатний ідентифікувати випадки, коли пропущений концепт, наявний пустий тип об’єктів, або існує властивість, що не є притаманною жодному типові об’єктів всередині доменної моделі. Варто зазначити, що створений метод ще не є готовим до практичного використання на будь–яких проектах, але ця робота створює підґрунтя для майбутніх досліджень у даній сфері.

У вступі до дипломної роботи було обговорено проблему верифікації та валідації програмного забезпечення, розглянуто схожі дослідження у даній сфері, сформовано предмет, об’єкт та мету дослідження. Перша частина даної дипломної роботи містить огляд кроків необхідних для побудови доменної моделі, аналіз матричних технік для роботи з доменними моделями та розгляд принципів формального аналізу концептів і його взаємодію із методологією MERODE. Друга частина містить мотивацію необхідності використання автоматизованих засобів для побудови решіток, створення решітки для заданої доменної моделі та її аналіз з використанням правил сформованих раніше.

В кінці роботи було проведено аналіз ефективності розробленого методу, де було заявлено про необхідність подальших тестів із

використанням інших проектів. Було обговорено можливі подальші покрашення алгоритму та способи, які можуть спростити його використання.

Отже, на основі даної дипломної роботи та результатів, які були описані вище, можливо дійти до висновку, що використання принципів формального для валідації цілісності доменних моделей є виправданим та вимагає подальших досліджень, таким чином вдалося досягти необхідних цілей та мети, а також вирішити поставлені завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Verification and validation in systems engineering / M. Debbabi et al. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. URL: https://doi.org/10.1007/978-3-642-15228-3 (date of access: 04.05.2023).
2. Wand Y., Weber R. Research commentary: information systems and conceptual modeling–a research agenda. Information systems research. 2002. Vol. 13, no. 4. P. 363–376. URL: https://doi.org/10.1287/isre.13.4.363.69 (date of access: 04.05.2023).
3. Paige R. F., Ostroff J. S. The single model principle. The journal of object technology. 2002. Vol. 1, no. 5. P. 63. URL: https://doi.org/10.5381/jot.2002.1.5.c6 (date of access: 04.05.2023).
4. Jackson M. A. System development. Englewood Cliffs, N.J : Prentice/Hall, 1983.
5. Snelting G. Software reengineering based on concept lattices. Fourth european conference on software maintenance and reengineering, Zurich, Switzerland. URL: https://doi.org/10.1109/csmr.2000.827299 (date of access: 04.05.2023).
6. Snelting G., Tip F. Reengineering class hierarchies using concept analysis. ACM SIGSOFT software engineering notes. 1998. Vol. 23, no. 6. P. 99–110. URL: https://doi.org/10.1145/291252.288273 (date of access: 04.05.2023).
7. Tonella P., Antoniol G. Inference of object-oriented design patterns. Journal of software maintenance and evolution: research and practice. 2001. Vol. 13, no. 5. P. 309–330. URL: https://doi.org/10.1002/smr.235 (date of access: 04.05.2023).

8. Düwel S. Enhancing system analysis by means of formal concept analysis. Advanced information systems engineering 6 th doctoral consortium, Heidelberg, 1999.
9. Hesse W., Düwel S. Identifying candidate objects during system analysis. Proc. caise'98/ifip 8.1 3rd int. workshop on evaluation of modeling methods in system analysis and design (EMMSAD), 1998.
10. Richards D., Boettger K., Aguilera O. A controlled language to assist conversion of use case descriptions into concept lattices. Lecture notes in computer science. Berlin, Heidelberg, 2002. P. 1–11. URL: https://doi.org/10.1007/3-540-36187-1_1 (date of access: 04.05.2023).
11. Richards D., Boettger K., Fure A. Using RECOCASE to compare use cases from multiple viewpoints. Australian conference on information systems, Melbourne, 2002.
12. Design of class hierarchies based on concept, (Galois) lattices / R. Godin et al. Theory and practice of object systems. 1998. Vol. 4, no. 2. P. 117–134. DOI:10.1002/ (sici)1096-9942(1998)4:2%3C117::aid-tapo6%3E3.0.co;2-q (date of access: 04.05.2023).
13. Quality in conceptual modeling – new research directions / G. Poels et al. Lecture notes in computer science. Berlin, Heidelberg, 2003. P. 243–250. URL: https://doi.org/10.1007/978-3-540-45275-1_22 (date of access: 04.05.2023).
14. Wlaschin S. Domain modeling made functional: tackle software complexity with domain-driven design and F#. Pragmatic Bookshelf, 2018. 312 p.
15. CodeSonar static application security testing (SAST) software tool | grammatech. Grammatech. URL: https://www.grammatech.com/ our-products/codesonar/ (date of access: 04.05.2023).

16. Self-managed | SonarQube. Clean Code Quality, Security & Developer First Tools | Sonar. URL: https://www.sonarsource.com/products/sonarqube/ (date of access: 04.05.2023).
17. Ideal modeling & diagramming tool for agile team collaboration. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/ (date of access: 04.05.2023).
18. Modelio Open Source - UML and BPMN free modeling tool. Modelio Open Source - UML and BPMN free modeling tool. URL: https://www.modelio.org/index.htm (date of access: 04.05.2023).
19. Sparx systems. UML modeling tools for Business, Software, Systems and Architecture. URL: https://sparxsystems.com/ (date of access: 04.05.2023).
20. James M. Information engineering. Englewood Cliffs, N.J : Prentice Hall, 1990.
21. Keith R. Object oriented SSADM. New York : Prentice Hall, 1994. 524 p.
22. Object-oriented enterprise modelling with MERODE / ed. by S. Monique. Leuven : Leuven University Press, 1999. 227 p.
23. Snoeck M., Dedene G. Existence dependency: the key to semantic integrity between structural and behavioral aspects of object types. IEEE transactions on software engineering. 1998. Vol. 24, no. 4. P. 233–251. URL: https://doi.org/10.1109/32.677182 (date of access: 04.05.2023).
24. Chen P. P. S. The entity-relationship approach to logical database design. Wellesley, Mass : QED Information Sciences, 1991. 83 p.
25. Dedene G., Snoeck M. Formal deadlock elimination in an object oriented conceptual schema. Data & knowledge engineering. 1995. Vol. 15, no. 1. P. 1–30. URL: https://doi.org/10.1016/0169-023x(94)00031-9 (date of access: 04.05.2023).

26. The concept explorer. URL: https://conexp.sourceforge.net/ (date of access: 04.05.2023).
27. ToscanaJ. URL: https://toscanaj.sourceforge.net/ (date of access: 04.05.2023).
28. Galicia lattice builder home page. Département d'informatique et de recherche opérationnelle - Université de Montréal. URL: http://www.iro.umontreal.ca/~galicia/ (date of access: 04.05.2023).
29. Formal concept analysis research toolbox (FCART). Факультет комп'ютерних наук – Національний дослідницький університет "Вища школа економіки". URL: https://cs.hse.ru/en/ai/issa/proj_fcart (date of access: 04.05.2023).
30. ConExp-NG. URL: https://github.com/fcatools/conexp-ng (date of access: 04.05.2023).

АНОТАЦІЯ

Дана дипломна робота складається зі вступу, 2 розділів, висновків, списку використаних джерел. Загальний обсяг роботи становить 56 сторінок, де на 48 сторінках викладена основна частина, враховуючи 12 рисунків та 7 таблиць. Документ містить 30 посилань на використані джерела, що займають 4 сторінки.

Дане дослідження присвячене проблемі ефективності та доцільності використання формального аналізу концептів для валідації доменної моделі. Його метою є спрощення процесу перевірки цілісності даних моделей. Під час виконання даної роботи було розглянуто саму ідею доменної моделі, принципи формального аналізу концептів, та те як їх можливо використати для аналізу моделей. Для досягнення поставленої мети була створена штучна доменна модель, яка базується на існуючій моделі з реального проекту, побудовано решітку даної моделі та використано ідеї формального аналізу концептів для валідації її цілісності.

Результатом даного дослідження є успішний приклад використання формального аналізу концептів для валідації цілісності доменної моделі. Областю використання запропонованої моделі та методу є прискорення та полегшення процесу валідації цілісності доменної моделі програмного забезпечення.

Ключові слова: розробка програмного забезпечення, формальний аналіз концептів, доменна модель.

ANNOTATION

This diploma work consists of the introduction, 2 chapters, conclusions, list of references. The document contains 56 pages in total, where 48 pages are occupied by the main part including 12 images and 7 tables. The diploma paper contains 30 references at 4 pages.

The research is dedicated to the problem of efficiency and suitability of applying principles of the formal concept analysis to validate domain model consistency. Its target is to simplify the process of validation of the domain model consistency. Domain models, principles of the formal component analysis and their application for validating domain models were reviewed during the research. The artificial domain model based on real model from the project was created, a lattice out of it was built and principles of formal concept analysis were applied to it in order to achieve the set goal.

The result of this research is a successful example of applying format concept analysis to validate consistency of a domain model. The proposed model and method can be used in order to accelerate and simplify the domain model validation process.

Key words: software development process, formal concept analysis, domain model.