МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 07.00.00.000 ПЗ

Група ІПм-23-1

Боднарчук Назарій

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Боднарчук Назарій Володимирович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Методології візуального рівня моделювання автономних програмних систем

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Боднарчук Н.В.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Піх Володимир Ярославович, к.т.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

Боднарчуку Назарію Володимировичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Методології візуального рівня моделювання автономних програмних систем ”

керівник проекту (роботи) Піх Володимир Ярославович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

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 певного класу

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

1. Дослідження предметної області моделювання автономних програмних систем
2. Дослідження та інтерпретація підходу керованого моделлю для розробки візуального рівня
3. Опис методу та мови генерації артефактів Model Driven Architecture
4. Імплементація методологій візуального рівня моделювання автономних програмних систем

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

1. Багаторівнева модель специфікації знань (рис. 1.1)
2. Механізми оновлення знань (рис. 1.2)
3. Підхід Model-Driven Software Engineering (рис. 1.3)
4. Зв’язки між SysML і UML (рис. 2.1)
5. Діаграми SysML (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 предметної області моделювання автономних програмних систем	15.10.2024	виконано
4	Дослідження та інтерпретація підходу керованого моделлю для розробки візуального рівня	08.11.2024	виконано
5	Опис методу та мови генерації артефактів Model Driven Architecture	20.11.2024	виконано
6	Імплементація методологій візуального рівня моделювання автономних програмних систем	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 80 с., 58 рис., 4 табл., 53 джерела.

Тема: Методології візуального рівня моделювання автономних програмних систем

Об'єкт дослідження: автономні програмні системи, що розробляються за допомогою методологій візуального моделювання.

Мета роботи: розробка ефективного підходу до візуального моделювання автономних програмних систем з використанням методології SYSMOD і створення прототипу інструмента візуального моделювання для специфікацій знань.

Предмет дослідження: методи, методології та інструменти візуального моделювання автономних програмних систем із використанням системного підходу на основі моделей.

Результати дослідження:

Запропоновано гібридний підхід до розробки інструментів візуального моделювання для автономних програмних систем і розроблено прототип програмного забезпечення для візуалізації специфікацій, що дозволяє створювати автономні програмні системи з автоматичною генерацією коду.

Висновок

Реалізовано системний підхід до розробки засобів візуального моделювання для предметно-специфічних мов та створено прототип програмного забезпечення візуального моделювання.

АВТОНОМНІ ПРОГРАМНІ СИСТЕМИ, СИСТЕМНА ІНЖЕНЕРІЯ НА ОСНОВІ МОДЕЛІ, ВІЗУАЛЬНЕ МОДЕЛЮВАННЯ, ГЕНЕРАЦІЯ КОДУ, СПЕЦИФІКАЦІЯ ЗНАНЬ, ІНСТРУМЕНТИ МОДЕЛЮВАННЯ.

ABSTRACT

Master Thesis: 80 pp., 58 fig., 4 tab., 53 sources.

Thesis Subject: Methodology of visual level modeling of autonomous software systems

The object of research: autonomous software systems developed using the visual modeling methodology.

The purpose of the work: development of an effective approach to visual modeling of autonomous software systems using the SYSMOD methodology and creation of a prototype of a visual modeling tool for knowledge specification.

Research subject: methods, methodology and tools of visual modeling of autonomous software systems using a system approach based on models.

Research results

A hybrid approach to the development of visual modeling tools for autonomous software systems is proposed, and a prototype software for specification visualization is developed, which allows the creation of autonomous software systems with automatic code generation.

Conclusion

A systematic approach to the development of visual modeling tools for subject-specific languages was implemented and a prototype of visual modeling software was created.

AUTONOMOUS SOFTWARE SYSTEMS, MODEL-BASED SYSTEMS ENGINEERING, VISUAL MODELING, CODE GENERATION, KNOWLEDGE SPECIFICATION, SIMULATION TOOLS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ADS - Autonomous Driving System.

API - Application Programming Interface.

DDT - Dynamic Driving Task.

DSL - Domain Specific Language.

HKnowLang - Hybrid Knowledge Representation Language

MDSE - Model-Driven Systems Engineering.

ODD - Operational Design Domain.

SAE - Society of Automotive Engineers.

SOTIF - Safety Of The Intended Functionality.

SysML - Systems Modelling Language.

SYSMOD - SYStem MODeling.

VE - Visual Element.

VL - Visual Layer.

ВСТУП

Актуальність теми.

Автономні транспортні засоби здійснять значну трансформацію автомобільної промисловості. Представлення знань і логічне виведення в автономних транспортних засобах є ключовим аспектом досліджень для забезпечення функціональної безпеки таких систем. Мова HKnowLang була розроблена спеціально для досягнення цієї мети.

HKnowLang (Hybrid Knowledge Representation Language) — це предметно-орієнтована мова (Domain-Specific Language, DSL), спеціально розроблена для представлення знань та логічного виведення в системах, що вимагають високої надійності та функціональної безпеки, зокрема в автономних транспортних засобах. Вона поєднує різні підходи до представлення знань, включаючи онтології, правила логічного виведення та інші методи, що дозволяють моделювати складні системи й забезпечувати перевірку їхньої коректності. Основна мета HKnowLang — надати інструмент для моделювання поведінки автономних систем, таких як транспортні засоби, забезпечуючи при цьому, що ці системи дотримуються вимог безпеки, приймають правильні рішення в різних сценаріях та ефективно працюють у реальному часі.

На сьогодні HKnowLang має лише текстовий редактор, що обмежує її широке застосування в дослідженнях автономних транспортних засобів. Ця дисертація спрямована на додавання можливостей візуального моделювання до редактора HKnowLang шляхом розробки візуального шару (VL) з використанням підходу до системної інженерії, керованого моделями (MDSE).

В роботі представляється методологія MDSE на основі SYSMOD, яка реалізована для розробки візуального шару. Спершу визначаються вимоги до візуального шару, а потім моделюються варіанти використання, що відповідають цим вимогам. Ці варіанти використання реалізуються шляхом

проєктування візуального представлення конструкцій HKnowLang та розробки генератора, що перетворює моделі візуального шару на відповідний код. Далі ми здійснюємо валідацію візуального шару шляхом моделювання проєкту та перевірки згенерованого коду.

З розвитком автономних програмних систем і зростаючою складністю їх проєктування виникає потреба у нових підходах до моделювання таких систем. Методології, що базуються на моделюванні, зокрема MDSE, пропонують ефективний спосіб керування складністю шляхом абстракції систем на рівні моделей. Однак, наявні інструменти MDSE не повністю відповідають потребам розробки автономних систем. У цьому контексті актуальним є дослідження методологій візуального моделювання, таких як SYSMOD, для створення інструментів розробки автономних систем, зокрема для мов специфікацій знань (наприклад, HKnowLang), що використовуються в автономних транспортних засобах.

Процес розробки, представлений у цій роботі, пропонується як методологія для майбутньої розробки інструментів візуального моделювання для предметно-орієнтованих мов (DSL).

Мета дослідження - розробка ефективного підходу до візуального моделювання автономних програмних систем з використанням методології SYSMOD і створення прототипу інструмента візуального моделювання для специфікацій знань.

Об’єкт дослідження - автономні програмні системи, що розробляються за допомогою методологій візуального моделювання.

Предмет дослідження – методи, методології та інструменти візуального моделювання автономних програмних систем із використанням системного підходу на основі моделей.

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

- Дослідити та проаналізувати існуючі методології візуального моделювання, зокрема MDSE та SYSMOD, в контексті автономних програмних систем.

- Оцінити можливість застосування підходу SYSMOD для моделювання специфікацій знань у автономних системах на прикладі HKnowLang.
- Розробити прототип інструменту візуального моделювання, що дозволяє моделювати специфікації HKnowLang.
- Створити генератор коду для трансформації візуальних моделей у код HKnowLang, забезпечуючи автоматизацію процесу програмування.
- Перевірити працездатність створеного прототипу та його здатність до ефективного моделювання автономних систем.

Методи дослідження:

- Аналіз літератури та існуючих підходів: аналіз методологій візуального моделювання, зокрема MDSE та SYSMOD, для виявлення їх потенціалу в контексті автономних програмних систем.

- Моделювання системи на основі SYSMOD: застосування методології SYSMOD для побудови візуального рівня (Visual Layer).

- Розробка програмного забезпечення: створення користувацького інтерфейсу та функціональних модулів для візуального моделювання із застосуванням Java бібліотеки.

- Генерація коду: розробка генератора коду для автоматичної трансформації моделей у код.

- Емпіричне тестування: перевірка прототипу програмного забезпечення для підтвердження концепції та функціонування створеного інструменту.

Наукова новизна отриманих результатів.

Запропоновано гібридний підхід до розробки інструментів візуального моделювання для автономних програмних систем, що поєднує методології MDSE та SYSMOD і розроблено прототип програмного забезпечення для візуалізації специфікацій HKnowLang, що дозволяє створювати автономні програмні системи з автоматичною генерацією коду.

Практичне значення магістерської роботи.

Створено прототип програмного забезпечення для візуального моделювання, що може бути використаний для розробки автономних транспортних засобів, зокрема для представлення знань і міркувань за допомогою HKnowLang.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 80 сторінок, і містить 58 рисунків, 4 таблиці, список використаних джерел із 53 найменувань.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ МОДЕЛЮВАННЯ АВТОНОМНИХ ПРОГРАМНИХ СИСТЕМ

1.1. Опис предметної області та визначення проблематики дослідження

Автономне керування прогнозується як один із ключових чинників революційних змін у майбутній мобільності [1], і багато країн, активно підтримують розробку та тестування самокерованих транспортних засобів [2]. Досягнення 4-го та 5-го рівнів автоматизації за стандартом SAE [3] є необхідним для реалізації цього бачення майбутньої мобільності. Компанія BAESIS Automotive BV [4] має на меті подолати виклики, пов’язані з досягненням 4-го рівня автоматизації, застосовуючи HKnowLang [5 - 8] для представлення знань і логічного виведення в автономних транспортних засобах. HKnowLang була розроблена як текстова предметно-орієнтована мова (DSL) для автономних систем. У цьому дослідженні ми представляємо розробку візуального шару (VL) для забезпечення графічного моделювання в HKnowLang. Ми валідизуємо візуальний шар через моделювання проєкту у VL та перевірку згенерованого коду.

Автоматизація транспортного засобу може варіюватися від повністю керованого людиною до повністю автономного. Стандарт SAE International J3016B [3] надає таксономію, що визначає шість рівнів автоматизації — від відсутності автоматизації керування (SAE L0) до повної автоматизації керування (SAE L5).

Ці рівні автоматизації описуються в термінах Динамічного завдання керування (Dynamic Driving Task, DDT) та Операційної області дизайну (Operational Design Domain, ODD). Операційна область дизайну визначається як «умови експлуатації, в яких система автоматизації керування або її функції спеціально розроблені для роботи» [3].

Отже, точне визначення ODD та встановлення того, чи відповідають умови експлуатації визначеній ODD, є вирішальними для забезпечення функціональної безпеки систем автономного керування (ADS).

Мова формальних специфікацій HKnowLang [12, 13], пропонується для представлення знань і логічного виведення в автономних транспортних засобах 4-го рівня за стандартом SAE. Поточним обмеженням HKnowLang є те, що вона має лише текстовий редактор. Додавання можливостей візуального моделювання до редактора HKnowLang збільшить його зручність у застосуванні до автономних транспортних засобів і скоротить час розробки програмного забезпечення.

Розробка програмного забезпечення для автономних транспортних засобів є складним завданням. За оцінками, сучасний автомобіль містить понад 100 мільйонів рядків коду [14]. Очікується, що ця кількість рядків коду збільшиться для автономних транспортних засобів. HKnowLang розроблено як рішення для подолання складності розробки програмного забезпечення для автономних транспортних засобів. Однак наразі HKnowLang є текстовою мовою, реалізованою на Java [15]. Таким чином, використання HKnowLang в автомобільних проектах обмежене наступними проблемами, які дослідження ідентифікують як «три зла системної інженерії» [16]:

• Відсутність розуміння мови та знань для використання HKnowLang для вирішення проблеми автономних транспортних засобів. Вивчення семантики HKnowLang як текстової мови займає багато часу.

• Складність проектів автомобільного програмного забезпечення. Написання та підтримка численних рядків коду для автомобільного програмного забезпечення є складним завданням.

• Спілкування між зацікавленими сторонами. Розробка програмного забезпечення для автономних транспортних засобів залучає багато зацікавлених сторін. Важливо, щоб зацікавлені сторони могли зрозуміти систему, представлену в HKnowLang. Тому вимагати від зацікавлених сторін інтерпретувати код HKnowLang не є ідеальним для спілкування.

Для вирішення вищезазначених проблем можна застосувати модельно-керовану системну інженерію (MDSE) [17]. Редактор HKnowLang вимагає додавання VL для забезпечення можливостей візуального моделювання та підтримки підходу MDSE для розробки автономних транспортних засобів. Реалізація VL передбачає надання окремого візуального елемента (VE) для кожної з конструкцій специфікації HKnowLang. VL також потрібен для перетворення моделей у відповідний код HKnowLang, який можна скомпілювати в текстовому редакторі HKnowLang.

1.2. Багаторівнева модель специфікації знань

Метою цієї роботи є створення рівня візуалізації (VL), який забезпечить можливості візуального моделювання для редактора HKnowLang, тим самим забезпечуючи доказ концепції HKnowLang як життєздатного рішення для автомобільної промисловості.

Виконання мети даної роботи передбачає вирішення наступних завдань:

• Створення окремих візуальних елементів (VE) для кожної специфіки HKnowLang.

• Генерування коду HKnowLang, що відповідає моделі, розробленій користувачем у рівні візуалізації (VL).

Для цього проекту визначено таке обмеження:

• Створення візуальних елементів обмежується частиною онтології домену моделі специфікації HKnowLang.

HKnowLang — це структура для представлення знань та логічного виведення (Knowledge Representation and Reasoning, KRR) [5]. HKnowLang спрямована на ефективне та всеосяжне структурування знань і підвищення обізнаності на основі логічного та статистичного виведення. Знання в HKnowLang представляються за допомогою бази знань (Knowledge Base, KB), яка визначає контекст представлення знань (Knowledge Representation,

KR). HKnowLang є формальною мовою з багатошаровою моделлю специфікації знань, яка допомагає вирішувати такі завдання:

- Явне представлення концептів та відносин у доменній області.

- Явне представлення фактичних знань.

- Представлення невизначених знань за допомогою ймовірнісного підходу.

Важливою особливістю HKnowLang є представлення знань для самопристосовувальної поведінки.

HKnowLang реалізує багатошарову модель специфікації, зображену на рис. 1.1, для побудови структури бази знань у трьох рівнях:

- Корпуси Знань (Knowledge Corpuses).

- Оператори Бази Знань (KB Operators).

- Примітиви Виведення (Inference Primitives).

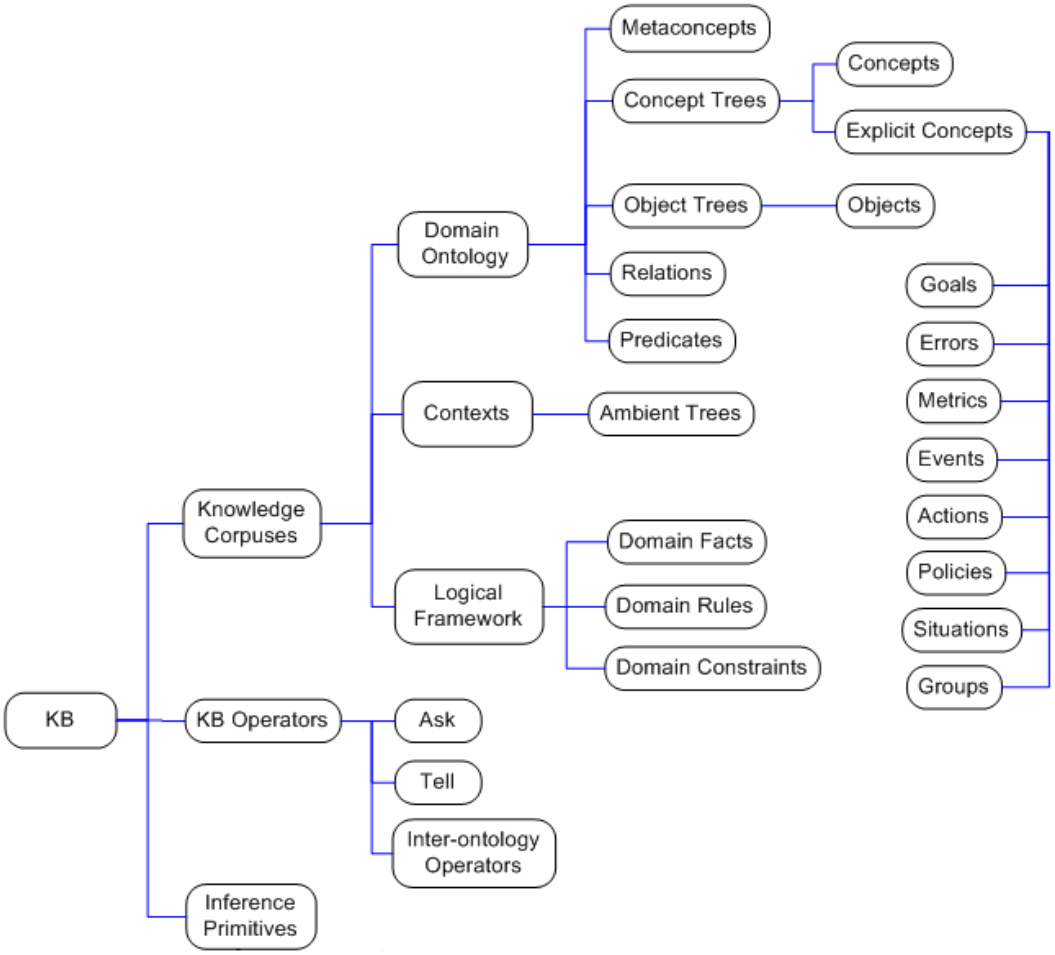

Рис. 1.1. Багаторівнева модель специфікації знань [5]

Корпуси знань використовуються для визначення структур представлення знань. Оператори бази знань забезпечують доступ до корпусів знань через спеціальні класи операторів ASK і TELL. Оператор ASK призначений для запитів та отримання знань, тоді як Оператори TELL дозволяють оновлювати знання. Логічна рамка забезпечує простий спосіб логічного виведення на основі правил, який альтернативно представлений через Онтологію Домену.

Три основні компоненти корпусу знань включають:

- Доменні онтології.
- Контексти.
- Логічний фреймворк.

Доменні онтології складаються з наступних елементів:

- Мета-концепти.
- Дерева концептів.
- Дерева об'єктів.
- Відносини.
- Предикати.

Мета-концепти дозволяють переглядати онтології з різних контекстних перспектив, встановлюючи різні значення для деяких ключових концептів. Дерева концептів складаються з семантично пов'язаних концептів та/або явних концептів. Кожен концепт має набір властивостей, функцій, батьківських та дочірніх концептів. Для логічного виведення кожен концепт, визначений у HKnowLang, має внутрішній атрибут стану, який може бути пов'язаний із набором можливих значень стану, в яких можуть перебувати екземпляри цього концепту. Явні концепти — це концепти, які обов'язково повинні бути присутніми в Базі Знань (KB) системи. Вони головним чином підтримують:

- Автономну поведінку системи.
- Розподілене виведення та обмін знаннями між системами.

Ці концепти можуть бути діями, помилками, метриками, групами, подіями, цілями, ситуаціями та політиками.

Дерева об'єктів — це концептуалізація того, як об'єкти існують у світі та як вони пов'язані між собою. Відносини пов'язують два концепти та можуть мати ймовірнісний розподіл. Відношення може мати опціональну назву, тобто коли назва відсутня, це стає неявним відношенням. Ймовірнісний розподіл підтримує ймовірнісне логічне виведення.

Предикати — це спеціальна структура Бази Знань, яка визначає відносини між станами або схеми для оцінки складних станів.

Моделювання знань у HKnowLang включає такі етапи:

- Початковий збір знань — із залученням експертів у галузі для визначення основних понять, відносин та функцій (операцій) доменної області.

- Визначення поведінки — ідентифікатори, ситуації та політики поведінки як "керувальні дані", що допомагають ідентифікувати важливі сценарії самопристосування.

- Структурування знань — інкапсуляція доменних сутностей, ситуацій та поведінки в структури HKnowLang, такі як концепти, об'єкти, відносини, факти та правила.

Контексти призначені для вилучення відповідних знань з онтології. Контексти забезпечують інтерпретацію для деяких мета-концептів. Контекст робить акцент на ключових концептах онтології, що допомагає механізму виведення звузити доменні знання, досліджуючи дерево концептів лише до зазначених ключових концептів. Контекст включає амбітні дерева. Амбітне дерево — це дерево концептів, описане онтологією, яке містить амбітні концепти, що є частиною дерева концептів, та опціональну інтерпретацію контексту.

Логічний фреймворк HKnowLang допомагає розробникам реалізувати явне представлення конкретних та загальних фактичних знань.

HKnowLang використовує спеціальні структури знань і механізм логічного виведення для моделювання автономної самопристосувальної поведінки [33]. Самопристосувальна поведінка виражається через політики, події, дії, ситуації та відносини між політиками і ситуаціями. Політики є основою автономної поведінки. Політика має мету, ситуацію політики, відносини політика-ситуація та умови політики, пов'язані з діями політики. Умова — це булевий вираз, що базується на онтології, наприклад, виникнення певної події.

Компілятор HKnowLang компілює модель Представлення Знань (KR), зазначену в HKnowLang, у HKnowLang Binary. HKnowLang Binary є ядром Бази Знань (KB), яка працює через механізм виведення HKnowLang.

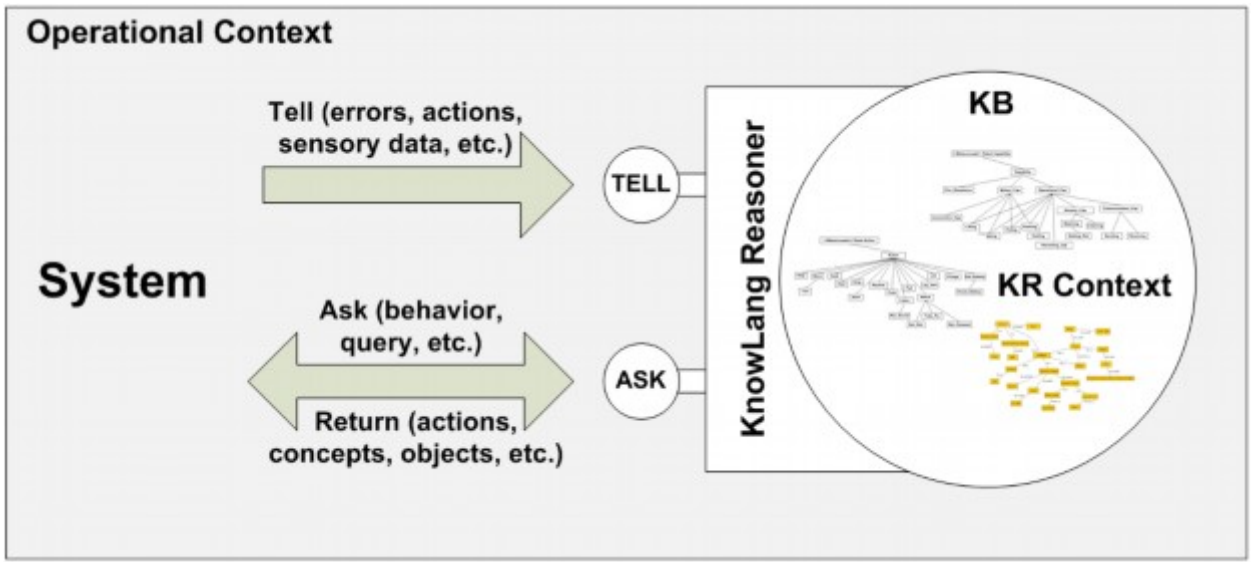

Рис. 1.2. Механізми оновлення знань

Механізм виведення HKnowLang надається як компонент для системи, в якій він працюватиме в операційному контексті системи, як і будь-який інший компонент системи. Він функціонує в контексті KR та з символами KR. Система взаємодіє з механізмом виведення через оператори ASK і TELL, які дозволяють здійснювати запити та оновлення знань, як показано на рис. 1.2. За потреби механізм виведення HKnowLang може також створювати й

повертати модель самопристосувальної поведінки, яка складається з ланцюжка дій для реалізації в середовищі або системі.

1.3. Дослідження підходу керованого моделлю

MDSE (Model-Driven Software Engineering) - це сучасний підхід до розробки програмного забезпечення, який ставить у центр уваги створення і використання моделей системи на всіх етапах життєвого циклу розробки. Цей підхід дозволяє автоматизувати багато рутинних завдань і підвищити якість кінцевого продукту.

Основні принципи MDSE:

- Моделі як першоджерело: Модель системи створюється на ранніх етапах розробки і служить єдиним джерелом істини для всіх подальших робіт.

- Автоматизація: Багато процесів, таких як генерація коду, створення документації та тестування, автоматизуються на основі моделей.

- Високий рівень абстракції: Моделі дозволяють працювати на більш високому рівні абстракції, що спрощує розуміння системи і зменшує кількість помилок.

- Повторне використання: Моделі можуть бути повторно використані для створення різних варіантів системи або для розробки нових систем на їх основі.

Етапи розробки за допомогою MDSE:

1. Створення концептуальної моделі: Визначаються основні поняття, зв'язки і правила домену.

2. Розробка моделі системи: На основі концептуальної моделі створюється детальна модель системи, яка описує її структуру, поведінку і взаємодію з зовнішнім середовищем.

3. Генерація коду: За допомогою спеціальних інструментів (генераторів коду) модель трансформується в виконуваний код.

4 Тестування і верифікація: Проводиться перевірка відповідності отриманого коду моделі і виявлення можливих помилок.

5. Розгортання і підтримка: Завершений продукт розгортається в виробничому середовищі і підтримується протягом усього життєвого циклу.

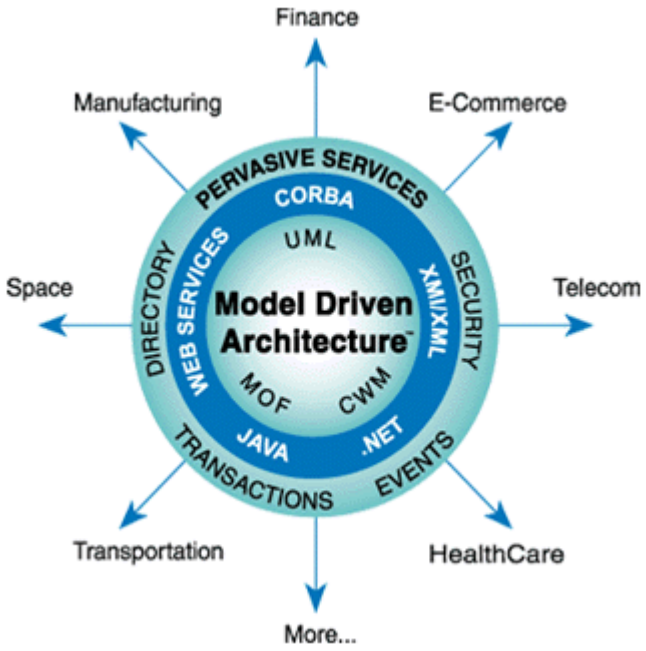

Рис. 1.3. Підхід Model-Driven Software Engineering

MDA пропонує рекомендації щодо структурування специфікацій програмного забезпечення, які виражаються у вигляді моделей.

MDA відокремлює бізнес-логіку та логіку додатків від технологічної платформи. Платформонезалежні моделі бізнес-функціональності та поведінки додатка або інтегрованої системи, створені за допомогою UML та інших стандартів моделювання OMG, можуть бути реалізовані через MDA на практично будь-якій платформі — відкритій чи власній, включаючи Web Services, .NET, CORBA, J2EE та інші. Ці платформонезалежні моделі документують бізнес-функціональність та поведінку додатка окремо від технологічного коду, що його реалізує, тим самим ізолюючи основну частину програми від змін технологій. Це дозволяє забезпечити інтероперабельність як всередині платформи, так і між різними платформами. Оскільки бізнесові та технічні аспекти додатка чи інтегрованої системи не залежать один від

одного, вони можуть розвиватися власними темпами: бізнес-логіка — у відповідь на бізнес-потреби, а технології — використовуючи новітні досягнення.

Переваги MDSE:

- Підвищення продуктивності: Автоматизація багатьох процесів дозволяє скоротити час розробки.

- Покращення якості: Завдяки використанню моделей знижується кількість помилок і підвищується надійність системи.

- Збільшення повторного використання: Моделі можуть бути використані для створення різних варіантів системи.

- Покращення комунікації: Моделі є універсальною мовою для спілкування між різними учасниками проекту.

Інструментами для MDSE є наступні:

- UML (Unified Modeling Language): Стандартна мова для візуального моделювання об'єктно-орієнтованих систем.

- SysML (Systems Modeling Language): Розширення UML, спеціально розроблене для моделювання складних систем.

- MDA (Model-Driven Architecture): Стандартний набір специфікацій, що описує процес розробки програмного забезпечення на основі моделей.

- Різноманітні генератори коду: Перетворюють моделі в код на різних мовах програмування.

MDSE є потужним інструментом для підвищення ефективності і якості розробки програмного забезпечення. Завдяки використанню моделей, автоматизації і високому рівню абстракції, MDSE дозволяє створювати більш складні і надійні системи.

Метою даної роботи є додавання візуального рівня (VL) як функції системи, призначеної для представлення знань. Основними причинами невдачі проектів системної інженерії є складність, спілкування та відсутність розуміння, як зазначено у визначенні проблеми. MDSE вирішує ці проблеми системної інженерії, як показали дослідження [16, 18]. Значення

використання підходу MDSE для вирішення цих проблем полягає в наступному:

• Складність: розробка VL вимагає моделювання кількох окремих візуальних елементів. Цей VL є модулем більшої системи, призначеної для розробки програмного забезпечення за допомогою HKnowLang. Управління кодом цієї системи є складним завданням. Використання моделей для представлення завдань, пов’язаних із розробкою системи, дозволяє впоратися зі складністю завдання.

• Зв'язок: використання HKnowLang для реалізації транспортних засобів SAE L4 є триваючим дослідженням. У розробці системи беруть участь численні зацікавлені сторони, які зараз беруть участь у проекті, а також майбутні дослідники. Отже, потрібен ефективний спосіб комунікації та документування. Підхід MDSE задовольняє цю вимогу.

• Відсутність розуміння: поширеною причиною провалу проекту є нерозуміння рішення. Проблеми можуть виникнути на будь-якому етапі життєвого циклу проекту. Впровадження MDSE забезпечує єдину методологію на кожному етапі життєвого циклу проекту та забезпечує можливість відстеження.

Системна інженерія, керована моделями, спрямована на стандартизований і цілісний підхід до системної інженерії. INCOSE, Міжнародна рада з системної інженерії, визначає MDSE як «формалізоване застосування моделювання для підтримки системних вимог, проектування, аналізу, верифікації та валідації діяльності, що починається на етапі концептуального проектування та продовжується протягом розробки та наступних фаз життєвого циклу» [17].

Життєвий цикл проекту включає різні артефакти, включаючи, але не обмежуючись, вимоги, варіанти використання, архітектуру системи, дизайн системи та тестові випадки. Традиційний підхід системної інженерії на основі документів розглядає кожен артефакт окремо та зберігає інформацію переважно в текстових документах [19, 20]. Це ускладнює передачу

інформації між доменами та вимагає ручного аналізу, перегляду та перевірки [21]. MDSE забезпечує цілісний підхід до системної інженерії, використовуючи взаємопов’язані моделі для зберігання та зображення артефактів, які представляють аспект системи [22]. Основна ідея полягає в тому, що кожен артефакт представляє погляд на модель системи, а модель системи охоплює всі аспекти проекту. Зміни в будь-якому представленні моделі системи викликають відповідні зміни в інших відповідних представленнях моделі системи, таким чином забезпечуючи відстежуваність. Наприклад, коли вимога змінюється на будь-якій стадії проекту, команда управління тестуванням повідомляється про зміну свого погляду на модель системи.

Підхід MDSE спирається на наступні три концепції [23]:

• Метод: метод генерації артефактів MDSE;

• Мова: мова, що використовується для вираження артефактів MDSE;

• Інструменти: інструменти, що використовуються для побудови артефактів MDSE.

Дані концепції будуть детальніше описані в наступному розділі.

Висновки до розділу

У результаті дослідження предметної області моделювання автономних програмних систем було визначено основні проблеми, з якими стикаються розробники при створенні складних програмних систем. Зокрема, акцентовано на важливості відокремлення бізнес-логіки від технологічної платформи для забезпечення гнучкості та незалежності програмних рішень від постійно змінюваних технологій. Проаналізовано багаторівневу модель специфікації знань, яка дозволяє більш ефективно структурувати інформацію та забезпечувати її інтероперабельність між різними платформами.

Дослідження підходу керованого моделлю (Model Driven Architecture, MDA) продемонструвало його значні переваги в контексті проєктування та розробки автономних програмних систем. MDA дозволяє створювати платформонезалежні моделі, які відображають функціональність та поведінку системи на рівні бізнес-логіки, ізолюючи їх від технологічних аспектів. Це забезпечує можливість еволюції як бізнесових, так і технічних складових системи незалежно одна від одної, що сприяє підвищенню адаптивності та гнучкості програмних рішень.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТА ІНТЕРПРЕТАЦІЯ ПІДХОДУ КЕРОВАНОГО МОДЕЛЛЮ ДЛЯ РОЗРОБКИ ВІЗУАЛЬНОГО РІВНЯ

2.1. Опис методу та мови генерації артефактів Model Driven Architecture

Наразі не існує стандартного методу для створення графічного представлення DSL і генерації коду DSL. Ми використовуємо адаптацію методології SYSMOD [24] для цілей даного проекту. SYSMOD — це методологія MDSE. Процеси SYSMOD, що стосуються цього дослідження, як зазначено нижче:

1. Вимоги до моделі: цей процес передбачає визначення вимог, які визначають особливості системи. Як DSL, HKnowLang має власні унікальні конструкції. Початкова мета — зрозуміти різні конструкції специфікації в HKnowLang і визначити вимоги до VL.

2. Варіанти використання моделі: Вичерпний список варіантів використання визначено для покриття вимог і функцій, які надаватиме VL. Варіанти використання будуть змодельовані та реалізовані у вибраному інструменті моделювання. Як правило, моделювання варіанту використання передбачає визначення таких функцій:

• асоційовані актори системи.

• тригер для ініціювання варіанту використання.

• результат варіанту використання.

• опис варіанту використання.

• передумови та післяумови системи.

• нефункціональні вимоги, що стосуються варіанту використання.

• простежувані шляхи до відповідних функціональних вимог.

• випадок використання.

3. Візуальний рівень моделі: цей процес стосується реалізації варіантів використання. Моделювання візуального рівня включає два основні аспекти:

• моделювання графічного інтерфейсу користувача для HKnowLang.

• моделювання генератора коду для перетворення графічних моделей у код HKnowLang.

4. Перевірка візуального рівня: VL остаточно перевірено з використанням проекту eMobility [9] як тестового сценарію. Проект eMobility графічно моделюється в розробленому VL, а відповідний код HKnowLang генерується з VL. Цей згенерований код перевіряється в існуючому текстовому редакторі HKnowLang. Крім того, згенерований код перевіряється вручну шляхом порівняння з еталонним кодом проекту eMobility.

Мова моделювання систем (SysML) [25] є мовою моделювання загального призначення для MDSE, яка широко використовується в інженерних програмах [24]. SysML є розширенням підмножини уніфікованої мови моделювання (UML) [26], як показано на рисунку 2.1.

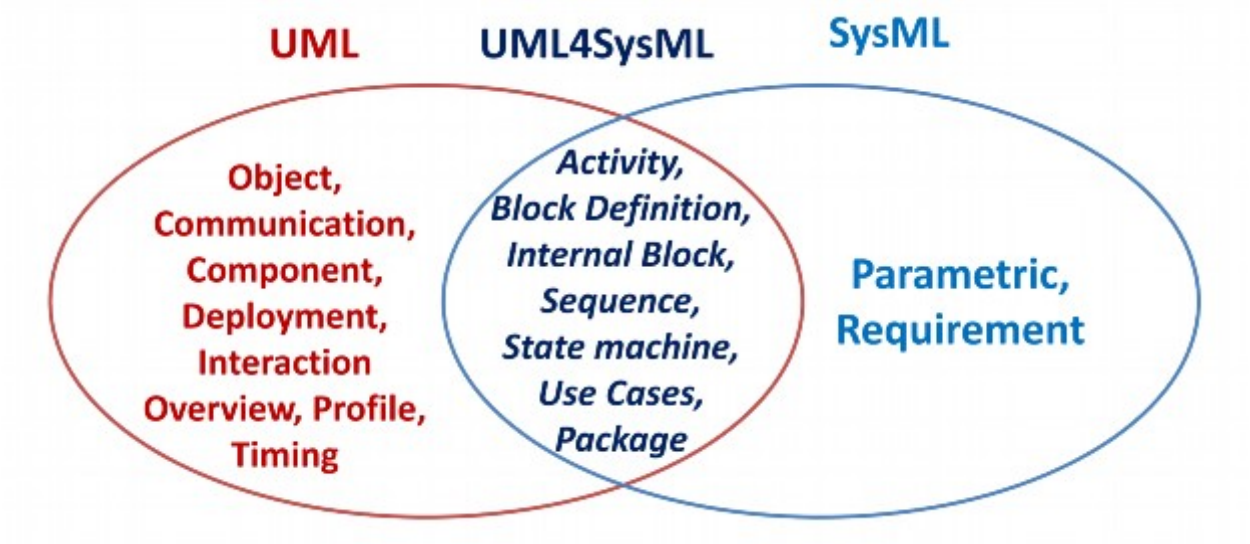

Рис. 2.1. Зв’язки між SysML і UML

Дев'ять стандартних типів діаграм використовуються для представлення різних поглядів на модель системи, як показано на рисунку 2.2. Класифікація стандартних діаграм така:

• Діаграма вимог [req] – вказання вимог та зв'язків.

• Структурні діаграми - виражають архітектуру системи:

– Block Definition Diagram [bdd] – виражають статичні структури системи.

– Internal Block Diagram [ibd] – виражає інтерфейси та внутрішні зв’язки між компонентами системи.

- Parametric Diagram [par] - спеціальний тип внутрішньої блок-схеми, який використовується для вираження математичних зв'язків у системі.

- Package Diagram [pkg] - виражає ієрархію моделі та структуру системи.

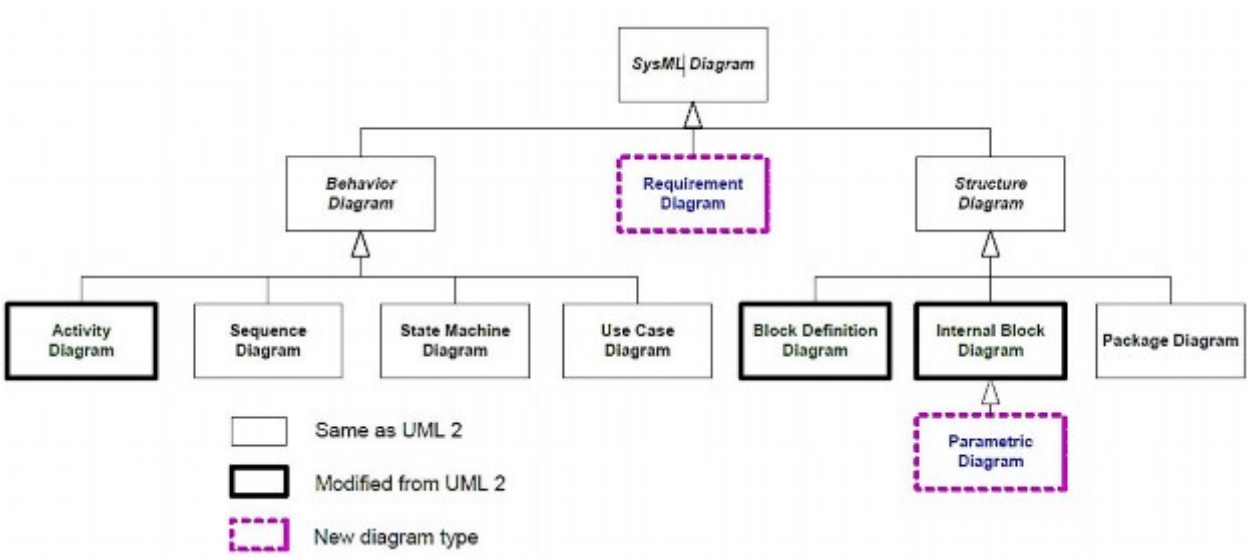

Рис. 2.2. Діаграми SysML

• Діаграми поведінки - виражають поведінку системного об'єкта:

– Use Case Diagram [uc] - виражає поведінку системи з точки зору взаємодії з акторами.

– Activity Diagram [act] – виражає потік даних і потік керування через систему.

– Sequence Diagram [seq] – виражає точні взаємодії між цікавою системою та акторами в її оточенні.

– State Machine Diagram [stm] - виражає стан системи та умови переходу між станами

Існуючий текстовий редактор HKnowLang розроблено на Java [15]. VL призначений для функціонування як автономна програма, яку можна інтегрувати з існуючим текстовим редактором. Тому Java-додаток потрібно створити на основі системних моделей. Однак, як буде пояснено в наступному пункті, існуючі інструменти моделювання не повністю підтримують створення автономних програм графічного інтерфейсу користувача. Отже, моделювання графічного інтерфейсу користувача виконується текстовою розробкою з використанням Java.

2.2. Дослідження та аналіз інструментів моделювання

Засоби моделювання дозволяють реалізувати методологію MDSE за допомогою мови моделювання. Доступні різноманітні ліцензовані інструменти, а також інструменти з відкритим кодом, які підтримують SysML. Інструменти відрізняються зовнішнім виглядом, функціями та підтримкою створення коду. Вибраний інструмент повинен підтримувати впровадження процесів методу моделювання, визначених для цього проекту. У цьому розділі ми досліджуємо та вибираємо інструмент, який підходить для визначених процесів. IBM Rational Rhapsody вважається придатним інструментом для цілей моделювання вимог і випадків використання. Проте вибір відповідного інструменту для моделювання VL потребує подальших досліджень.

Компанія BAESIS Automotive BV сформулювала певні вимоги до застосування візуального рівня, які впливають на вибір інструменту моделювання. Вимоги такі:

1. Візуальний рівень має надавати можливості графічного моделювання в HKnowLang.

2. Візуальний рівень автоматично генерує код HKnowLang з моделей, розроблених у цьому рівні.

3. Візуальний рівень має існувати як окрема програма.

4. Візуальний рівень має бути здатний інтегруватися з існуючим текстовим редактором HKnowLang, який є програмою Java.

5. Візуальний рівень розробляється з використанням безкоштовних інструментів або інструментів з відкритим кодом.

Відповідно до літературного огляду, проведеного [27, 28, 29], існує вибір інструментів, які дозволяють керовану моделлю розробку DSL. Тут ми коротко представляємо деякі з відомих інструментів, визначених у дослідницьких роботах [27, 28, 29].

Eclipse Modeling Framework, наданий Eclipse, є основною структурою моделювання та генерації коду з використанням моделей, визначених у Ecore [28]. Деякі інструменти моделювання, такі як Sirius і Graphiti, побудовані на Eclipse Modeling Framework.

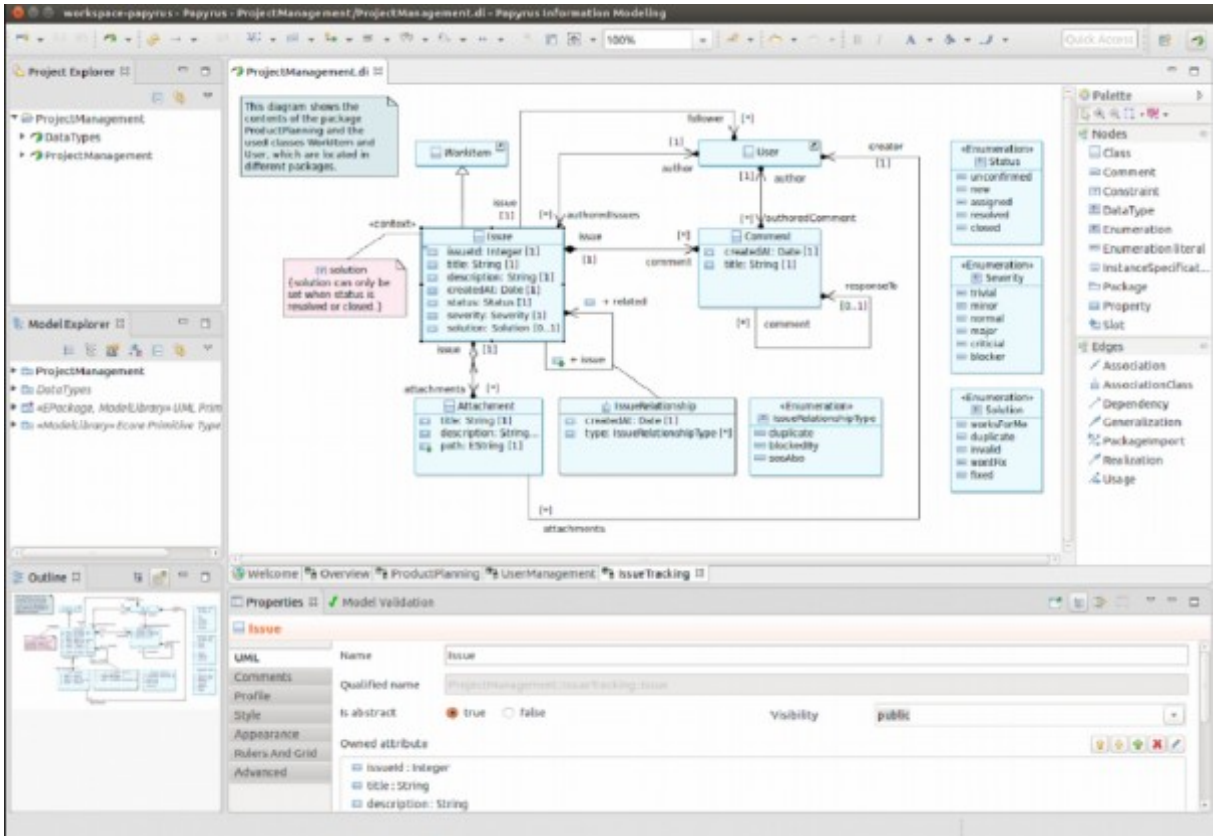

Рис. 2.3. Модель виконана засобами Eclipse Modeling Framework

Проєкт EMF (Eclipse Modeling Framework) — це каркас для моделювання та засіб автоматичної генерації коду, який використовується

для створення інструментів та інших програм на основі структурованої моделі даних. Зі специфікації моделі, описаної за допомогою XMI (XML Metadata Interchange), EMF надає інструменти та підтримку під час виконання для створення набору класів Java, що відповідають моделі, а також набір адаптерних класів, які дозволяють переглядати та редагувати модель за допомогою команд, і базовий редактор.

EMF (ядро) є загальним стандартом для моделей даних, на якому базуються численні технології та фреймворки. До них належать серверні рішення, фреймворки для збереження даних, інтерфейси користувача (UI), а також підтримка перетворень. Проєкт моделювання пропонує огляд технологій EMF.

Основні компоненти EMF:

- EMF (ядро) — головний фреймворк EMF включає метамодель (Ecore) для опису моделей і підтримку під час виконання, що забезпечує сповіщення про зміни, підтримку збереження з використанням стандартної серіалізації XMI, а також високоефективний відбивний API для маніпуляцій об'єктами EMF.

- EMF.Edit — фреймворк EMF.Edit включає універсальні класи, придатні для створення редакторів моделей EMF. Він забезпечує:

- Класи для забезпечення вмісту та позначок, підтримку джерел властивостей та інші допоміжні класи, що дозволяють відображати моделі EMF за допомогою стандартних інструментів (JFace) та панелей властивостей.

- Фреймворк команд, що включає набір універсальних класів команд для створення редакторів із повністю автоматичною підтримкою скасування та повторення дій.

- EMF.Codegen — засіб генерації коду EMF здатний генерувати все необхідне для створення повного редактора моделі EMF. Він містить графічний інтерфейс користувача для налаштування параметрів генерації та

виклику генераторів. Засіб генерації використовує компонент JDT (Java Development Tooling) Eclipse.

EMF підтримує три рівні генерації коду:

- Модель — генерує інтерфейси та класи реалізації для всіх класів моделі, а також класи фабрики та пакета (метадані).

- Адаптери — генерує класи реалізації (ItemProviders), які адаптують класи моделі для редагування та відображення.

- Редактор — створює структурований редактор, що відповідає рекомендованому стилю для редакторів моделей EMF у середовищі Eclipse та слугує початковою точкою для подальшого налаштування.

Sirius базується на Eclipse Modeling Framework і дозволяє користувачам створювати графічні інструменти моделювання. Інструмент дозволяє легко визначати графічні представлення, такі як діаграми, таблиці або дерева, з багатою взаємодією з користувачем, приховуючи складність [29]. Однак Sirius не підтримує генерацію коду.

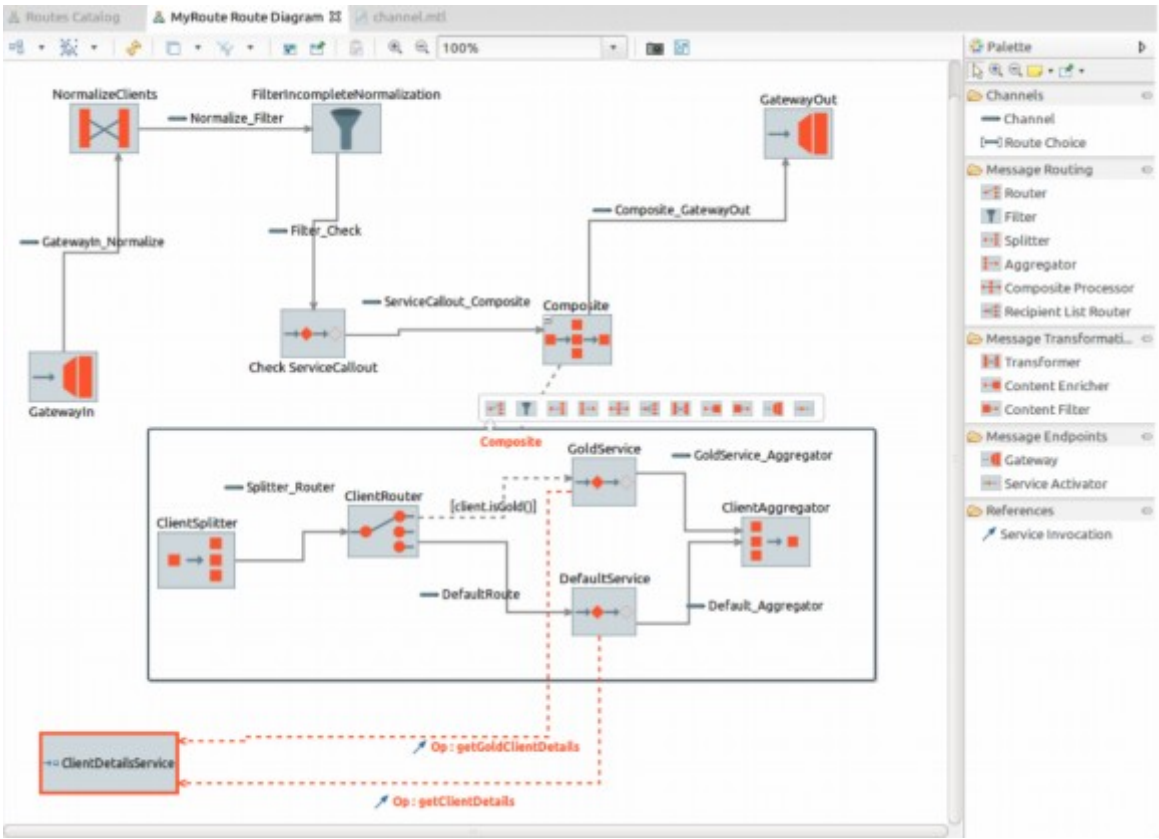

Рис. 2.4. Модель засобами Sirius від Eclipse Modeling Framework

Eclipse Sirius надає можливість створювати моделювальне середовище у вигляді графічних, табличних або деревоподібних редакторів з правилами валідації та діями, використовуючи декларативні описи.

Усі характеристики форм та їх поведінку можна легко налаштувати з мінімальними технічними знаннями. Цей опис динамічно інтерпретується для візуалізації робочого середовища безпосередньо в Eclipse IDE. Генерація коду не потрібна, і розробник середовища моделювання може отримувати миттєвий зворотний зв'язок під час налаштування опису.

Після завершення, моделювальне середовище може бути розгорнуте як стандартний плагін Eclipse.

Graphiti — ще один інструмент, побудований на Eclipse Modeling Framework [29]. Graphiti надає добре структурований Java API для створення графічних інструментів. Він використовує Eclipse Draw2D для створення складних редакторів. Для специфікації метамоделі використовуються еко-діаграми [27]. Інструмент за замовчуванням забезпечує стандартизований вигляд і відчуття для розроблених візуальних редакторів.

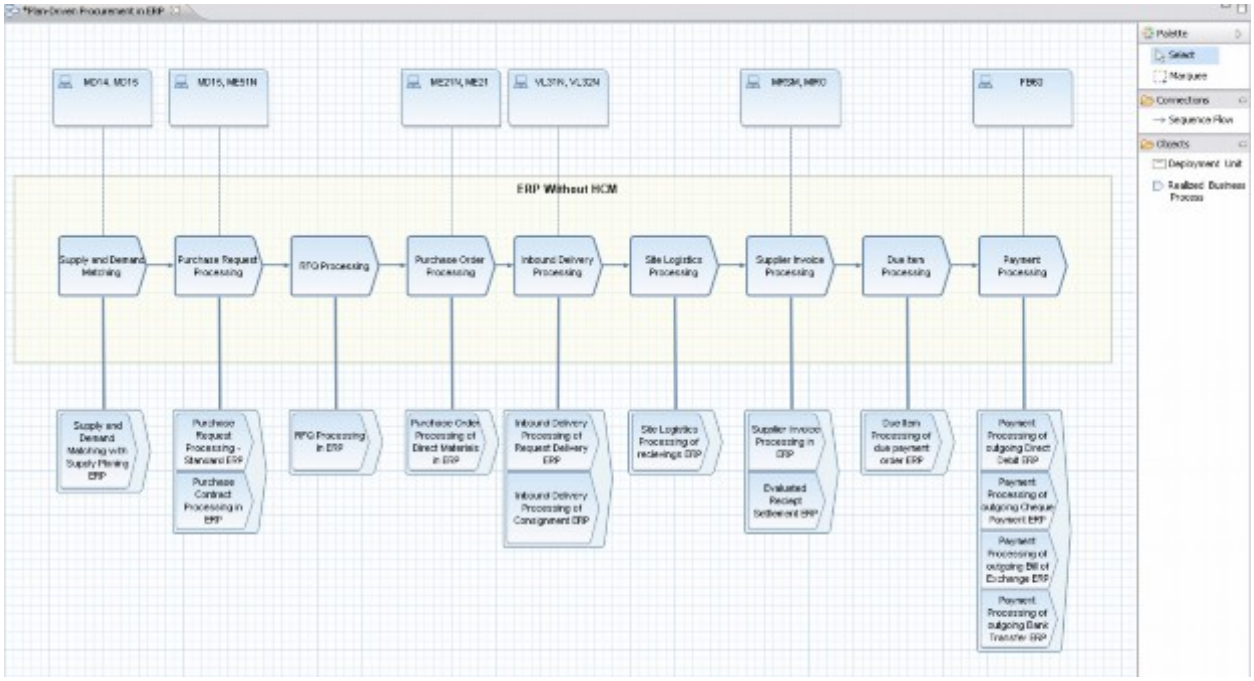

Рис. 2.5. Модель засобами Graphiti (Eclipse Modeling Framework)

MetaEdit+ — це повністю інтегрований інструмент середовища, розроблений для побудови та використання рішень для предметно-спеціального моделювання. Інструмент надає стандартний набір функцій інструменту CASE, включаючи графічні редактори та керування проектними даними. MetaEdit+ базується на GOPPRR, запатентованій мові метамоделювання [29].

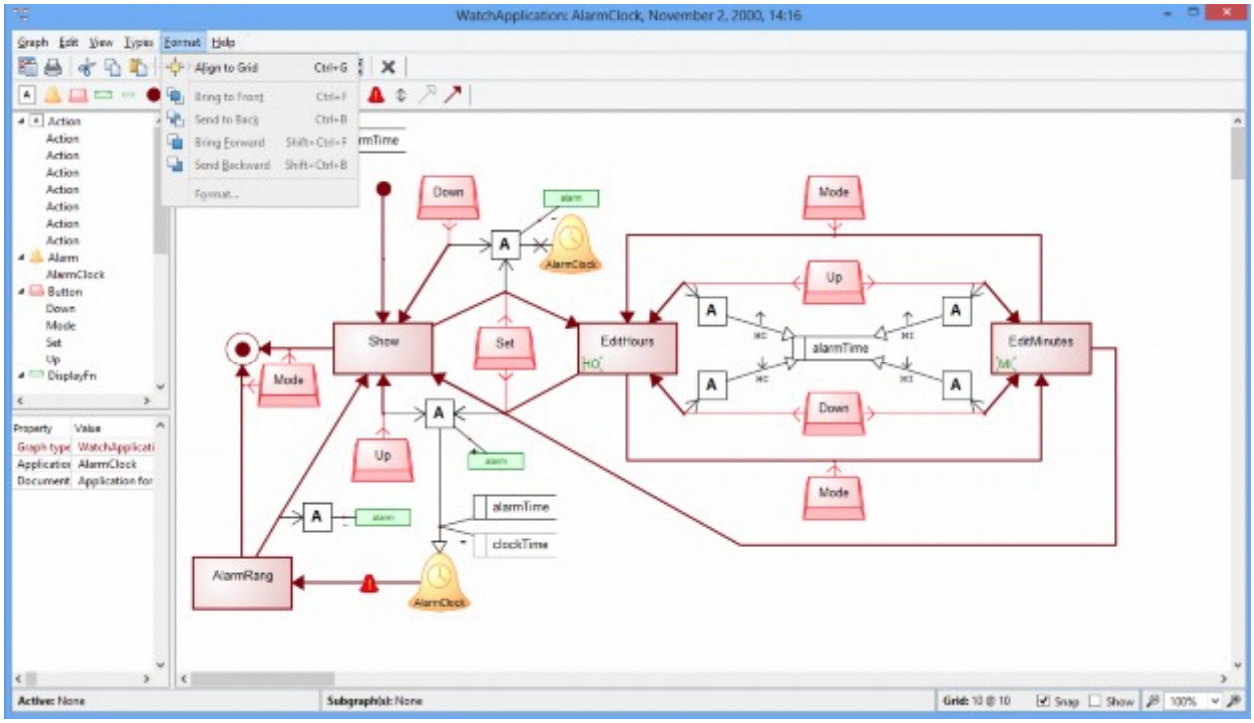

Рис. 2.6. Середовище моделювання MetaEdit+

MetaEdit+ - це середовище, яке дозволяє створювати інструменти моделювання та генератори, що відповідають конкретним предметним областям, без написання жодної рядки коду. Можливість визначати інструменти моделювання та генератори є важливою, оскільки вона дозволяє підвищити рівень абстракції проєктних робіт від коду до концепцій предметної області, а підвищення рівня абстракції призводить до неминущого зростання продуктивності, як показує досвід останніх років.

У моделюванні конкретної предметної області та MetaEdit+ експерт визначає мову конкретної предметної області як метамодель, що містить

концепції та правила предметної області, і визначає відображення з неї в код за допомогою генератора коду для конкретної предметної області.

Для реалізації методу MetaEdit+ пропонує мову метамоделювання та набір інструментів для визначення понять методу, їх властивостей, пов'язаних правил, символів, звітів про перевірку та генераторів. Після того, як експерт визначить метод моделювання або навіть частковий прототип, решта команди може почати використовувати його в MetaEdit+ для створення моделей за допомогою мови моделювання, а необхідний код автоматично генерується з цих моделей.

На основі метамоделі MetaEdit+ автоматично забезпечує функціональність інструментів CASE (Computer-Aided Software Engineering - Засоби автоматизованої підтримки розробки програмного забезпечення): редактори діаграм, браузери, генератори, підтримку мультикористувацького режиму/проєкту/платформі тощо.

Демонстрація MetaEdit+ зосередиться на тому, щоб показати, як створюються мови конкретних предметних областей та генератори; з повними прикладами методів конкретних предметних областей та пов'язаних генераторів коду.

Microsoft Visual Studio пропонує DSL Tools, інтегрований набір інструментів, який підтримує визначення DSL і генерацію коду [28].

JetBrains Meta Programming System — це програмне забезпечення з відкритим вихідним кодом, розроблене JetBrains і не має граматики та парсера. Він підтримує змішані нотації, включаючи текстові, символьні, табличні, графічні та низку функцій композиції мови на основі BaseLanguage, яка є мета-моделлю MPS. BaseLanguage розширено для створення нових мов [28].

Таблиця 2.1 показує відповідні інструменти, доступні для розробки DSL на основі моделі, згідно з дослідженнями [27, 28, 29]. У таблиці оцінюється життєздатність кожного інструменту щодо вимог до візуального рівня.

Таблиця 2.1.

Оцінка інструментів моделювання

Req / Tool	Visual modelling	Code generator	Standalone GUI	Plugin support	Free/Open source
Eclipse Modelling Framework	Yes	Yes	No	Possible	Yes
MetaEdit+	Yes	Yes	No	Possible	No
Microsoft DSL Tools	Yes	Yes	No	Possible	Yes
JetBrains MPS	Limited	Yes	No	No	Yes
Sirius	Yes	No	No	Possible	Yes
Graphiti	Yes	Possible	No	Possible	Yes
IBM Rational Rhapsody	Yes	Yes	No	Limited	Academic free

Критичний аналіз показує, що жоден із існуючих інструментів MDSE не підтримує розробку автономної програми графічного інтерфейсу користувача (GUI) з використанням моделей. Наприклад, інтерфейс, змодельований за допомогою Graphiti, працює в інтегрованому середовищі розробки (IDE) Eclipse. Ми пропонуємо вирішити цю проблему, розділивши завдання моделювання, як зазначено нижче:

• Створення метамоделі: створення метамоделей для HKnowLang виконується в інструменті моделювання MDSE. Для метамоделювання обрано IBM Rational Rhapsody.

• Створення графічного інтерфейсу користувача: розробка графічного інтерфейсу користувача виконується за допомогою традиційних бібліотек графічного інтерфейсу користувача Java, таких як Java Swing в Apache NetBeans IDE.

Потім мета-модель і GUI об’єднуються для створення VL.

2.3. Вимоги до візуального рівня моделювання систем

HKnowLang було розроблено як текстову мову. VL надасть можливість візуального моделювання за допомогою HKnowLang. Реалізація VL вимагає

визначення вимог для графічного представлення конструкцій онтології домену HKnowLang. Дослідження HKnowLang тривають і зосереджені на використанні в галузі автономних транспортних засобів. Тому в майбутньому вимоги можуть бути змінені. Наприклад, може знадобитися змінити мовні конструкції на основі результатів дослідження та відгуків користувачів. Цей розділ спрямований на вирішення цих проблем шляхом створення структури вимог і виявлення функціональних вимог до VL. Вимоги будуть використані для розробки VL для підтвердження концепції, як описано в наступних розділах. Як інструмент для моделювання вимог використовується IBM Rational Rhapsody. Для позначення пріоритету вимог ми використовуємо трирівневу класифікацію кольорового коду для позначення вимог високого, середнього та низького пріоритету, як показано на рисунку 2.7. Вимоги високого пріоритету будуть реалізовані в цій роботі, щоб продемонструвати візуальний рівень підтвердження концепції для моделювання.

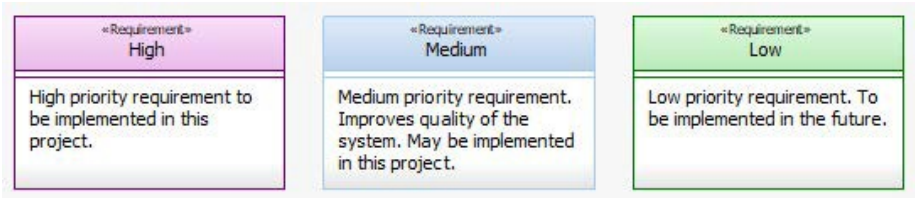

Рис. 2.7. Ілюстрація пріоритету вимог

2.3.1. Бізнес-вимоги

Бізнес-вимоги описують ринок і місію бізнесу. Відповідні бізнес-вимоги згідно з обговореннями представлені на рисунку 2.8.

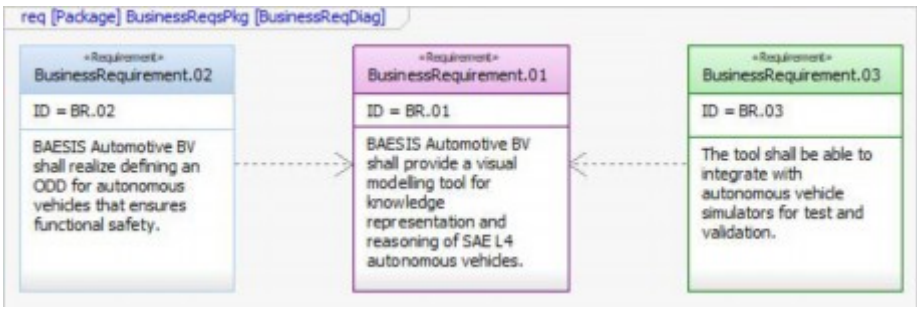

Рис. 2.8. Діаграма бізнес-вимог

2.3.2. Пакети вимог

Бізнес-вимоги формують основу для визначення функціональних і нефункціональних вимог. Вимоги до HKnowLang класифіковані за пакетами, щоб забезпечити структуру та відстежуваність. Пакетна діаграма встановлює основу для управління вимогами.

Фокус цього дослідження обмежений реалізацією BusinessRequirement.01. Модель специфікації знань HKnowLang, яка вимагає візуального моделювання, була представлена в першому розділі. Для кожної конструкції моделі специфікації, яка вимагає візуального моделювання, створюється унікальний пакет вимог. Діаграма пакета вимог зображена на рисунку 2.9.

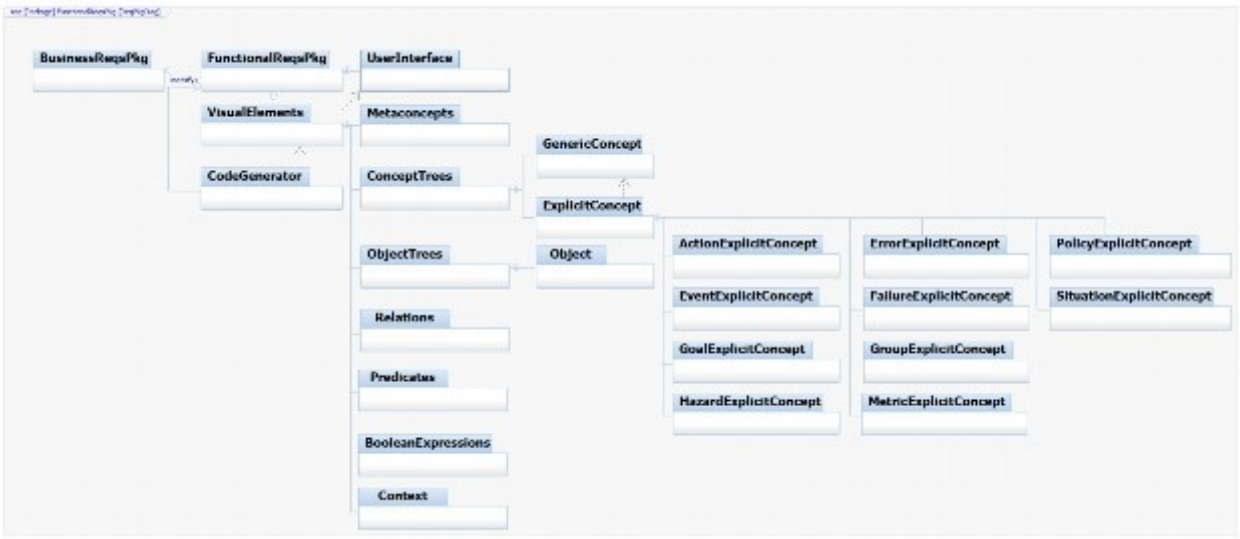

Рис. 2.10. Діаграма пакета вимог

2.4. Реалізація діаграм вимог

Ми виявляємо вимоги для кожного пакета та використовуємо діаграми вимог, щоб візуалізувати залежності між вимогами. На рисунку 2.11 зображено функціональні вимоги. У таблиці 2.2 також представлені функціональні вимоги у формі таблиці.

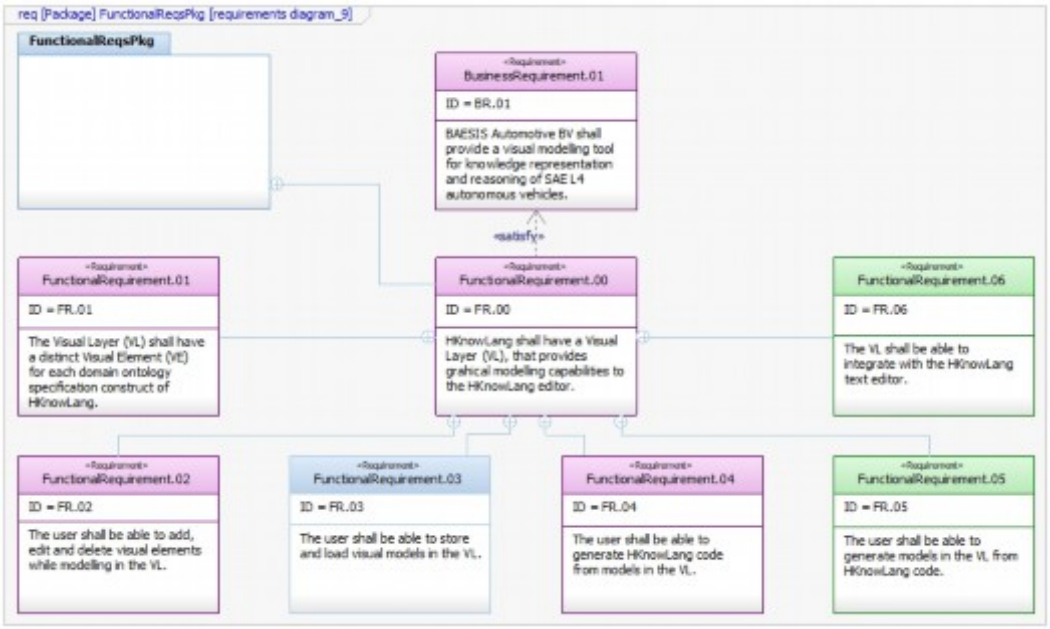

Рис. 2.10. Діаграма функціональних вимог

Таблиця 3.1.

Таблиця функціональних вимог

Назва вимоги	**Специфікація**
Функціональна вимога.00	HKnowLang має мати візуальний рівень (VL), який надає редактору HKnowLang можливості графічного моделювання.
Функціональна вимога.01	Візуальний рівень (VL) повинен мати окремий візуальний елемент (VE) для кожної конструкції специфікації онтології домену HKnowLang.
Функціональна вимога.02	Користувач повинен мати можливість додавати, редагувати та видаляти візуальні елементи під час моделювання у VL.
Функціональна вимога.03	Користувач повинен мати можливість зберігати та завантажувати візуальні моделі в VL.
Функціональна вимога.04	Користувач повинен мати можливість генерувати код HKnowLang з моделей у VL.
Функціональна вимога.05	Користувач повинен мати можливість генерувати моделі у VL з коду HKnowLang.
Функціональна вимога.06	VL повинен мати можливість інтегруватися з текстовим редактором HKnowLang.

Функціональні вимоги далі розкладаються на пакети, як показано на рисунку 2.11.

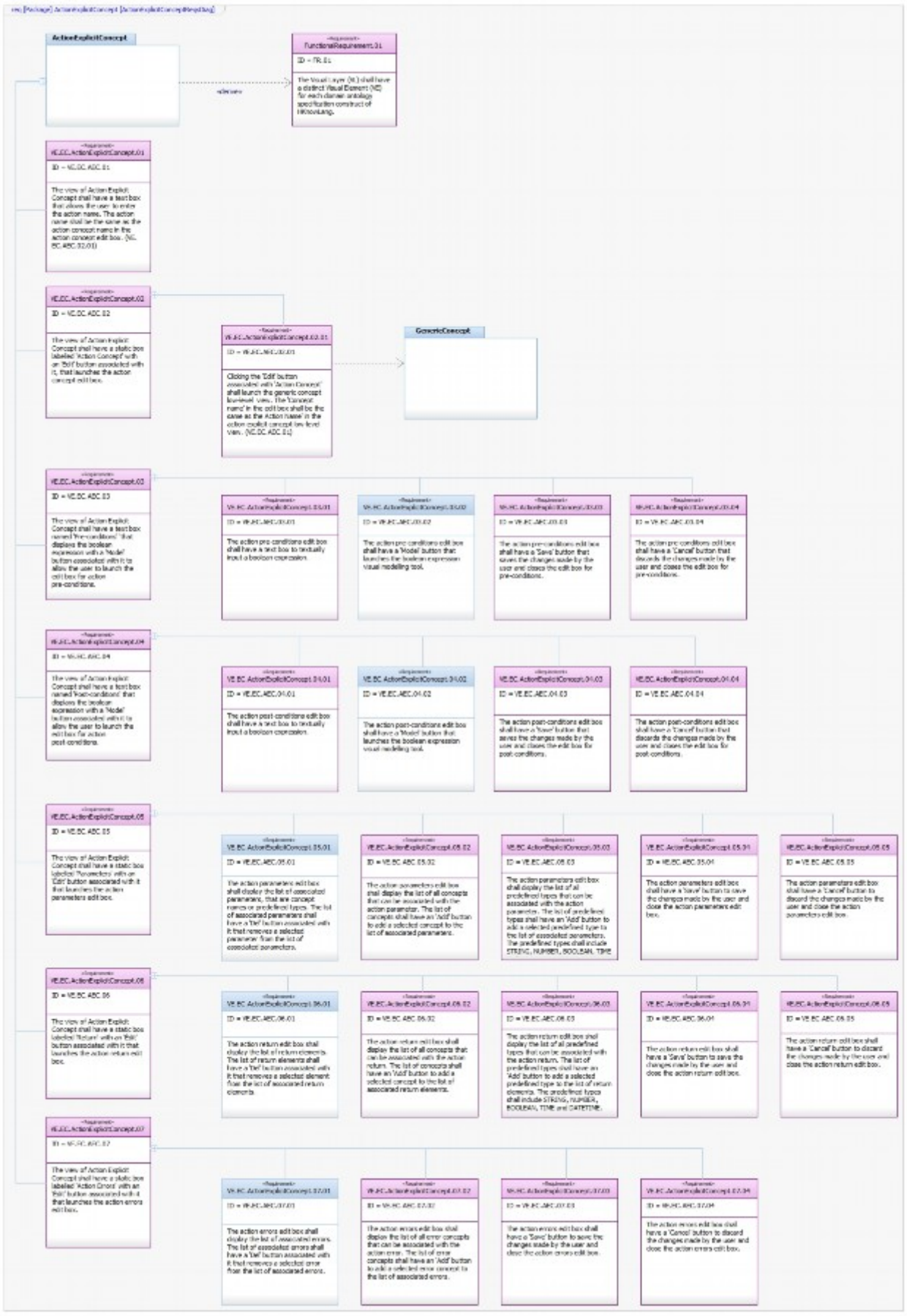

Рис. 2.11. Діаграма явних вимог концепції

Кожен пакет вимог містить діаграму вимог, що представляє зв’язок між вимогами, що містяться в пакеті. Як приклад, рисунок 2.11 представляє діаграму вимог до пакета Action Explicit Concept. Нижче представлено декілька діаграми вимог.

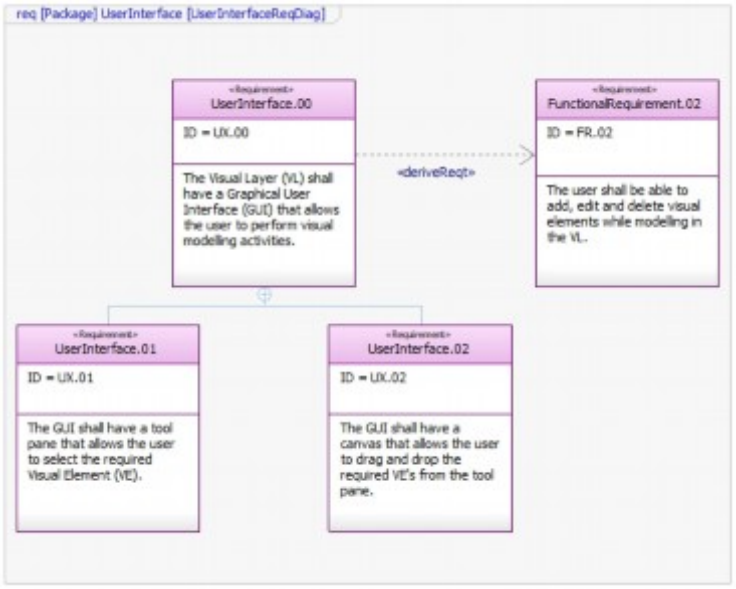

Рис. 2.12. Діаграма вимог до інтерфейсу користувача

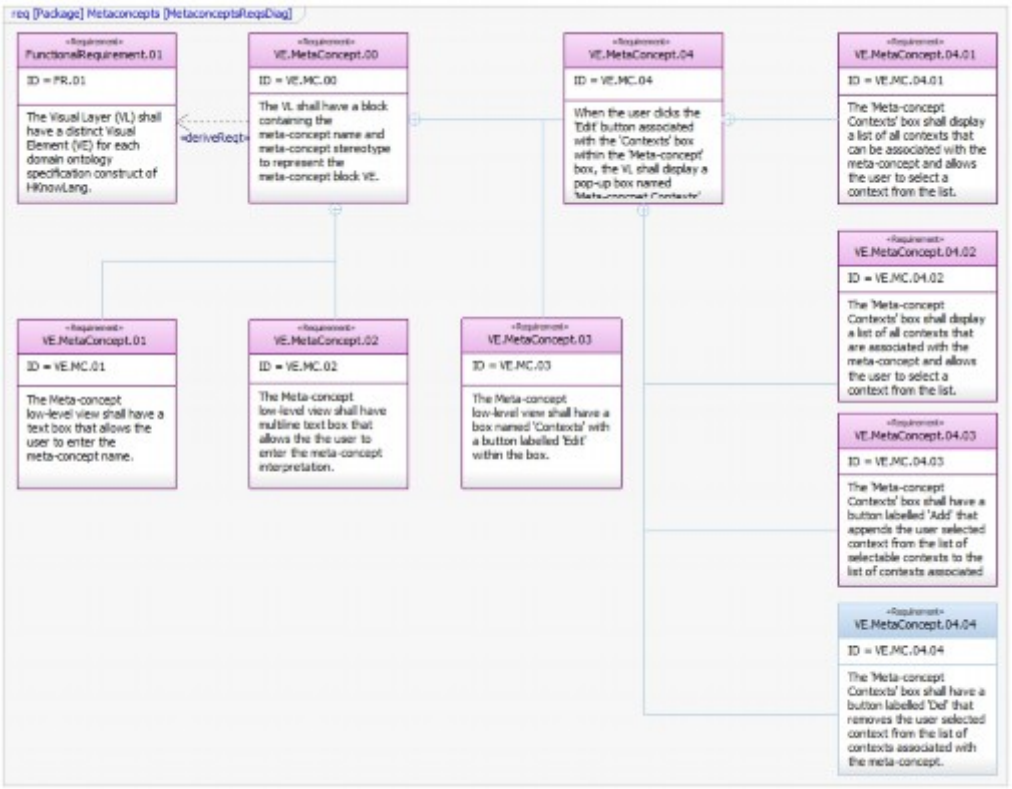

Рис. 2.13. Діаграма вимог до мета концепції

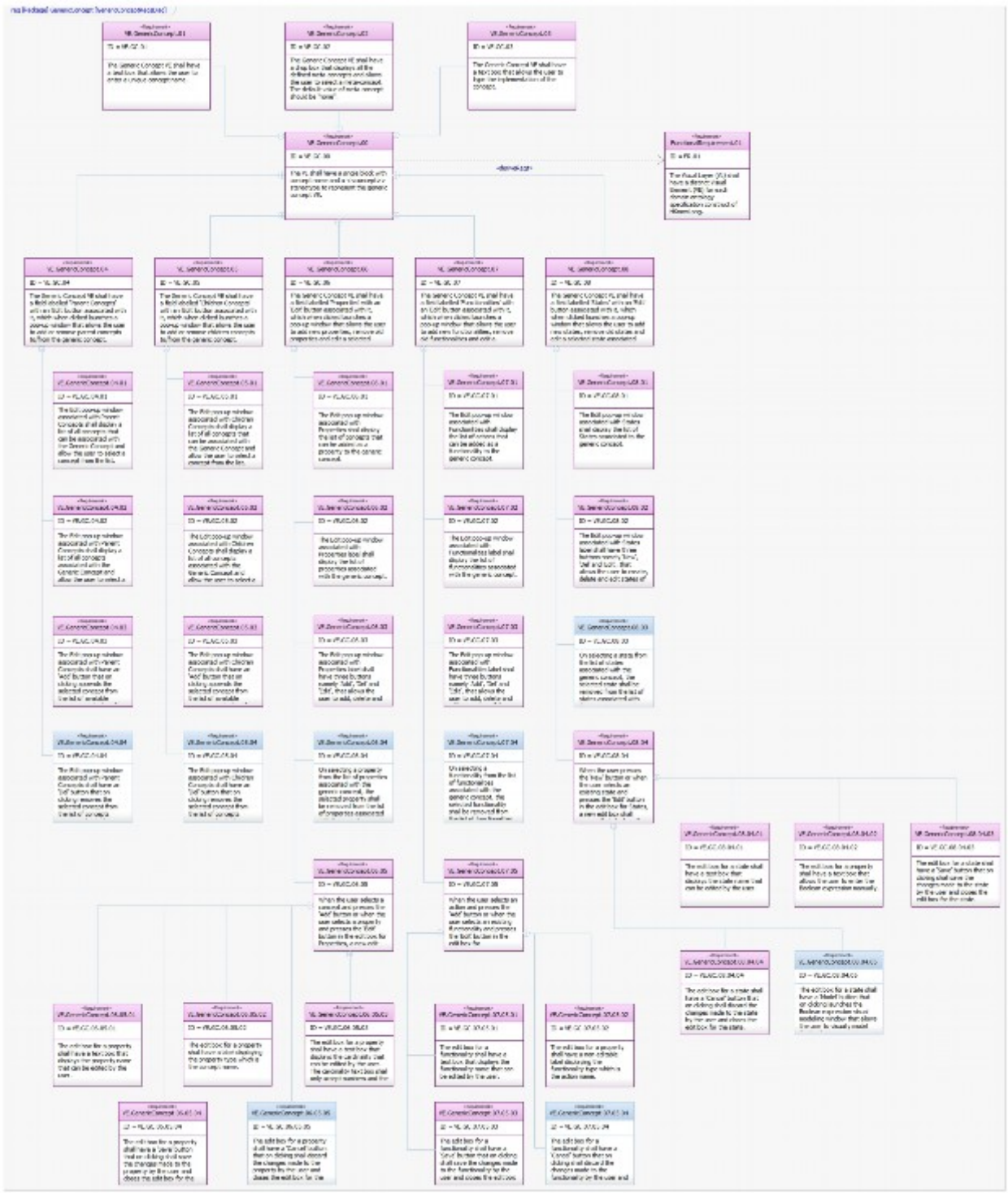

Рис. 2.14. Діаграма вимог загальної концепції

Висновки до розділу

У цьому розділі запропоновано застосувати підхід системної інженерії, керованої моделями (Model-Driven Systems Engineering, MDSE), для розробки візуального рівня (VL) моделювання автономних систем. Спочатку

окреслено основні концепції та переваги підходу MDSE. Далі проведено аналіз доцільності впровадження MDSE у процес розробки VL.

Також було детально розглянуто бізнес-вимоги, представлено схему пакету вимог, яка відображає організаційну структуру вимог. Розроблено діаграму вимог, що демонструє процес ідентифікації функціональних вимог системи.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МЕТОДОЛОГІЙ ТА ПІДХОДІВ ВІЗУАЛЬНОГО РІВНЯ МОДЕЛЮВАННЯ АВТОНОМНИХ ПРОГРАМНИХ СИСТЕМ

3.1. Процес моделювання варіантів використання

SYSMOD (System Modification) – це фундаментальний елемент операційної системи IBM Z/OS, призначений для управління установкою та оновленням програмних продуктів. Іншими словами, SYSMOD – це пакет змін, який вноситься в систему для додавання нових функцій, виправлення помилок або оновлення існуючих компонентів.

Процес SYSMOD, представлений в другому розділі створює зв’язок між варіантами використання та вимогами, визначеними на етапі моделювання вимог. На рисунку 3.1 представлені варіанти використання, визначені для візуального рівня (VL), і пов’язані варіанти використання з пакетами вимог, представленими в другому розділі 3.2. Вимоги згруповані в пакети, які дозволяють зрозуміле візуальне представлення зв’язку між запропонованими варіантами використання та пакетами вимог.

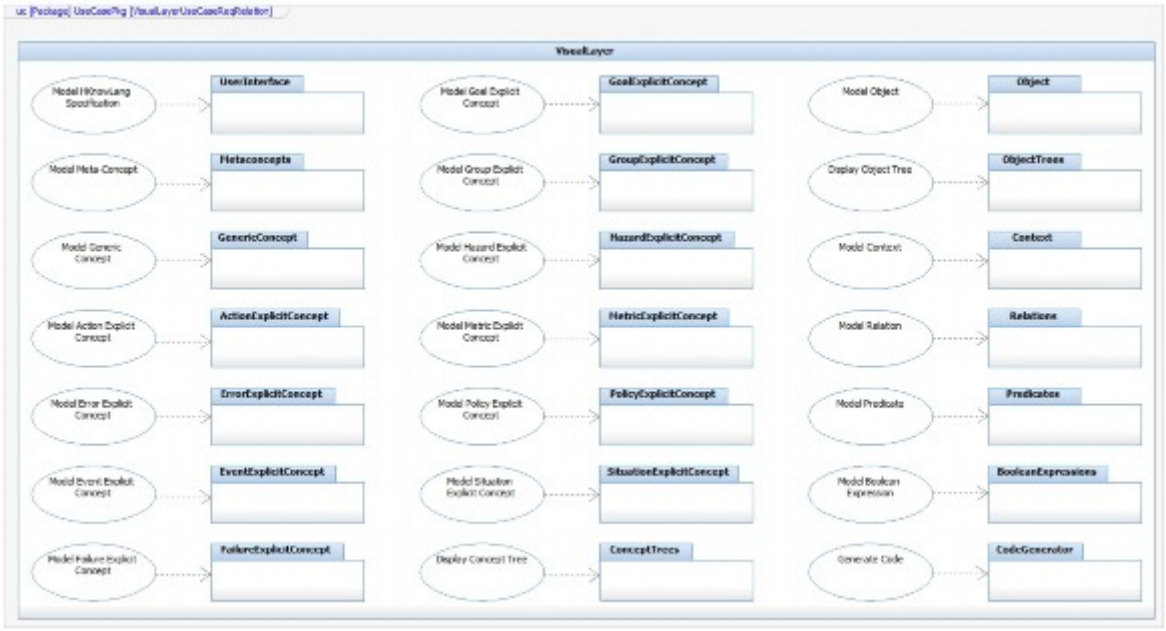

Рис. 3.1. Взаємозв’язок варіантів використання та вимог

Розглянемо діаграми варіантів використання, які ілюструють внутрішні зв'язки між варіантами використання системи VL і взаємозв'язок між варіантами використання та суб'єктами, які взаємодіють із системою VL. Модель варіантів використання проілюстровано за допомогою трьох діаграм, які представляють різні рівні абстракції. На рисунку 3.2 зображено взаємозв'язок між візуальним рівнем VL та зовнішніми акторами.

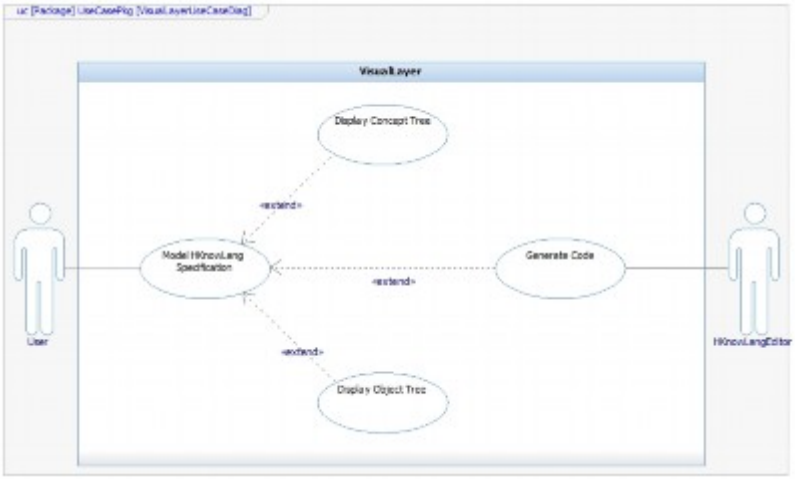

Рис. 3.2. Діаграма варіантів використання, що зображує (VL)

HKnowLang поєднує кілька підходів до подання знань, таких як логічне подання, продукційні системи, семантичні мережі, онтології та нечітку логіку. Це дозволяє використовувати найвідповідніші інструменти для кожного конкретного завдання або ситуації. Однією з ключових характеристик HKnowLang є підтримка інтеграції знань із різних джерел і систем, що працюють на різних моделях і парадигмах подання знань. Це досягається через підтримку онтологій та семантичних технологій, що дозволяють інтерпретувати та зіставляти різні форми знань.

HKnowLang базується на стандартах подання знань, таких як OWL (Web Ontology Language) та RDF (Resource Description Framework), що забезпечує сумісність з іншими системами та підвищує її інтеграційні можливості.

Випадок використання специфікації моделі HKnowLang є узагальненим представленням інших варіантів використання, як показано на рисунку 3.3.

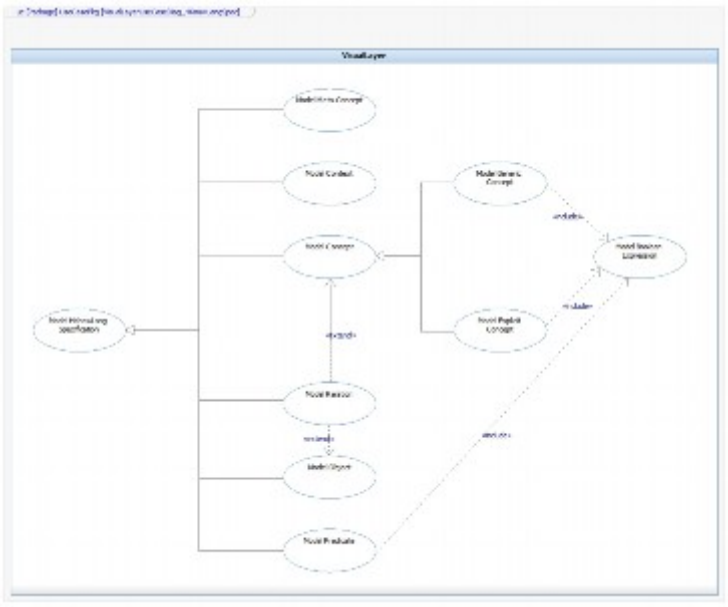

Рис. 3.3. Діаграма варіантів використання, що відображає узагальнення варіантів використання специфікації HKnowLang

Випадок використання моделі Explicit Concept на рисунку 3.3 є подальшим узагальненням усіх випадків використання, пов’язаних із явними концепціями, як показано на рисунку 3.4.

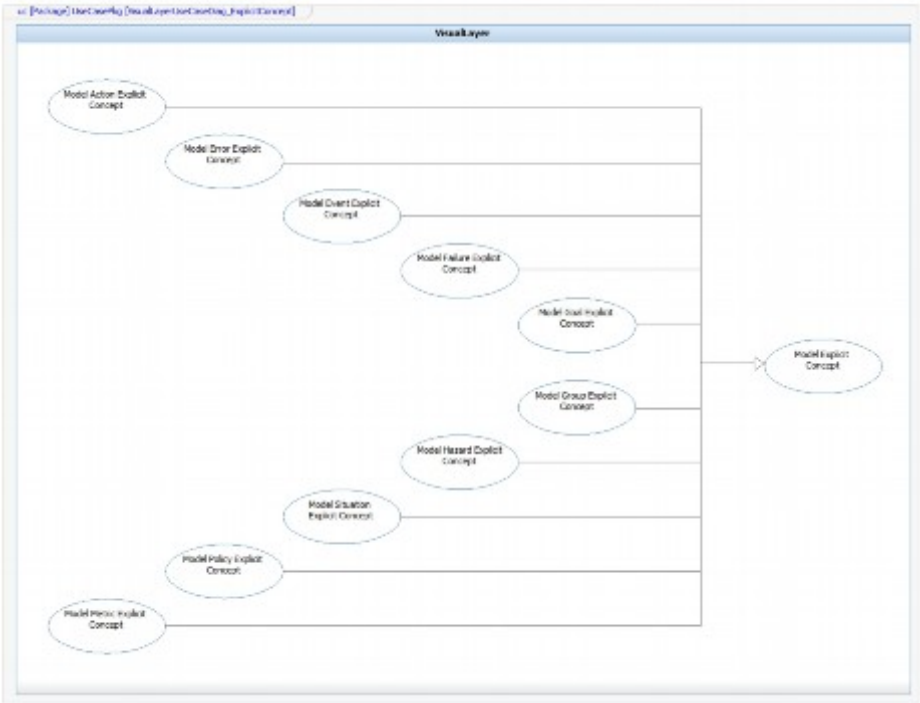

Рис. 3.4. Діаграма варіантів використання, що зображує узагальнення явної концепції

Кожен варіант використання має детальний опис. Детальний опис моделі дії Explicit Concept показано на рисунку 3.5.

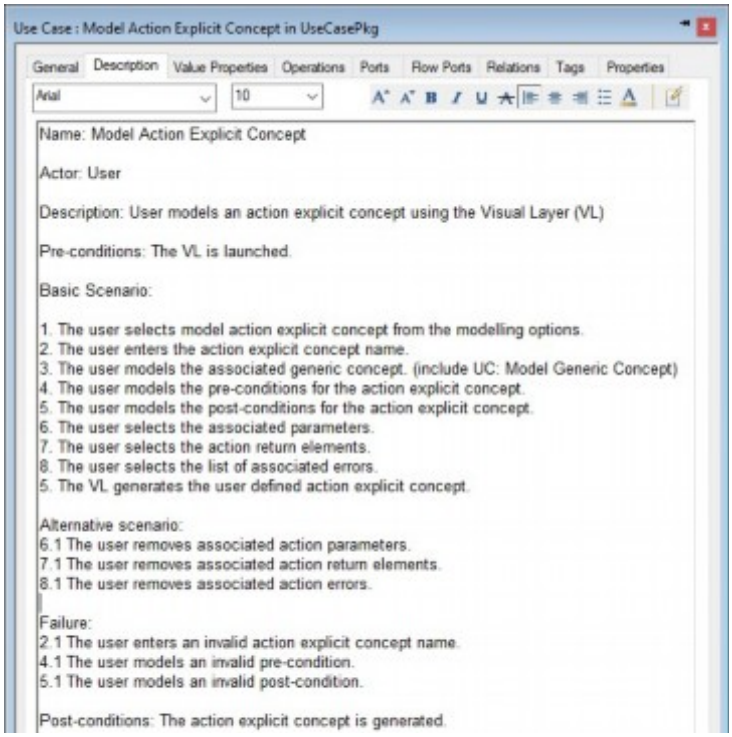

Рис. 3.5. Детальний опис випадку використання явної концепції моделі дії

Розглянемо опис декількох випадків використання.

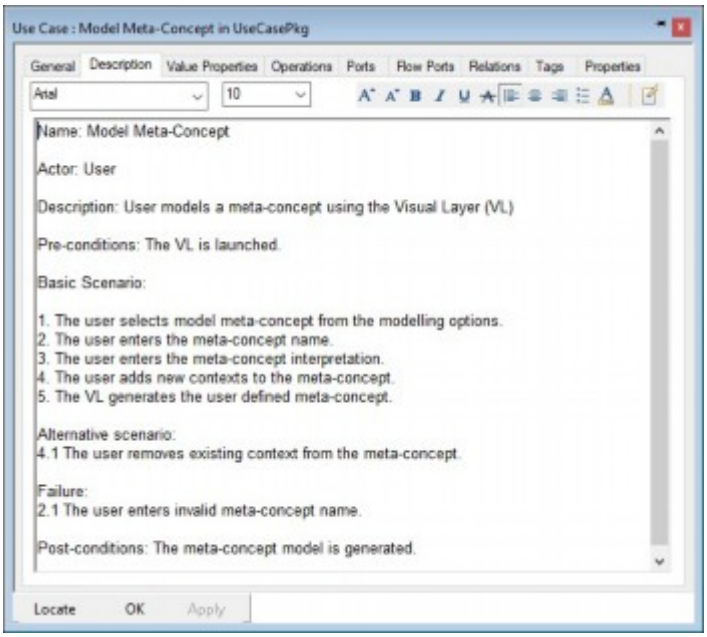

Рис. 3.6. Опис сценарію використання моделі Metaconcept

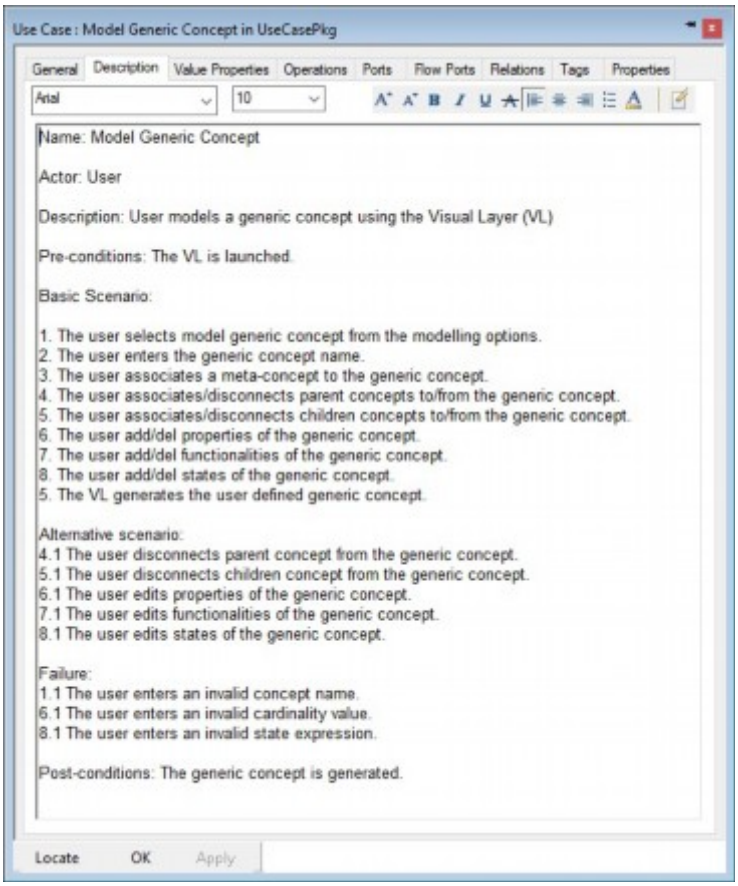

Рис. 3.7. Опис загального випадку використання моделі

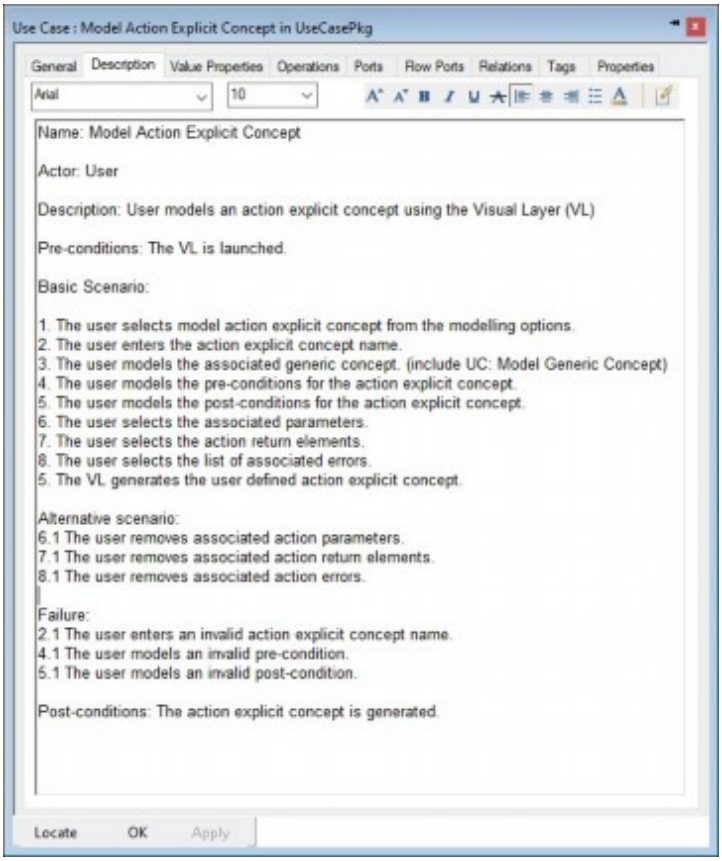

Рис. 3.8. Опис явного випадку використання дії моделі

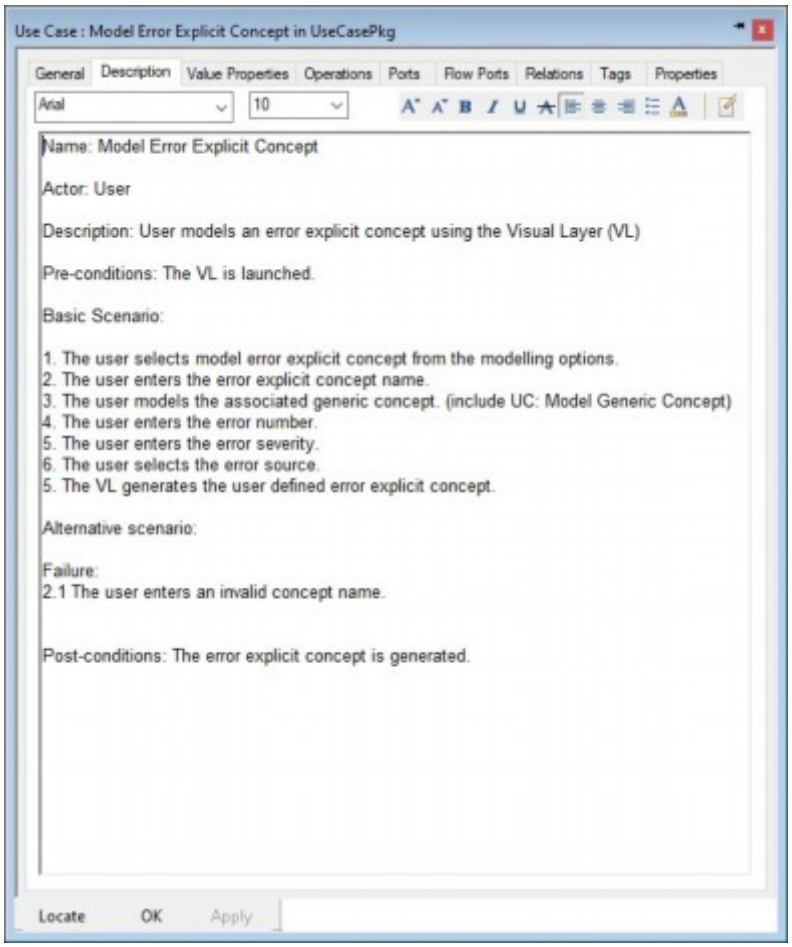

Рис. 3.9. Опис явного випадку використання концепції помилки моделі

Цей варіант підкреслює, що мова йде про конкретну ситуацію, коли модель допускає помилку.

3.2. Мета-модель дизайну візуального рівня

Розробка метамоделі виконується в IBM Rational Rhapsody, як зазначено в другому розділі. Конструкції HKnowLang, які потребують моделювання, представлені на рисунку 1.1. Метамодель представляє зв’язок між конструктами HKnowLang, які мають бути змодельовані для VL. Ми використовуємо блок-схеми, які представляють різні рівні абстракції, щоб представити мета-модель. На рисунку 3.10 представлено високорівневий вигляд мета-моделі HKnowLang. Додаткові блок-схеми використовуються для детальнішої ілюстрації метамоделі кожного блоку, показаного на

рисунку 3.10. IBM Rational Rhapsody генерує код Java, що відповідає метамоделям. Цей згенерований код потім використовується для розробки внутрішньої моделі даних для VL.

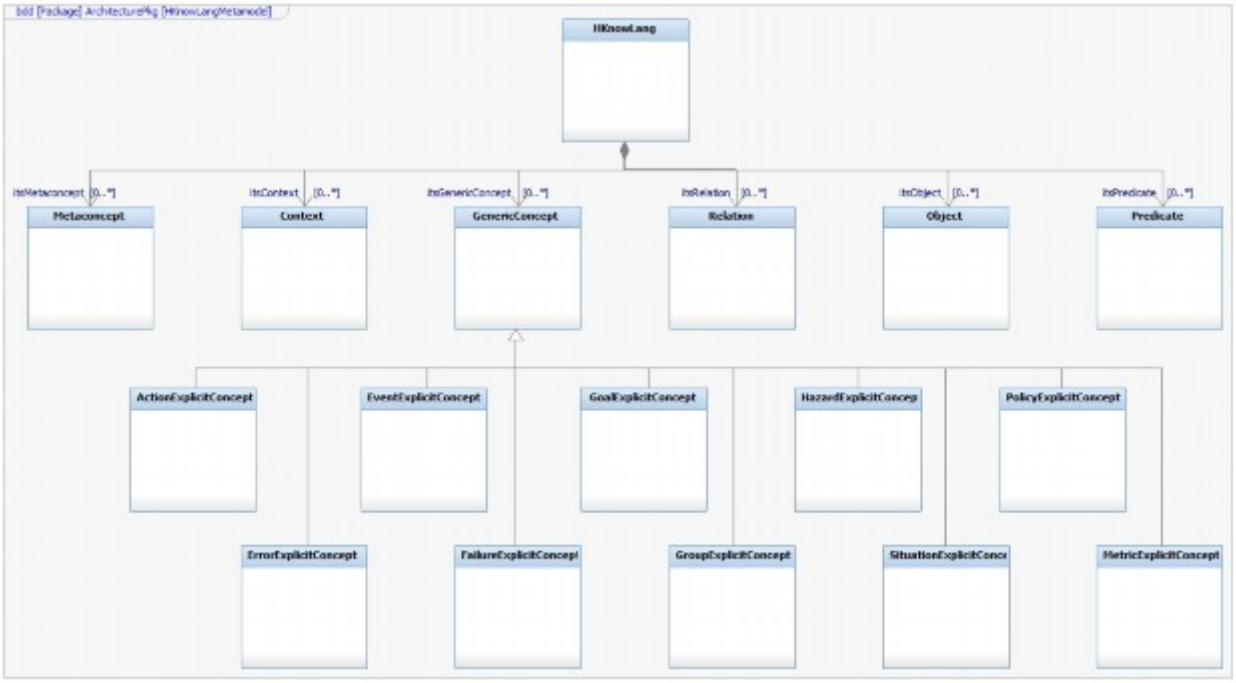

Рис. 3.10 - Архітектура метамоделі

Метаконцепти забезпечують контекстно-орієнтовану інтерпретацію концептів і можуть бути необов'язково пов'язані з конкретними контекстами. Мета метаконцепцій полягає в тому, щоб дозволити онтології розглядатися з різних контекстних перспектив шляхом встановлення різних значень для деяких ключових концепцій [30]. На рисунку 3.11 представлена мета-модель метаконцепції.

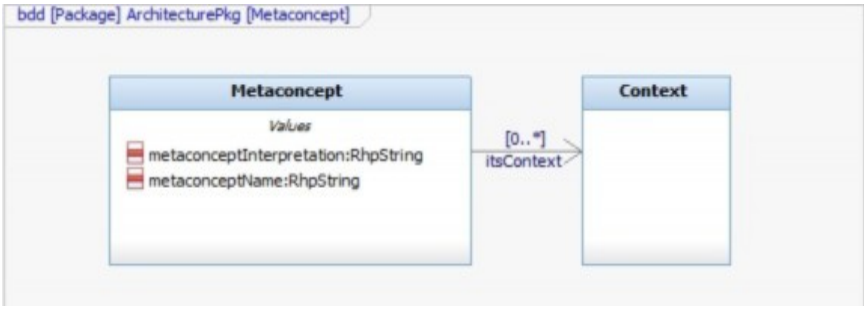

Рис. 3.11. Архітектура метамоделі специфікації мета концепції

3.3. Реалізація архітектури основних концепцій

Поняття складають основну термінологію HKnowLang. Поняття організовані в дерева, які семантично групують пов’язані поняття. Концепт може успадкувати мета-концепт. Кожна концепція має додатковий набір властивостей, функціональних можливостей, батьківських концепцій і дочірніх концепцій. Поняття також може мати асоційовані з ним стани [30]. На рисунку 3.12 представлена загальна концептуальна мета-модель.

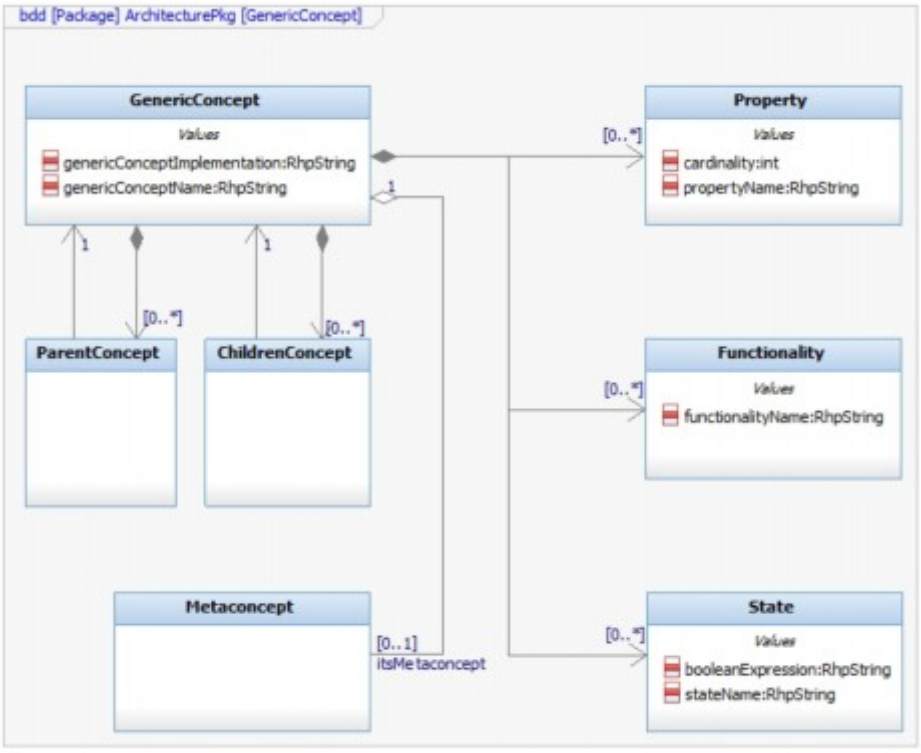

Рис. 3.12. Архітектура мета-моделі специфікації загальної концепції

3.3.1. Явна концепція дії

У HKnowLang дії — це дії, які можуть виконуватися системою та реалізовані в середовищі виконання або в самій системі. Концепція дії відноситься до реальної реалізації, такої як клас або метод [30]. На рисунку 3.13 представлена мета-модель явних понять дії.

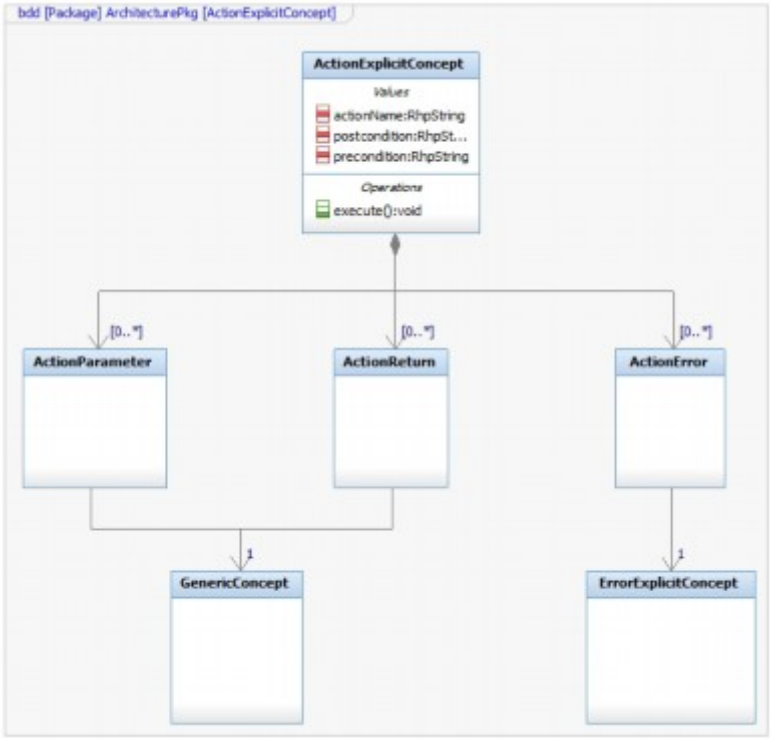

Рис. 3.13. Архітектура мета-моделі специфікації явної концепції дії

3.3.2. Концепція явної помилки

У HKnowLang помилки — це явні поняття, що представляють простір помилок, які можуть виникнути в самій системі або в середовищі виконання. Помилка вказується з інформацією про помилку та оптимальним набором помилкових дій, які можна розглядати як можливі джерела цієї помилки. Виникнення помилки може спричинити перехід стану.

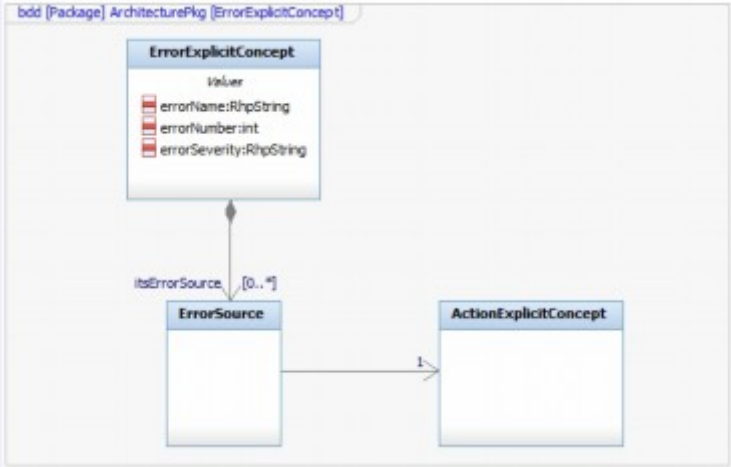

Рис. 3.14. Архітектура мета-моделі специфікації явної концепції помилки

Помилки забезпечують прогностичний простір цінної інформації, яка може бути використана для міркувань [30]. На рисунку 3.14 представлена мета-модель явної концепції.

3.3.3. Явна концепція події

Подія в HKnowLang є засобом високопріоритетного моніторингу та обміну повідомленнями. Подія може бути активована різними факторами, такими як час, ціль, показники, помилки, дії та інші події. Спеціальний захист, представлений у вигляді логічного виразу, може обмежити активацію події. Події можуть брати участь у логічних виразах або використовуватися для визначення політики, цілей і ситуацій, керованих подіями [30]. На рисунку 3.15 представлена мета-модель явної концепції події.

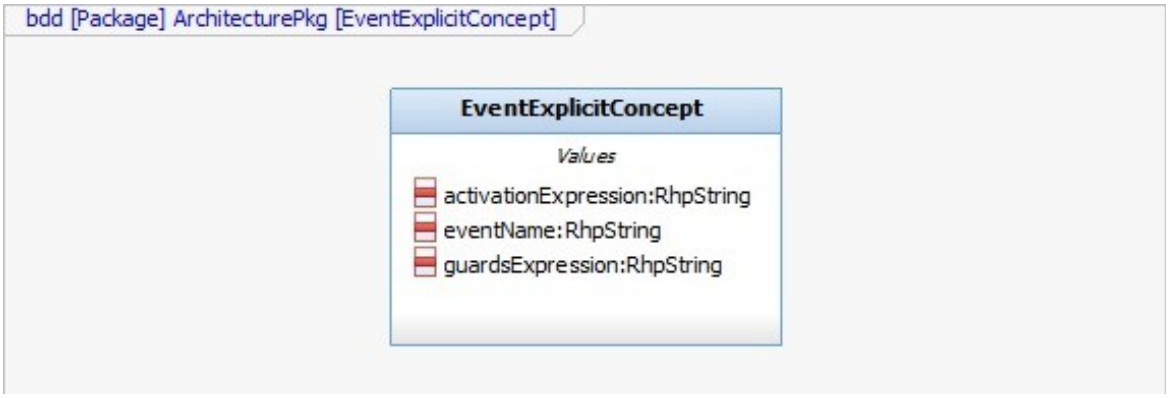

Рис. 3.15. Архітектура метамоделі специфікації явної концепції події

3.3.4. Явна концепція відмови

Концепція явної відмови забезпечує вбудовану підтримку класів помилок, таких як упущення та вчинення. Концепція відмови може бути пов’язана з несправною частиною та може використовуватися для вказівки спеціальної відмови. Спеціальна логіка відмов реалізована для оцінених відмов. Нижче, на рисунку 3.16 представлена мета-модель явної концепції відмови.

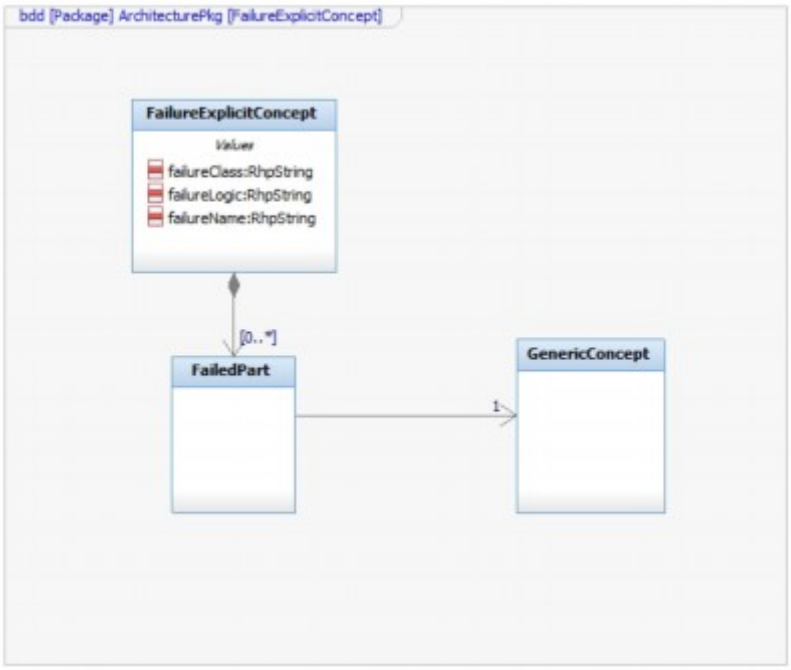

Рис. 3.16. Архітектура мета-моделі специфікації явної концепції відмови

3.3.5. Явна концепція цілі

Ціль HKnowLang — це чітке поняття, яке використовується для вираження системних і самоадаптивних цілей. Цілі HKnowLang використовуються для визначення політик HKnowLang. Ціль виражається як перехід від стану відправлення до стану прибуття. На рисунку 3.17 представлена мета-модель чіткої концепції мети.

Рис. 3.17. Архітектура мета-моделі специфікації чіткої концепції мети

3.3.6. Явна концепція групи

Група включає загальні поняття, явні поняття та об’єкти, пов’язані один з одним через чіткий набір відносин. Групи — це явні поняття,

призначені для представлення знань про структуру системи [30]. На рисунку 3.18 представлена мета-модель групової явної концепції.

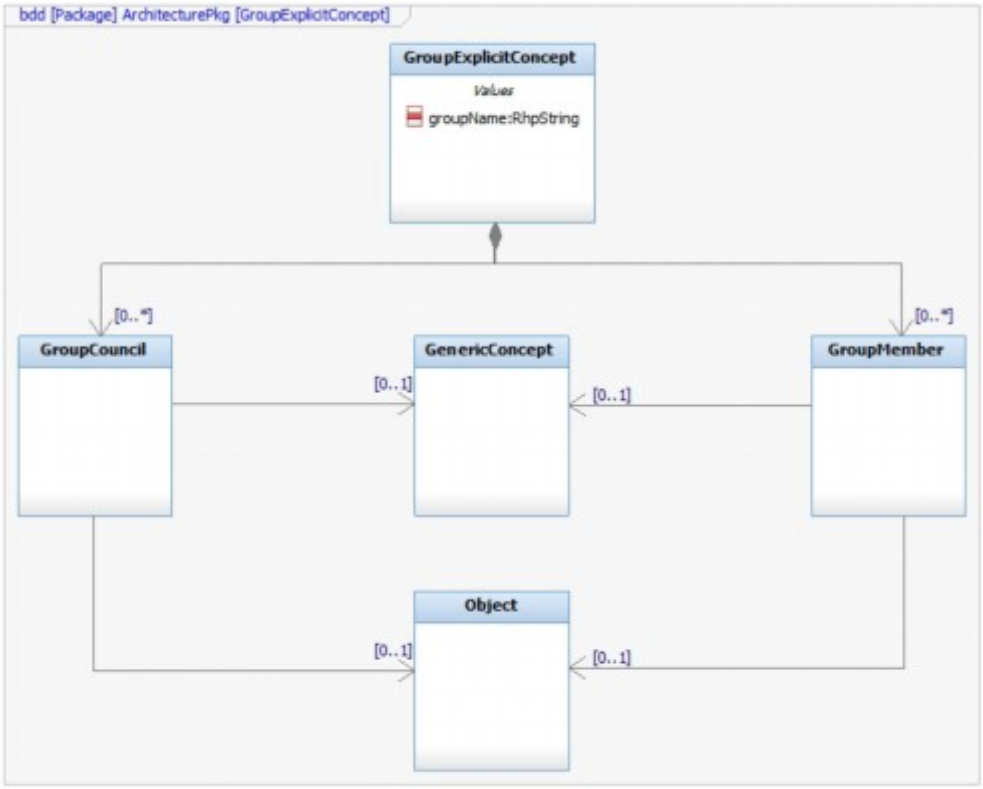

Рис. 3.18. Архітектура мета-моделі специфікації явної концепції групи

3.3.7. Явна концепція небезпеки

Небезпека — це тип явної концепції HKnowLang, яка представляє небезпеки, з якими система повинна мати справу. Концепція явної небезпеки реалізована для підвищення виразності HKnowLang з точки зору цілей безпеки та вирішення вимог функціональної безпеки автомобіля [30]. На рисунку 3.19 представлена мета-модель явної концепції небезпеки.

3.3.8. Явна концепція ситуації

У HKnowLang ситуація представляє випадок життя системи, коли реалізується певний стан. Ситуації вказуються на критичні моменти, при яких система потребуватиме самоадаптації. Основними атрибутами ситуації

є стани ситуації та дії ситуації. На рисунку 3.20 представлена явна концептуальна мета-модель ситуації.

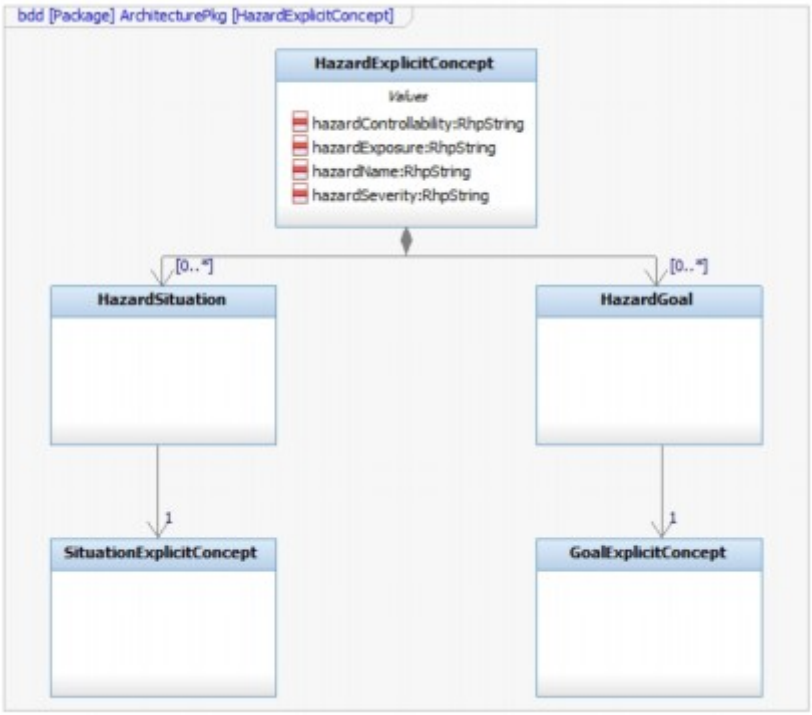

Рис. 3.19. Архітектура мета-моделі специфікації явної концепції небезпеки

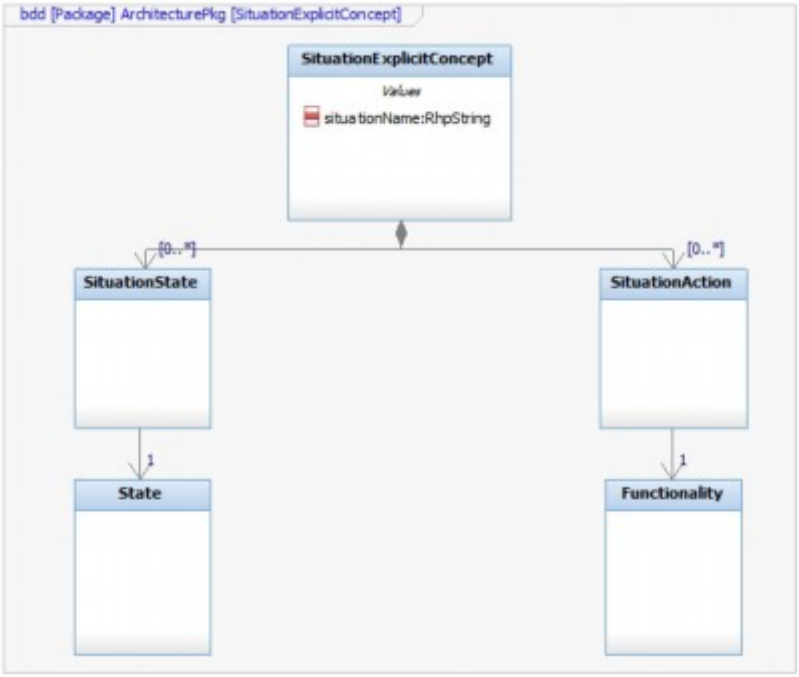

Рис. 3.20. Архітектура мета-моделі специфікації явної концепції ситуації

3.3.9. Концепції чіткої політики та явної метрики

Політика HKnowLang призначена для забезпечення самоадаптивної поведінки. Політика є складною структурою специфікації, оскільки вона поєднує в собі специфікацію цілей, ситуацій, дій і відносин. Політики також вбудовують розподіл ймовірностей у вказану модель поведінки, яка використовується аргументом HKnowLang для машинного навчання [30]. На рисунку 3.21 представлена мета-модель явної концепції політики.

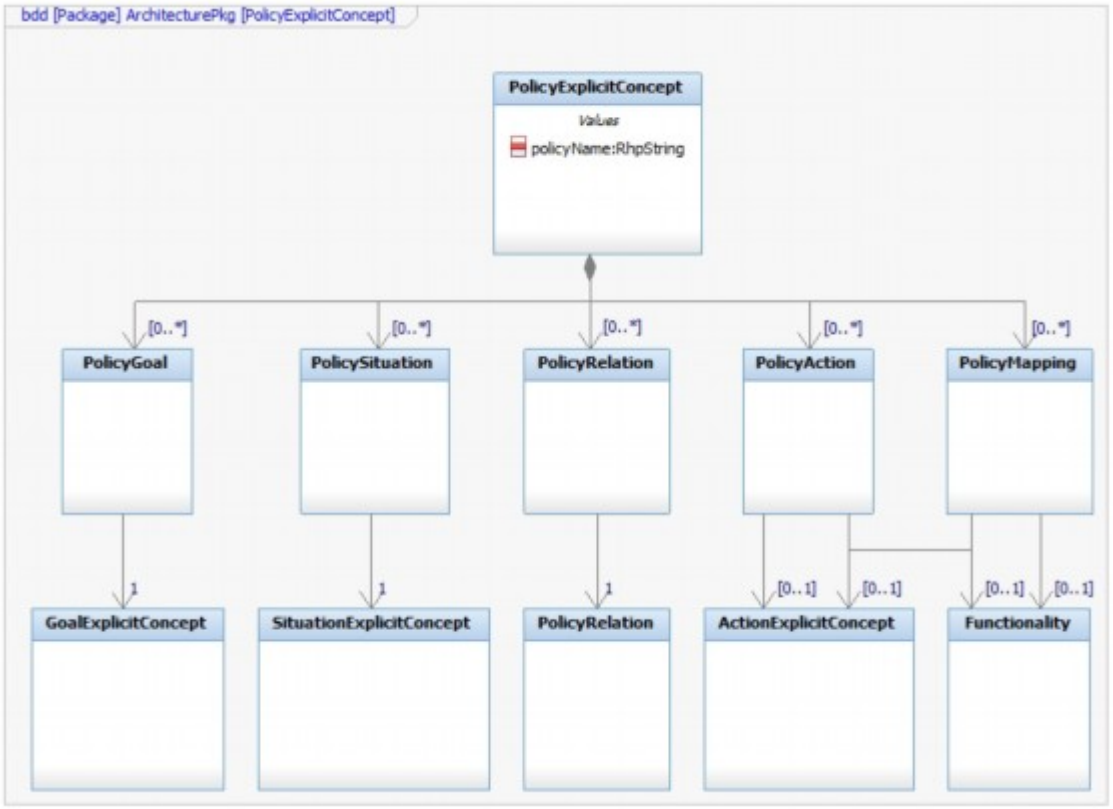

Рис. 3.21. Архітектура мета-моделі специфікації чіткої концепції політики

Метрики — це чіткі концепції, що забезпечують прогностичний простір цінної інформації, яку можна зібрати з середовища або з самої системи. Метрика вказується з джерелом метрики та даними. Джерело метрики може являти собою системний датчик, який використовується для моніторингу навколишнього середовища, або саму систему [30]. На рисунку 3.22 представлена мета-модель метричної явної концепції.

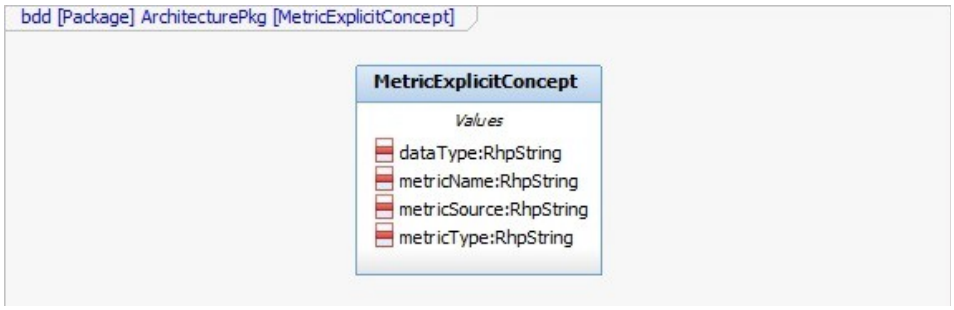

Рис. 3.22. Архітектура мета-моделі специфікації явної концепції метрики

3.3.10. Контекст, відношення та об’єкт

Контексти HKnowLang призначені для отримання відповідних знань з онтології HKnowLang. Контексти містять інтерпретацію деяких мета-концептів, що може призвести до нової інтерпретації концептів-нащадків [30]. На рисунку 3.23 представлена мета-модель контекстної концепції.

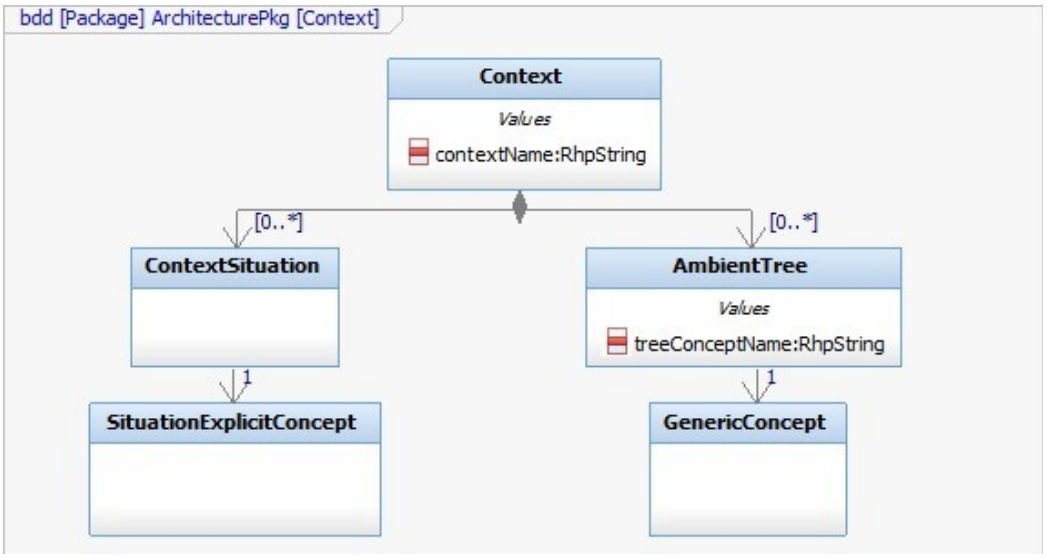

Рис. 3.23. Архітектура метамоделі специфікації контексту

Зв’язки HKnowLang з’єднують поняття та об’єкти, забезпечуючи залежності, виміряні ймовірною вагою. Будь-яке поняття або об'єкт може бути пов'язане з іншим поняттям або об'єктом. Відношення HKnowLang є двійковим. На рисунку 3.24 представлена мета-модель відношення.

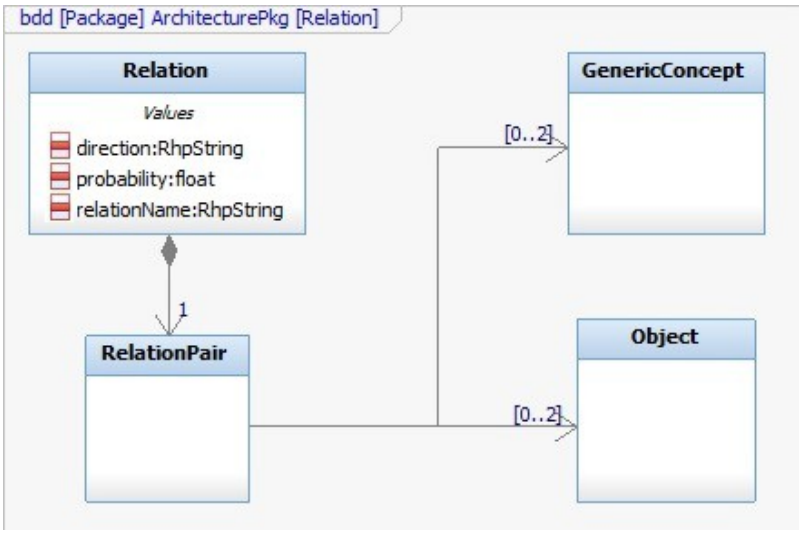

Рис. 3.24. Архітектура метамоделі специфікації зв’язку

Об’єкти в HKnowLang є екземплярами концептів і, отже, пов’язані з концептом, який вони створюють. На рисунку 3.25 представлена мета-модель об’єкта.

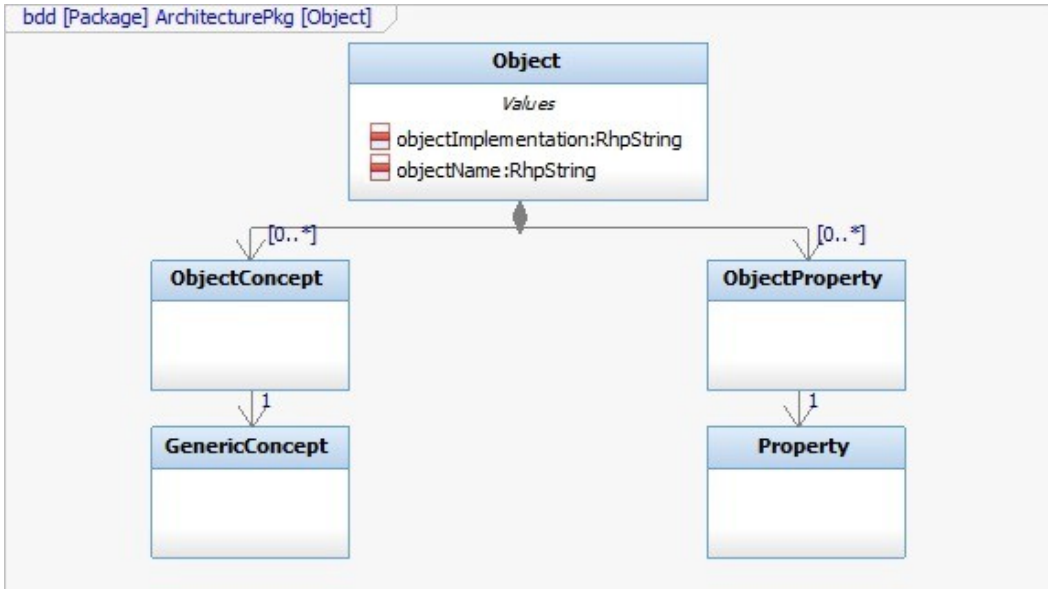

Рис. 3.25. Архітектура метамоделі специфікації об’єкта

Предикати в HKnowLang поєднують стани кількох концептів і можуть розглядатися як стани складної системи, оскільки їх оцінка залежить від оцінки станів залучених концептів [30]. На рисунку 3.26 представлена мета-модель предикату.

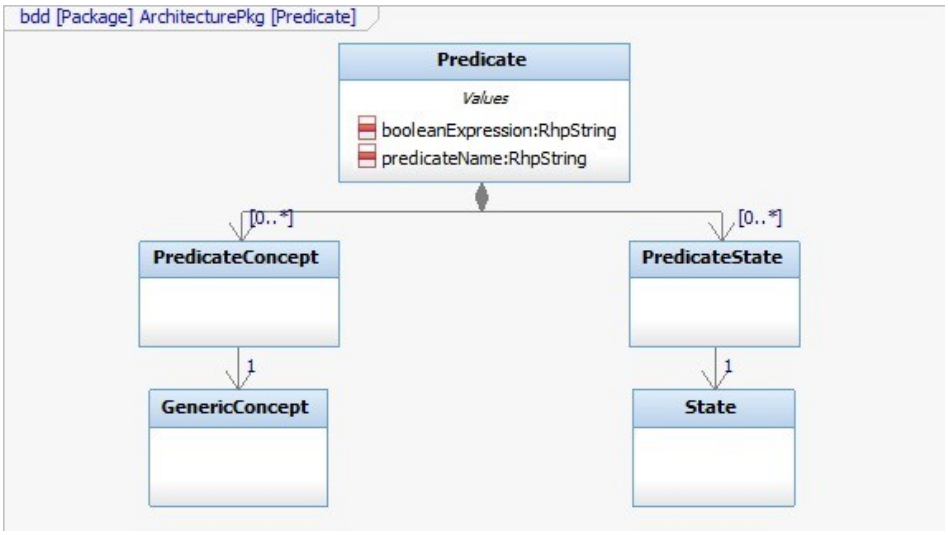

Рис. 3.26. Архітектура метамоделі специфікації предикату

3.4. Реалізація графічного інтерфейсу для візуального рівня

У цьому розділі докладно розглядається дизайн графічного інтерфейсу користувача (GUI). Розробка програми VL виконується в IDE Apache NetBeans. Проектування VL складається з таких двох основних завдань:

• Створення полотна та палітри інструментів.

• Розробки графічного інтерфейсу для кожної конструкції HKnowLang на панелі інструментів.

Для цієї задачі використовується NetBeans Visual Library API [31] для розробки полотна та палітри інструментів. Палітра інструментів містить конструкції HKnowLang, які можна використовувати для розробки онтології HKnowLang.

Полотно — це робочий простір для розробки онтології HKnowLang. Користувач може перетягнути необхідну конструкцію HKnowLang з палітри інструментів на полотно. Щоб створити відношення, потрібно натисніть ліву кнопку миші та виділити конструкції відношення, утримуючи кнопку Ctrl. Інтерфейс користувача для реалізації візуального рівня (VL) показано на рисунку 3.27.

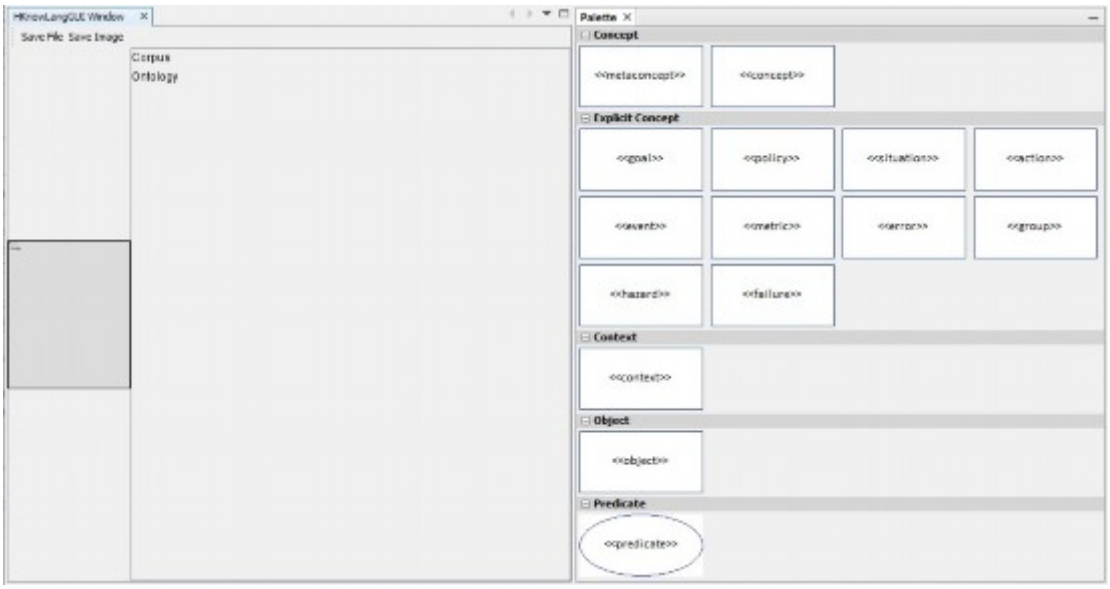

Рис. 3.27. Вигляд інтерфейсу користувача реалізації візуального рівня

Графічний інтерфейс для конструкцій HKnowLang розроблено з використанням бібліотеки Java Swing [32]. Рисунки 3.28 – 3.33 ілюструють графічний інтерфейс верхнього рівня, розроблений для кожної конструкції.

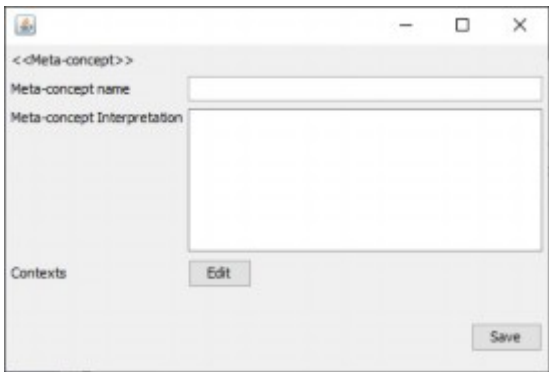

Рис. 3.28. Інтерфейс користувача мета концепції

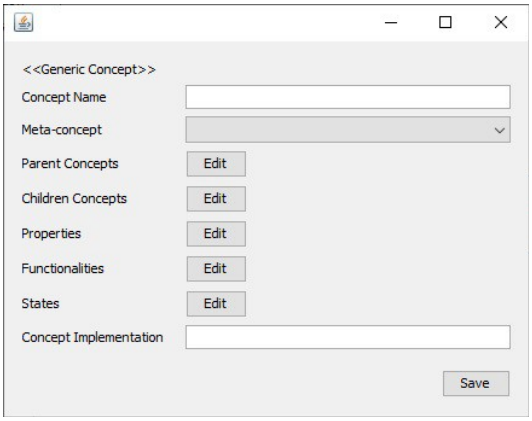

Рис. 3.29. Загальна концепція інтерфейсу користувача

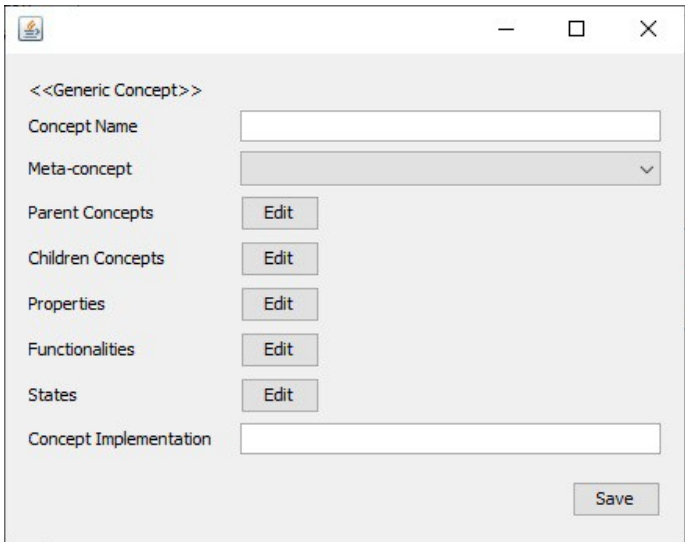

Рис. 3.30. Інтерфейс користувача з явною концепцією помилок

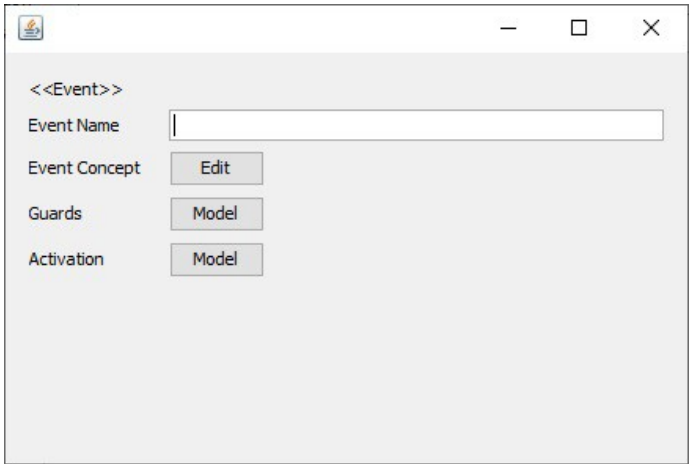

Рис. 3.31. Інтерфейс користувача з явною концепцією події

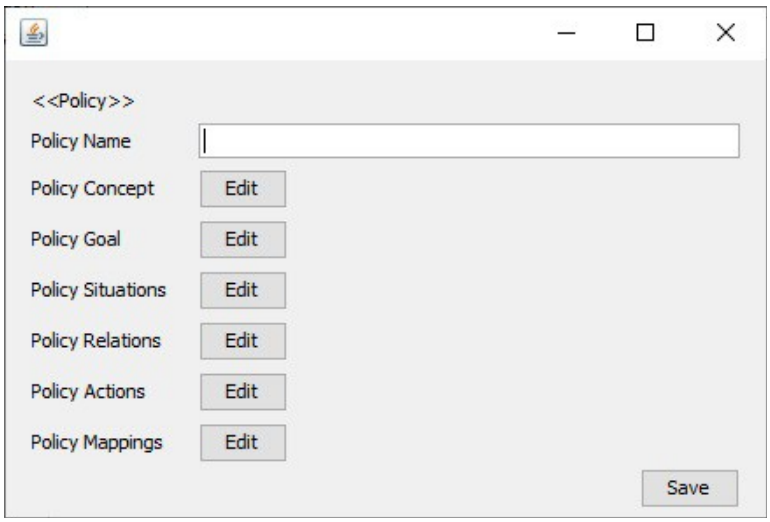

Рис. 3.32. Інтерфейс користувача з чіткою концепцією політики

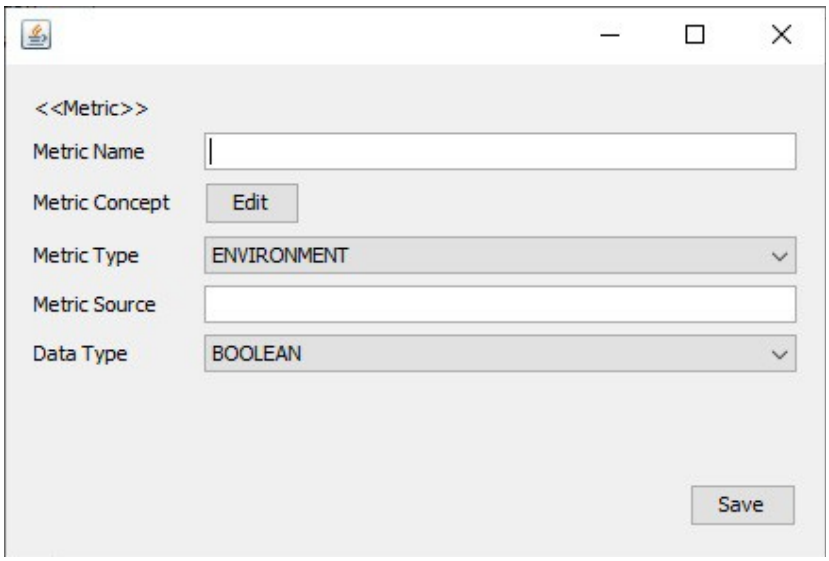

Рис. 3.33. Інтерфейс користувача з явною концепцією метрики

Ми використовуємо спеціально розроблений генератор коду для перетворення визначеної користувачем моделі HKnowLang у код HKnowLang, сумісний із компілятором HKnowLang.

Інтерфейс VL GUI інтегровано з внутрішньою мета-моделлю HKnowLang для створення програми VL. Специфікаційна функція визначається для кожної специфікації HKnowLang для перетворення моделі специфікації на еквівалентний код HKnowLang. Ця функція інтегрована з кодом внутрішньої мета-моделі.

VL функціонує на основі концепції об’єкта. Коли нова специфікація HKnowLang додається до полотна у VL, створюється екземпляр вибраної специфікації. VL підтримує список усіх об'єктів. Щойно створений об’єкт додається до цього списку об’єктів у поточному проекті візуального моделювання. Під час збереження поточного проекту моделювання запускається функція для перетворення даних моделі на необхідний синтаксис HKnowLang. Функція отримує доступ до кожного об’єкта в моделі та викликає певний метод, пов’язаний із класом об’єкта, який перетворює дані специфікації HKnowLang на еквівалентний код. На рисунку 3.34 показана діаграма активності, яка ілюструє процес генерації коду.

У лістингу 3.1 зображено функцію для перетворення даних моделі в код. Лістинг 3.2 демонструє генератор коду для конкретної конструкції HKnowLang на прикладі концепції Action Explicit.

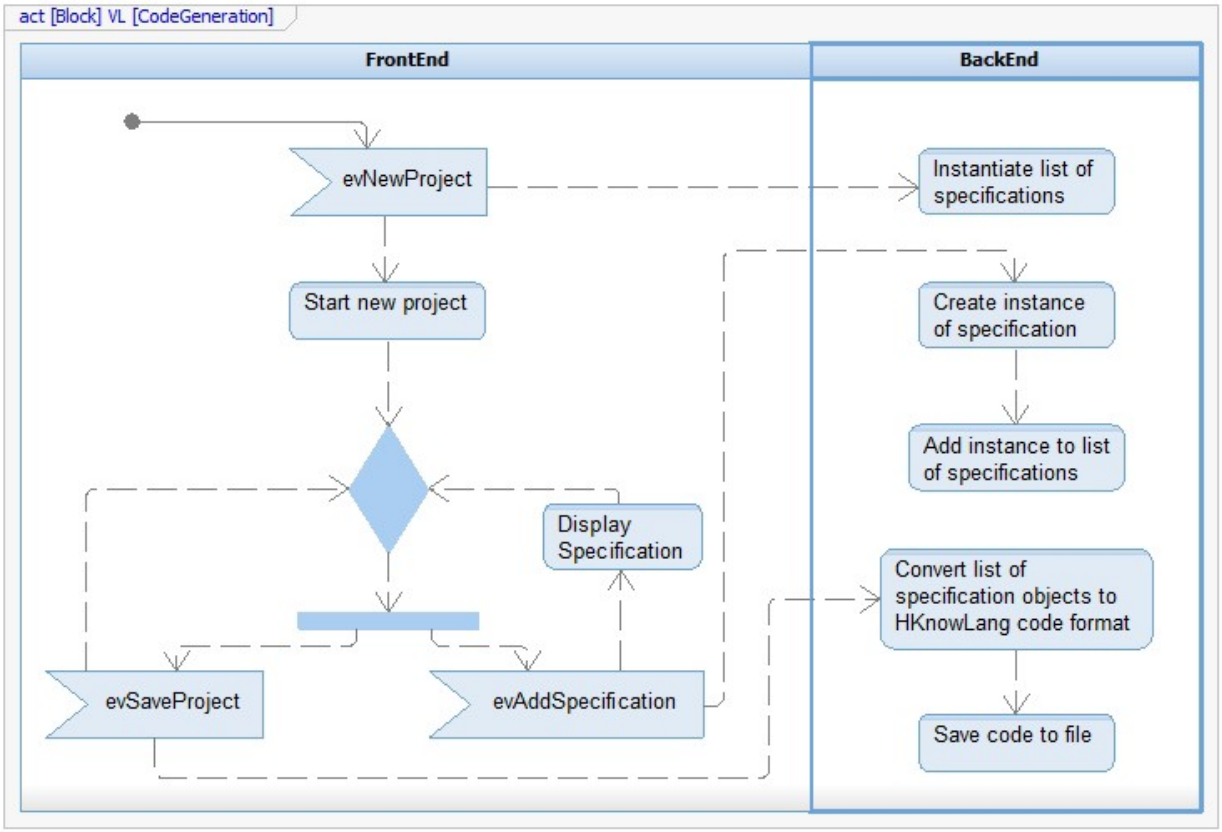

Рис. 3.34. Діаграма діяльності, що зображує процес генерації коду

Лістинг 3.1.

Функція для перетворення даних моделі в код

```
private String dataToString() {

    text = "// KnowLang Specification\n"
            + "// " + this.corpus + "\n
              //=============================================================\n"
            + "CORPUS " + this.corpus + " {\n"
            + "\tONTOLOGY " + this.ontology + " {\n\n";

    //Metaconcepts
    text += "\t\tMETACONCEPTS {\n";
    for (int i=0; i<metas.size(); i++) {
        text += metas.get(i).toString();
    }
    text += "\t\t}\n\n";

    //Concept trees
    text += "//Concept trees\n"
            + "\t\tCONCEPT_TREES {\n";
```

```
    //Concept
    for (int i=0; i<concepts.size(); i++) {
        text += concepts.get(i).toString();
    }
    text += "\n";

    //Actions
    text += "\n\n// Actions \n";
    for (int i=0; i<actions.size(); i++) {
        text += actions.get(i).actionToString();
    }

    //Metrics
    text += "\n\n// metrics \n";
    for (int i=0; i<metrics.size(); i++) {
        text += metrics.get(i).metricToString();
    }

    //Situations
    text += "\n\n// Situations \n";
    for (int i=0; i<situations.size(); i++) {
        text += situations.get(i).situationToString();
    }

    //Goals
    text += "\n\n// Goals \n";
    for (int i=0; i<goals.size(); i++) {
        text += goals.get(i).goalToString();
    }

    //Policies
    text += "\n\n// Policies \n";
    for (int i=0; i<policies.size(); i++) {
        text += policies.get(i).policyToString();
    }

    //end concept tree
    text += "\n\t\t}  // concept tress \n";


    //Relations
    text += "\n\n// Relations \n";
    if (!relations.isEmpty()) {
        text += "\n\t\tRELATIONS {\n";
        for (int i=0; i<relations.size(); i++) {
            text += relations.get(i).relationToString();
        }
        text += "\t\t}\n";


    }

    //Object Trees
    text += "\n\n// Object Trees \n";

    //end of Ontology
    text += "\n\t} //end of Ontology \n}\n";

    return text;
}
```

Лістинг 3.2.

Генератор коду для конструкції на прикладі концепції Action Explicit

```
public String actionToString()
{
    String text;
    text = "\t\t\tCONCEPT_ACTION " + concept_name + " {\n";
    if (!parentconcepts.isEmpty()) {
        text += "\t\t\t\tPARENTS {";
        for (int i=0; i<parentconcepts.size(); i++) {
            if (i==0)
                text += path + "CONCEPT_TREES." + parentconcepts.get(i).
                    getConcept_name();
            else
                text += ", " + path + "CONCEPT_TREES." + parentconcepts.get(i).
                    getConcept_name();
        }
        text += "}\n";
    }
    //Children Concepts
    if (!childrenconcepts.isEmpty()) {
        text += "\t\t\t\tCHILDREN {";
        for (int i=0; i<childrenconcepts.size(); i++) {
            if (i==0)
                text += path + "CONCEPT_TREES." + childrenconcepts.get(i).
                    getConcept_name();
            else
                text += ", " + path + "CONCEPT_TREES." + childrenconcepts.get(i).
                    getConcept_name();
        }
        text += "}\n";
    }
    //Spec
    text += "\t\t\t\tSPEC {";
    if (!returns.isEmpty()) {
        text += "\n\t\t\t\t\tRETURN {";
        for (int i=0; i<returns.size(); i++) {
            if(i==0) {
                String compare = returns.get(i).getConcept().getConcept_name();
                if ("BOOLEAN".equals(compare)||"NUMBER".equals(compare)||"STRING".
                    equals(compare)||"TIME".equals(compare)||"DATETIME".equals(
                    compare))
                    text += returns.get(i).getConcept().getConcept_name();
                else
                    text += path + "CONCEPT_TREES." + returns.get(i).getConcept().
                        getConcept_name();
            }
            else {
                String compare = returns.get(i).getConcept().getConcept_name();
                if ("BOOLEAN".equals(compare)||"NUMBER".equals(compare)||"STRING".
                    equals(compare)||"TIME".equals(compare)||"DATETIME".equals(
                    compare))
                    text += ", " + returns.get(i).getConcept().getConcept_name();
                else
                    text += ", " + path + "CONCEPT_TREES." + returns.get(i).
                        getConcept().getConcept_name();
            }


        }
        text += "}\n\t\t\t\t";
    }
    if (!parameters.isEmpty()) {
        text += "\n\t\t\t\t\tPARAMS {";
```

```
        for (int i=0; i<parameters.size(); i++) {
            if(i==0) {
                String compare = parameters.get(i).getConcept().getConcept_name();
                if ("BOOLEAN".equals(compare)||"NUMBER".equals(compare)||"STRING".
                    equals(compare)||"TIME".equals(compare)||"DATETIME".equals(
                    compare))
                    text += parameters.get(i).getConcept().getConcept_name();
                else
                    text += path + "CONCEPT_TREES." + parameters.get(i).getConcept
                        ().getConcept_name();
            }
            else {
                String compare = parameters.get(i).getConcept().getConcept_name();
                if ("BOOLEAN".equals(compare)||"NUMBER".equals(compare)||"STRING".
                    equals(compare)||"TIME".equals(compare)||"DATETIME".equals(
                    compare))
                    text += ", " + parameters.get(i).getConcept().getConcept_name()
                        ;
                else
                    text += ", " + path + "CONCEPT_TREES." + parameters.get(i).
                        getConcept().getConcept_name();
            }
        }
        text += "}\n\t\t\t\t";
    }
    text += "}\n\t\t\t";
    text += "}\n";
    return text;
}
```

3.5. Перевірка роботи візуального рівня моделювання

Необхідно перевірити здатність генерувати код із специфікацій, змодельованих у VL, щоб підтвердити доказ концепції VL-додатку. У цьому розділі детально описано валідацію розробленого додатку. Процес перевірки передбачає виконання тестового сценарію в VL і перевірку коду HKnowLang, згенерованого VL. Метод перевірки коду складається з двох етапів, а саме:

• Компіляція згенерованого коду HKnowLang в існуючий компілятор HKnowLang, щоб перевірити синтаксис.

• Порівняння вручну згенерованого коду HKnowLang з еталонним тестовим сценарієм коду HKnowlang, щоб перевірити, чи всі функції зафіксовано.

Тестовий сценарій виведено з дослідження формалізації eMobility за допомогою HKnowLang [9]. eMobility – це транспортна концепція, заснована на мережі електричних транспортних засобів, яка враховує численні вимоги глобальної дорожньої ситуації та окремих водіїв, а також інфраструктурні та

експлуатаційні вимоги, включаючи наявність паркінгу, станції підзарядки та термін служби акумулятора [9]. Дослідження [9] представляє формальний підхід до моделювання систем eMobility та визначає специфікацію eMobility в HKnowLang.

Пропонований сценарій тестування передбачає моделювання специфікації eMobility у VL та генерацію відповідного коду HKnowLang. Синтаксис коду перевіряється шляхом імпортування коду в існуючий текстовий редактор HKnowLang і його компіляції. Згенерований код також перевіряється на контрольний код.

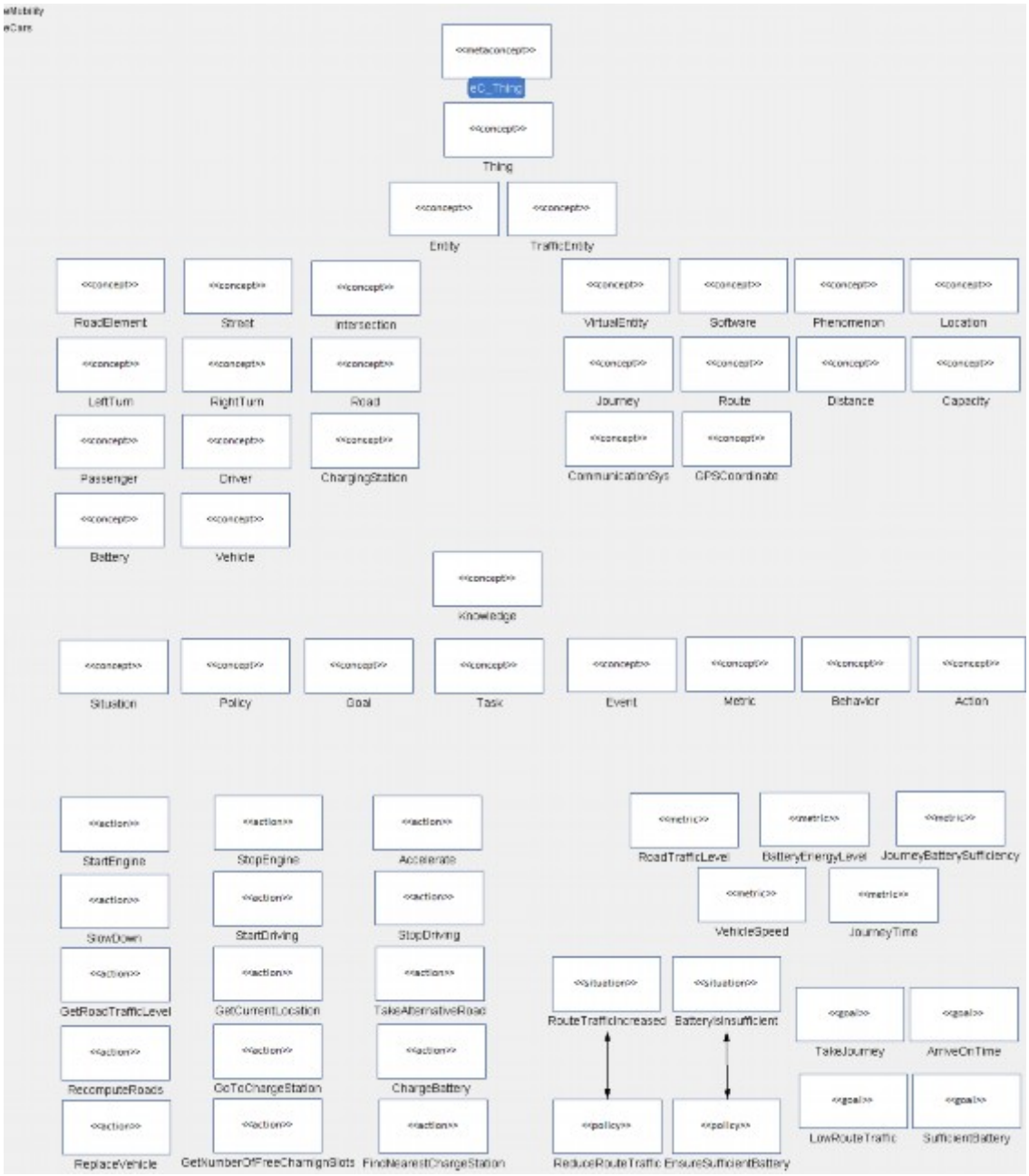

Рис. 3.35. Тестова модель специфікації

Виконання тестового сценарію передбачає моделювання специфікації HKnowLang eMobility за допомогою конструкцій HKnowLang у VL. Рисунок 3.35 ілюструє модель HKnowLang для eMobility у VL.

Щоб продемонструвати детальне представлення загальної концепції, візуальне моделювання концепції Street зображено на рисунку 3.36.

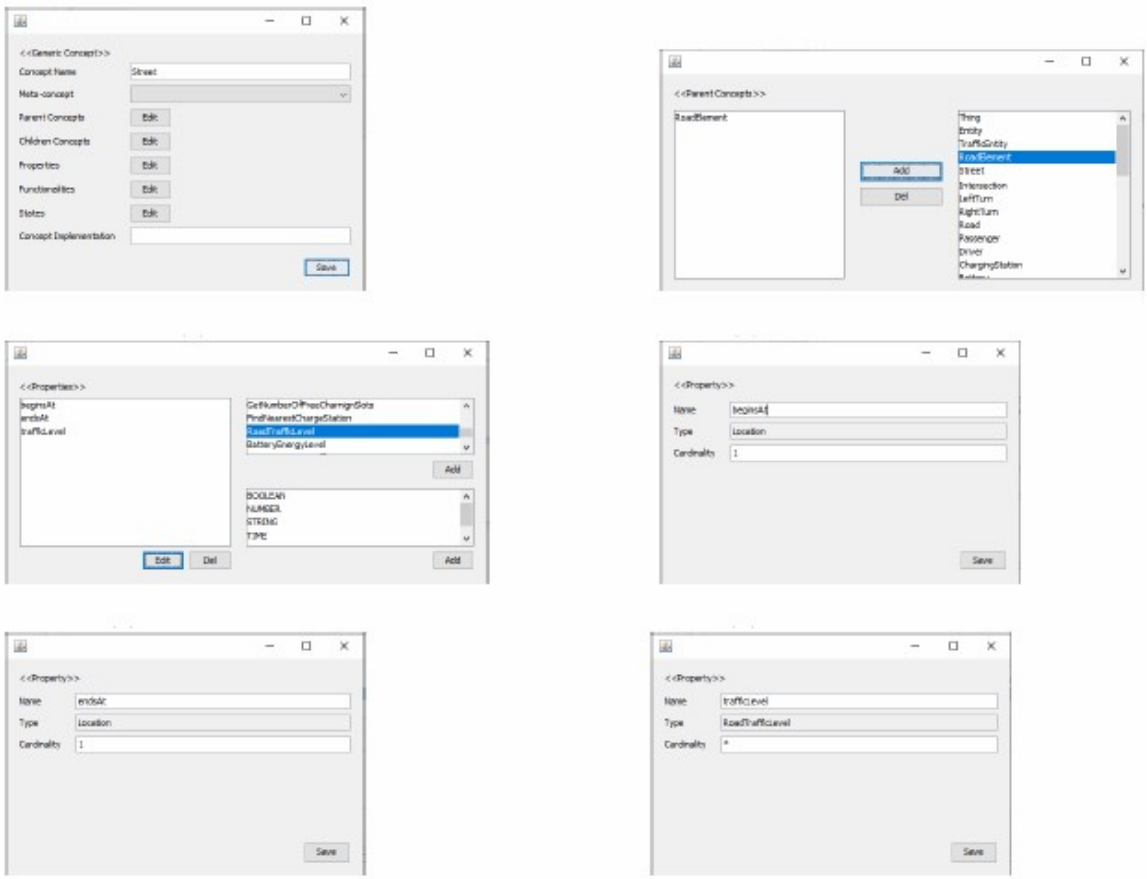

Рис. 3.36. Детальне моделювання концепції

Рисунок 3.37 ілюструє моделювання явної концепції з використанням концепції дії FindNearestChargeStation як приклад. Лінія з двома стрілками зображує явний зв’язок між концептуальними блоками, об’єднаними лінією.

Рисунок 3.38 ілюструє моделювання зв’язку між поняттями BatteryIsInsufficient і EnsureSufficientBattery. Після завершення моделювання код HKnowLang генерується та зберігається в системі. Згенерований код потім імпортується в текстовий редактор HKnowLang і компілюється.

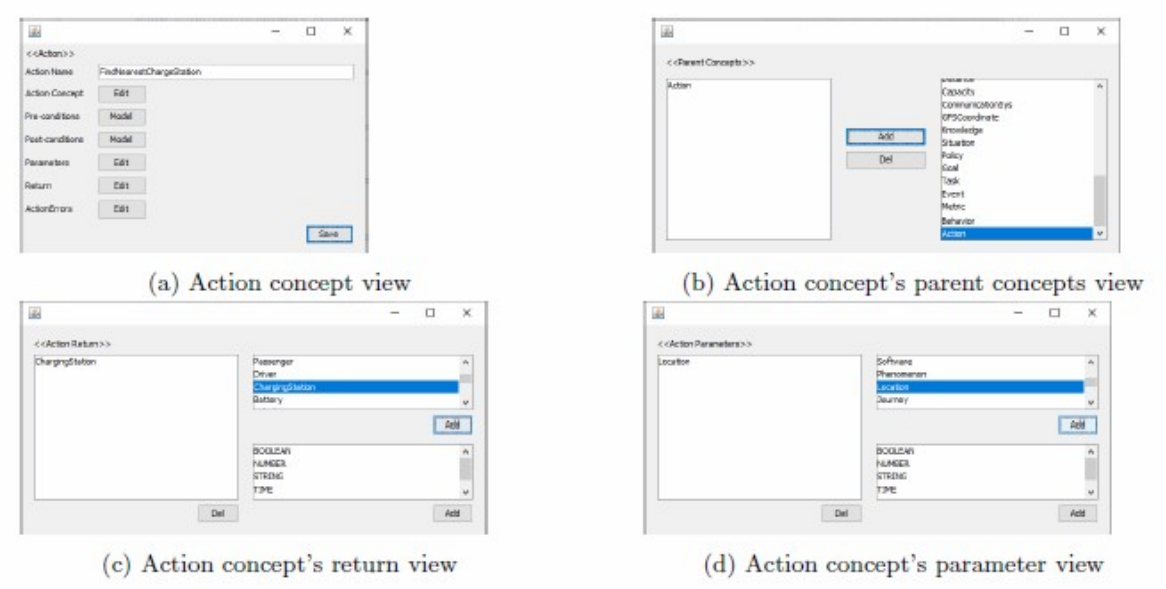

Рис. 3.37. Детальне моделювання явної концепції дії

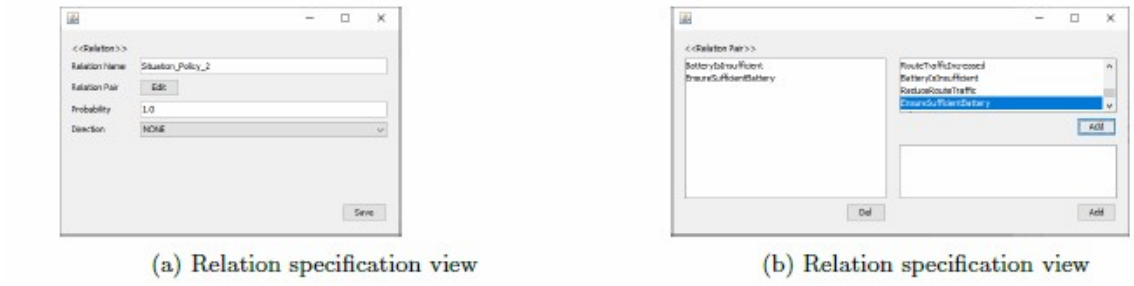

Рис. 3.38. Детальне моделювання специфікації відносин

Код, згенерований VL, перевіряється у два етапи:

1. Згенерований код компілюється в компіляторі HKnowLang. Результатом компіляції є узгоджена специфікація. Тому синтаксис згенерованого коду правильний і послідовний.

2. Згенерований код перевіряється вручну шляхом порівняння з еталонним кодом. Нижче наведено порівняння кодів для сегментів загальної концепції Street, концепції дії FindNearestChargeStation і відношення, зображеного на рисунку 3.38.

```
CONCEPT Street {
  PARENTS {eMobility.eCars.
      CONCEPT_TREES.RoadElement}
  CHILDREN {}
  PROPS {
    PROP beginsAt {TYPE {eMobility.
        eCars.CONCEPT_TREES.Location}
        CARDINALITY{1}}
    PROP endsAt {TYPE {eMobility.
        eCars.CONCEPT_TREES.Location}
        CARDINALITY{1}}
    PROP trafficLevel {TYPE {
        eMobility.eCars.CONCEPT_TREES
        .RoadTrafficLevel}
        CARDINALITY{*}}
  }
}
```

а) вихідний код

```
CONCEPT Street {
  PARENTS {eMobility.eCars.
      CONCEPT_TREES.RoadElement}
  CHILDREN {}
    PROPS {
    PROP beginsAt {TYPE{eMobility.
        eCars.CONCEPT_TREES.Location}
        CARDINALITY{1}}
    PROP endsAt {TYPE{eMobility.eCars
        .CONCEPT_TREES.Location}
        CARDINALITY{1}}
    PROP trafficLevel {TYPE {
        eMobility.eCars.CONCEPT_TREES
        .RoadTrafficLevel}
        CARDINALITY {*}}
  }
}
```

б) згенерований код

```
CONCEPT_ACTION
    FindNearestChargeStation {
  PARENTS {eMobility.eCars.
      CONCEPT_TREES.Action}
  SPEC {
    RETURN {eMobility.eCars.
        CONCEPT_TREES.ChargingStation
        }
    PARAMS {eMobility.eCars.
        CONCEPT_TREES.Location}
  }
}
```

в) вихідний код

```
CONCEPT_ACTION
    FindNearestChargeStation {
  PARENTS {eMobility.eCars.
      CONCEPT_TREES.Action}
  SPEC {
    RETURN {eMobility.eCars.
        CONCEPT_TREES.ChargingStation
        }

    PARAMS {eMobility.eCars.
        CONCEPT_TREES.Location}
  }
}
```

г) згенерований код

Рис. 3.39. Порівняння отриманих кодів

Порівняння згенерованого коду з вихідним кодом показує, що всі функції охоплено.

Висновки до розділу

У цьому розділі представлено процес моделювання варіантів використання пропонованого підходу. Варіанти використання визначаються з точки зору акторів, які взаємодіють із моделлю системи, та слугують для врахування функціональних вимог системи. Детально розглянуто зв’язок між

визначеними варіантами використання та функціональними вимогами, встановленими, наведено діаграми варіантів використання та їх опис.

Виконано моделювання візуального рівня, що базується на підходах, викладених у другому розділі. Функціональні можливості варіантів використання реалізуються через процес моделювання. Процес розробки VL виконується в три ключові етапи:

- Дизайн мета-моделі HKnowLang;
- Дизайн графічного інтерфейсу користувача;
- Конструкція генератора коду.

Графічний інтерфейс користувача є основним інструментом для взаємодії з програмою VL. Мета-модель HKnowLang використовується для генерації базової моделі даних для програми VL. На основі цієї моделі генератор коду створює код HKnowLang, який відповідає специфікації, визначеній користувачем у системі VL.

ВИСНОВКИ

У магістерській роботі проведено дослідження методологій візуального рівня моделювання автономних програмних систем. Зокрема, вивчено підхід MDSE (Model-Driven Systems Engineering), із фокусом на методі SYSMOD, як інструменті розробки візуального рівня (VL - Visual Layer). Було встановлено, що SYSMOD забезпечує раціональний і систематичний підхід до моделювання візуального рівня, дозволяючи спростити керування складністю проєкту та забезпечити зрозумілість його статусу для всіх зацікавлених сторін. Проте, через обмеження існуючих інструментів MDSE, реалізація повного процесу MDSE для розробки виявилася неможливою.

Аналіз доступних інструментів показав, що поточні інструменти MDSE не підтримують розробку автономних програмних систем, які б використовували підхід, заснований на моделях. У зв'язку з цим запропоновано гібридний підхід до створення інструментів розробки.

Було виконано графічне представлення конструкції специфікації онтології домену HKnowLang у візуальному рівні. Для цього застосовано процес SYSMOD, що включав детальну специфікацію вимог і моделювання варіантів використання для візуального рівня. На основі вимог було розроблено користувацький інтерфейс за допомогою бібліотеки Java. Таким чином, вдалося створити візуальне представлення для конструкцій специфікації HKnowLang.

Для генерації коду з моделей розроблено спеціальний генератор моделі-коду для кожної специфікації HKnowLang. Під час додавання нової специфікації HKnowLang до полотна в VL відповідний об’єкт створюється та додається до списку об’єктів проєкту. Після збереження проєкту здійснюється доступ до кожного об’єкта зі списку, і дані перетворюються на синтаксис коду HKnowLang.

Таким чином, було досягнуто наступних результатів:

- Реалізовано системний підхід до розробки засобів візуального моделювання для предметно-специфічних мов.

- Створено прототип програмного забезпечення візуального моделювання для підтвердження концепції представлення знань і міркування в автономних транспортних засобах на основі HKnowLang.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Andreas Herrmann, Walter Brenner, and Rupert Stadler. Autonomous driving: how the driverless revolution will change the world. Emerald Group Publishing, 2018.
2. Self-driving vehicles. https://www.government.nl/topics/mobility-public-transport-and-road-safety/self-driving-vehicles.
3. J SAE. 3016: 2014 taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. Society of Automotive Engineers, 2014.
4. BAESIS Automotive BV. http://www.baesis.nl. Accessed: 2021-09-01.
5. KnowLang Framework for Knowledge Representation and Reasoning for Self-Adaptive Systems. http://www.knowlang.engineeringautonomy.com/
6. Emil Vassev. Requirements engineering for self-adaptive systems with are and knowlang. EAI Endorsed Trans. Self Adapt. Syst., 1(1):e6, 2015.
7. Emil Vassev and Mike Hinchey. Knowledge representation for adaptive and self-aware systems. In Software Engineering for Collective Autonomic Systems, pages 221–247. Springer, 2015.
8. Emil Vassev and Mike Hinchey. Knowlang: knowledge representation for self-adaptive systems. Computer, 48(02):81–84, 2015.
9. Emil Vassev, Nicklas Hoch, Henry P Bensler, and Mike Hinchey. Formalizing emobility with knowlang. In Proceedings of the 2014 International C* Conference on Computer Science & Software Engineering, pages 1–8, 2014.
10. ISO 26262-4:2018(en) Road vehicles — Functional safety — Part 4: Product development at the system level. https://www.iso.org/obp /ui/#iso:std:iso:26262:-4:ed-2:v1:en.
11. ISO/PAS 21448:2019(en) Road vehicles — Safety of the intended functionality. https://www.iso.org/obp/ui/#iso:std:iso:pas:21448:ed-1:v1:en.

12. Emil Vassev - Google Академія Emil Vassev. https://scholar.google.com/citations?user=J8j_EZ8AAAAJ.
13. Emil Vassev's Webpage. – Emil Vassev. http://www.vassev.com/
14. Robert N Charette. This car runs on code. IEEE spectrum, 46(3):3, 2009.
15. Ken Arnold, James Gosling, and David Holmes. The Java programming language. Addison Wesley Professional, 2005.
16. Jon Holt. UML for Systems Engineering: watching the wheels, volume 4. IET, 2004.
17. T INCOSE. Systems engineering vision 2020. INCOSE, San Diego, CA,
18. Tim Weilkiens. Systems engineering with SysML/UML: modeling, analysis, design. Elsevier, 2011.
19. Roy S Kalawsky, John O'Brien, Seng Chong, ChiBiu Wong, Haibo Jia, Hongtao Pan, and Philip R Moore. Bridging the gaps in a model-based system engineering workflow by encompassing hardware-in-the-loop simulation. IEEE systems Journal, 7(4):593–605, 2013.
20. Brian London and Piero Miotto. Model-based requirement generation. In 2014 IEEE Aerospace Conference, pages 1–10. IEEE, 2014.
21. Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Panagiotis Katsaros, Konstantinos Mokos, Viet Yen Nguyen, Thomas Noll, Bart Postma, and Marco Roveri. Spacecraft early design validation using formal methods. Reliability engineering & system safety, 132:20–35, 2014.
22. Joe Gregory, Lucy Berthoud, Theo Tryfonas, Alain Rossignol, and Ludovic Faure. The long and winding road: Mbse adoption for functional avionics of spacecraft. Journal of Systems and Software, 160:110453, 2020.
23. Lenny Delligatti. SysML distilled: A brief guide to the systems modeling language. Addison-Wesley, 2013.
24. Tim Weilkiens. SYSMOD-The systems modeling toolbox-pragmatic MBSE with SysML. Lulu. com, 2016.
25. Sanford Friedenthal, Alan Moore, and Rick Steiner. A practical guide to SysML: the systems modeling language. Morgan Kaufmann, 2014.

26. Martin Fowler. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional, 2004.
27. Santiago P J´acome-Guerrero, Marcelo Ferreira, and Alexandra Corral. Software development tools in model-driven engineering. In 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT), pages 140–148. IEEE, 2017.
28. Alberto Rodrigues Da Silva. Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures, 43:139–155, 2015.
29. Amine El Kouhen, Cedric Dumoulin, S´ebastien Gerard, and Pierre Boulet. Evaluation of modeling tools adaptation. 2012.
30. Emil Vassev. Graphical representation of hknowlang. 2021.
31. Netbeans visual library. http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/overview-summary.html#overview.description.
32. Java swing library. https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.
33. Emil Vassev, Mike Hinchey, and Benoit Gaudin. Knowledge representation for self-adaptive behavior. In Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering, pages 113–117, 2012
34. Broy, M., & Rumpe, B. (2005). Modelling languages in software engineering. Springer.
35. Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM Computing Surveys (CSUR), 37(4), 316-344.
36. Fowler, M. (2010). Domain-Specific Languages. Addison-Wesley.
37. González-Baixauli, B., et al. (2012). Model-based engineering in the context of automotive functional safety. SAE International Journal of Passenger Cars-Electronic and Electrical Systems, 5(2012-01-0033), 223-234.

38. Jackson, D., & Damon, C. (2012). Software abstractions: logic, language, and analysis. MIT Press.
39. Rajkumar, R., Lee, I., Sha, L., & Stankovic, J. (2010). Cyber-physical systems: the next computing revolution. In Design Automation Conference (DAC), 2010.
40. Fainekos, G., et al. (2009). Temporal logic motion planning for dynamic mobile robots. Automatica, 45(2), 343-352.
41. Schätz, B. (2012). Model-based development of embedded systems. Springer.
42. Alur, R. (2015). Principles of cyber-physical systems. MIT Press.
43. Lunghi, P., Pieri, F., & Velardocchia, M. (2018). Towards the autonomous vehicle: Research and development perspectives. Springer.
44. Cetinkaya, D., & Verbraeck, A. (2011). Model-driven development of complex systems: A visual approach. Simulation Modelling Practice and Theory, 19(1), 168-182.
45. Qi, J., Sun, J., Dong, J. S., & Liu, Y. (2010). Towards model checking real-time properties for complex systems. Theoretical Computer Science, 412(20), 2021-2036.
46. Murru, N., et al. (2019). A formal framework for hybrid knowledge representation and reasoning. Knowledge-Based Systems, 164, 257-267.
47. Borg, J., & Pace, G. J. (2012). A DSL for autonomous vehicle verification. Journal of Software Engineering and Applications, 5(12), 915-923.
48. Ebert, J., & Engels, G. (2005). Graph-based modelling languages and their visual representation. Software & Systems Modeling, 4(2), 175-188.
49. Masiero, R., Oliveira, P., & Ruggiero, W. (2018). Model-driven development of real-time systems: Visual languages and methodologies. Springer.
50. Johnson, R., & Wren, D. (2016). Advances in Model-Driven Engineering for Cyber-Physical Systems. IEEE Transactions on Cyber-Physical Systems, 2(3), 45-57.

51. Stepanenko, A., & Sukhanov, S. (2021). Towards visual DSLs for robotic systems. Robotics and Autonomous Systems, 98, 132-148.
52. Wei, Z., Zhang, F., & Zhao, J. (2018). Model-based design for safety-critical autonomous systems. International Journal of Advanced Robotic Systems, 15(3), 1-15.
53. Sangiovanni-Vincentelli, A., & Martin, G. (2001). Platform-based design and software design methodology for automotive systems. IEEE Design & Test of Computers, 18(6), 23-33