КВАЛІФІКАЦІЙНА РОБОТА

Група МІПЗс-22

Сак В.І.

2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Сак Владислав Іванович

УДК 004.4

Імплементація формальних моделей оптимізації методів побудови адаптивних розподілених систем

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації магістра

Нормоконтроль

_____Стисло О.В.

(підпис, дата, розшифрування підпису)

Студент

_____ Сак В.І.

(підпис, дата, розшифрування підпису)

Допускається до захисту

Завідувач кафедри

_____к.т.н., доц. Ващишак С.П.

(підпис, дата, розшифрування підпису)

Керівник роботи

_____ к.т.н., доц. Демчина М.М.

(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «магістр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« 19 » лютого 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сак Владислава Івановича

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Імплементація формальних моделей оптимізації методів побудови адаптивних розподілених систем

керівник роботи:

Демчина Микола Миколайович, кандидат технічних наук, доцент

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

2. Термін подання студентом роботи 16.02.2024

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

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)

1. Аналіз формальних методів високої адаптивності програмних систем
2. Дослідження формальних методів верифікації рівня адаптивності
3. Побудова формальних шаблонів забезпечення адаптивності рішення
4. Розробка шаблонів специфікації поведінки для моделювання контурів зворотного зв'язку адаптивних систем

5. Дата видачі завдання 29.06.2023

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз формальних методів досягнення високої адаптивності програмних систем	26.09.2023	Виконано
2.	Дослідження формальних методів та моделей верифікації рівня адаптивності	20.10.2023	Виконано
3.	Побудова формальних шаблонів забезпечення адаптивності рішення	15.11.2023	Виконано
4.	Розробка шаблонів специфікації поведінки для моделювання контурів зворотного зв’язку MAPE-K	30.11.2023	Виконано
5.	Формування висновків	09.12.2023	Виконано
6.	Оформлення пояснювальної записки	22.12.2023	Виконано
7.	Оформлення графічного матеріалу та підготовка до захисту роботи	11.01.2024	Виконано

Студент _____ Сак В.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Демчина М.М.
(підпис) (прізвище та ініціали)

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

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
18	Представлення про самоадаптивне програмне забезпечення на основі архітектури	49	Виконання GPS SA
19	Еталонна модель MAPE-K для самоадаптивних систем	49	Ефектор служби GPS
20	Контур керування зворотним зв’язком	50	Відображення процесу для самовідновлення MVD
21	Контур керування зворотним зв’язком для адаптивних систем	50	Зонд вимоги MVD

АНОТАЦІЯ

Кваліфікаційна робота присвячена виконанню імплементації формальних моделей оптимізації методів побудови адаптивних розподілених систем шляхом розробки шаблонів специфікації поведінки для моделювання контурів зворотного зв’язку для реалізації адаптивності програмних рішень.

В першому розділі проаналізовано формальні методи досягнення високої адаптивності розподілених програмних систем. Наведено основні принципи самоадаптації в програмних системах, описана еталонна модель MAPE-K, трирівнева модель архітектури. Виконано дослідження якісних властивостей високої адаптивності програмних систем, наведені формальні методи в процесах адаптації.

В другому розділі наведені формальні методи оптимізації та моделі верифікації рівня адаптивності систем, здійснено опис передумов використання самоадаптивного підходу та техніки MAPE-K. Виконано опис процесу імітаційного моделювання для дослідження надійності мобільного додатку, розглянуто сценарій використання мобільної платформи з елементами високо адаптивності з додаванням самоадаптивного шару. Запропоновані процеси самоадаптації з використанням верифікованих циклів MAPE-K.

В третьому розділі побудовано формальні шаблони забезпечення адаптивності розподіленої системи, розглянуто цільовий домен для застосування шаблонів високо адаптивних систем на прикладі мобільного додатоку та логістичної роботизованої системи. Виконано розробку шаблонів специфікації поведінки для моделювання контурів зворотного зв’язку для реалізації адаптивності.

КЛЮЧОВІ СЛОВА: АДАПТИВНІ СИСТЕМИ, ЧАСОВИЙ АВТОМАТ, АРХІТЕКТУРА, САМОАДАПТИВНИЙ ШАР, МОДЕЛЬ, ФОРМАЛЬНИЙ ШАБЛОН, ВЕРИФІКОВАНИЙ ЦИКЛ.

SUMMARY

The qualification work is devoted to the implementation of formal models of optimization of methods of building adaptive distributed systems through the development of behavior specification templates for modeling feedback loops to implement the adaptability of software solutions.

The first chapter analyzes formal methods of achieving high adaptability of distributed software systems. The main principles of self-adaptation in software systems are presented, the reference model MAPE-K, the three-level architecture model are described. The study of qualitative properties of high adaptability of software systems is carried out, formal methods in adaptation processes are given.

In the second section, formal methods of optimization and models for verification of the level of adaptability of systems are presented, the prerequisites for using the self-adaptive approach and the MAPE-K technique are described. The description of the process of simulation modeling for the study of the reliability of the mobile application is carried out, the scenario of using a mobile platform with elements of high adaptability with the addition of a self-adaptive layer is considered. Proposed self-adaptation processes using verified MAPE-K cycles.

In the third section, formal templates for ensuring the adaptability of a distributed system are built, the target domain for the application of templates of highly adaptive systems is considered, using the example of a mobile application and a logistic robotic system. Developed behavior specification templates for modeling feedback loops to implement adaptability.

KEY WORDS: ADAPTIVE SYSTEMS, TIME AUTOMATIC, ARCHITECTURE, SELF-ADAPTIVE LAYER, MODEL, FORMAL TEMPLATE, VERIFIED CYCLE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MAPE-K - Monitor-Analyze-Plan-Execute

TA - Timed Automaton

TCTL - Timed Computational Tree Logic

MVD - Mobile Virtual Device

SA - Self-adaptation

ІС – інформаційна система

ВСТУП

Актуальність дослідження. З середини 90-х років актуальною є тема системної адаптивності, що широко досліджується, і наукове співтовариство докладає значних зусиль для пошуку нових підходів до з’ясування основних принципів теорії та практики самоадаптації. За останнє десятиліття важливість самоадаптації зросла та отримує все більше визнання в різних сферах застосування і пов'язаних технологіях. Розробка програмного забезпечення та суміжні теми є загальними елементами успішного впровадження самоадаптації. Отже, є нагальна необхідність систематичного дослідження підходів програмної інженерії для розробки самоадаптивних систем, які в ідеалі були б застосовні в багатьох доменах. Сучасні вбудовані системи програмного забезпечення повинні виконувати задачі в різних сценаріях. Ці системи все частіше очікуються функціональними залежно від мінливості середовищ і реагування на зміни всередині системи. Самоадаптивні вбудовані системи повинні приймати рішення щодо адаптивності під час виконання мінливих вимог.

Самоадаптивне програмне забезпечення здатне оцінювати та змінювати власну поведінку і під час оцінки показує, що програмне забезпечення не виконує те, що було задумано робити, або коли є можливі кращі функціональні можливості чи продуктивність. Штучний інтелект являє собою ефективний спосіб імітації адаптивності, високоорганізовані ефективні системи легше переконфігурувати та вони є більш адаптивні. Головна мета дослідження — переглянути останні досягнення самоадаптації з точки зору інформатики та кібернетики. Його аналіз і компіляція інформації з літератури визначає ключові науково-технічні проблеми та оцінює їх передові теорії, методи та прийоми проектування та розвитку самоадаптивних систем.

Самоадаптивність – це здатність системи до коригування своєї поведінки, відповідно до навколишнього середовища. Властивість вказує на

те, що системи самостійно приймають рішення (тобто з мінімальним втручанням або відсутністю такого), як адаптуватися або організувати себе так, щоб вони могли пристосуватися до змін у своєму контексті та середовищі. Хоча деякі самоадаптивні системи можуть це зробити без будь-якого втручання людини, керівництво у формі цілі вищого рівня (наприклад, через сценарії) є корисними та реалізованим в багатьох системах.

Мета і завдання дослідження. Метою дослідження є розроблення шаблону, в який вбудовано результати аналізу адаптивних розподілених систем, запитів користувачів з використанням штучного інтелекту. Дане дослідження спрямоване на досягнення наступних завдань:

- виконати аналіз формальних методів досягнення високої адаптивності програмних систем;
- представити формальні методи та моделі верифікації рівня адаптивності;
- дослідити якісні властивості високої адаптивності програмних систем;
- виконати побудову формальних шаблонів забезпечення адаптивності рішення;
- розробити шаблони специфікації поведінки для моделювання контурів зворотного зв’язку для реалізації адаптивності.

Об’єктом дослідження є процес аналізу і розробки запитів за допомогою методів та моделі оптимізації рівня адаптивності розподілених програмних систем.

Предметом дослідження є багаторівнева модель для оптимізації та спрямування пошуку необхідної інформації.

Методи дослідження базуються на використані методів побудови шаблонів дизайну, методів та моделі специфікації доменів властивостей, методів та моделі побудови шаблонів само адаптивності.

Наукова новизна одержаних результатів полягає в тому, щоб отримати подальший розвиток методів створення шаблонів та моделі

оптимізації, які дають змогу за допомогою штучного інтелекту здійснювати аналіз запиту користувача для конкретизації запитання. Це поширить розроблену модель на існуючі та майбутні створені системи обробки запитів і оптимізує їх.

Практичне значення одержаних результатів полягає в можливості впровадження самоадаптивних програмових системи у нових областях та використовувати інструменти, пов'язані з циклами зворотного зв'язку. Це робиться з метою комбінування концепцій та методів з технік керування та їх застосування до розподілених самоадаптивних систем з циклами управління зворотнім зв'язком.

Апробація результатів дослідження. Матеріали дослідження було представлено у матеріалах І Всеукраїнської науково-практичної інтернет конференції “ІТ екосистема: цифровізація бізнес-процесів в умовах війни”, у тезах доповіді “Застосування grid-сервісів для задач розподілення ресурсів”.

Структура. Кількість розділів – 3. Загальний обсяг основної частини – 83 сторінки. Список використаних джерел містить – 53 позиції.

РОЗДІЛ 1. АНАЛІЗ ФОРМАЛЬНИХ МЕТОДІВ ДОСЯГНЕННЯ ВИСОКОЇ АДАПТИВНОСТІ РОЗПОДІЛЕНИХ ПРОГРАМНИХ СИСТЕМ

1.1 Дослідження предметної області застосування високо адаптивних програмних систем

Використання систем, які залежать від програмного забезпечення, зростає зі швидкістю, яку раніше неможливо було уявити. Один великий клас таких систем характеризується тим, що вони розповсюджуються за допомогою мобільних пристроїв або компонентів, які можуть приходити та йти за бажанням. Деякими прикладами є нові мобільні системи навчання, де мобільні пристрої використовуються для спільної педагогічної діяльності; автономні роботи для транспортування та управління складом; розумні будинки, обладнані датчиками та приводами для надання щоденних домашніх послуг; та системи електронної охорони здоров'я.

Ці приклади мають деякі спільні риси. Вони складаються з кількох одиниць або вузлів; вузли мають чітке положення в середовищі, яке може бути динамічним, оскільки вузли можуть бути мобільними; вузли в цих системах мають постійний (або періодичний) комунікаційний доступ, який необхідний для реалізації цілей розподіленої системи; завдяки характеристикам цих систем, вони знаходяться в динамічних середовищах.

Динамічне середовище та умови експлуатації, що змінюються, можуть призвести до небажаної поведінки таких систем. Деякі приклади такої небажаної поведінки можуть виникати через збій певних ресурсів або вузлів у системі (таких як батареї живлення та частини вбудованих систем) і погіршення якості послуг, що надаються ресурсами (такими як веб-сервіси, точність служби геолокації та підключення до мережі). За таких обставин стає необхідним запровадити механізми пом’якшення. Самоадаптація є

загальновизнаним і ефективним підходом до роботи зі зростаючою складністю та динамічністю сучасних програмних систем. Самоадаптація відбувається за принципом поділу інтересів. В ідеалі проблемами домену керує спеціальний додаток , також відомий як керована система. Механізм адаптації, також відомий як система керування, розширює можливості, пропоновані керованою системою, щоб забезпечити деякі проблеми якості, такі як продуктивність, надійність і відкритість, які є бажаними в системі. Самоадаптивні системи вимагають уявлень про систему та її оточення, щоб міркувати про поведінку адаптації. Один із широко використовуваних методів застосовує моделі архітектури, які забезпечують абстракцію керованої системи, щоб підтримати міркування щодо самоадаптації, що виконується в системі керування. Це відомо як самоадаптація на основі архітектури. Одним із добре визнаних підходів до реалізації самоадаптації на основі архітектури є цикли зворотного зв’язку MAPE-K (монітор-аналіз-план-виконання) [1]. Цикл зворотного зв’язку MAPE-K складається з чотирьох окремих компонентів, які розділяють такі функції: моніторинг керованої системи і її середовище, аналізуючи поведінку керованої системи, плануючи адаптаційні дії для усунення виявленої небажаної поведінки системи та виконуючи заплановані адаптації архітектури для адаптації керованої системи. Іншими відомими підходами, заснованими на архітектурі, для проектування самоадаптивних систем є модель 3-рівневої архітектури [2] і контури керування зі зворотним зв'язком [3]. Модель 3-рівневої архітектури розділяє цілі, алгоритми змін і керування компонентами на трьох рівнях, як підхід до самоадаптації структури через поділ проблем. Пропонується багаторазову архітектурну структуру для побудови самоадаптивних систем. Архітектурний рівень, який займається самоадаптацією, нагадує схожість із циклом MAPE. Проте контроль адаптації централізований. Контури зворотного зв’язку керування [4] — це програмні підходи, натхненні принципами біології. Механізм контролю для самоадаптації живиться зворотним зв’язком від системи, який

може бути позитивним або негативним, незалежно від того, підсилює він зміни в системі чи протидіє їм.

Ці підходи забезпечують інженерні рекомендації, які підтримують архітектурний дизайн самоадаптивних систем і створюють значний вплив на самоадаптивну спільноту [5].

Однак надання доказів, які підтверджують, що цілі адаптації досягаються за допомогою конкретних підходів до самоадаптації, залишається серйозною проблемою в цій галузі. На даний час у центрі обговорення є гарантії для самоадаптивних систем [6]: «Одним із ключових аспектів самоадаптивних систем, який створює важливі проблеми, які ще потрібно вирішити глибоко, є гарантії: тобто надання доказів того, що системи задовольняють свої функціональні та нефункціональні вимоги під час роботи».

Для того, щоб забезпечити докази задоволення цілей системи, сучасний стан самоадаптації на основі архітектури підтримує використання формальних методів.

В [6] виконано систематичний огляд літератури, щоб вивчити сучасний стан використання формальних методів для самоадаптивних систем. Дослідження показало, що спостерігається збільшення використання формальних методів на місцях. Більшість цих досліджень використовували формальні методи для обґрунтування дизайну самоадаптивних рішень. Лише близько 30% досліджень, які використовують формальні методи, застосовують формалізми, щоб надати докази того, що механізми адаптації можуть досягти бажаних цілей адаптації. Ми виявили інтерес у спільноті до формалізації самоадаптивної поведінки, щоб гарантувати бажані властивості адаптації, навіть якщо це не є широко поширеним. Деякі репрезентативні дослідження [7,13] охоплюють такі аспекти, як техніка структурних перевірок властивостей безпеки; формальні моделі для інтеграції компонентів рефлексивного обчислення, розподіленої координації та циклу зворотного зв’язку; виконання на основі моделі для використання розподілу

ймовірностей різних шляхів виконання системи; марківські моделі та логіка дерева пробабілістичних обчислень для динамічної адаптації для досягнення QoS за допомогою циклів керування MAPE-K.

Документи, вивчені під час систематичного огляду літератури, були зосереджені на одному або кількох із таких дослідницьких зусиль: формалізації та аналізі моделей керованої системи, середовища, в якому це відбувається і цілей, які система ставить перед собою. Однак мало роботи було зроблено щодо надання строгих гарантій щодо самої адаптаційної поведінки через специфікацію самоадаптивної поведінки та властивостей. Як правило, дослідження надають докази самоадаптації за допомогою статистичного аналізу на основі серії експериментів [8]. Незважаючи на статистичні результати, ці підходи не можуть чітко гарантувати самоадаптацію.

Крім того, шляхом систематичного огляду літератури ми спостерігали незначне повторне використання для визначення формальних моделей поведінки та властивостей для конкретних самоадаптивних систем. Немає систематичної консолідації щодо розробки формальних моделей для самоадаптивних систем, тому їх можна застосовувати для повторного використання в майбутньому. Дослідники продемонстрували цінність багаторазово використовуваних шаблонів для специфікації та перевірки формальних моделей, наприклад, в [9] формально визначають багаторазово використовувані шаблони, а в [10] визначають багаторазово використовувані властивості для специфікації імовірнісних вимог. В роботі [11] зазначають, що визначення таких багаторазових шаблонів є важливою дослідницькою проблемою для самоадаптивних систем.

1.2 Основні принципи самоадаптації в програмних системах

Самоадаптивні програмні системи виникли через потребу мати справу зі зростаючою складністю систем, які мають стикатися з аспектами,

невизначеними під час проектування. Прикладами невизначеностей є динамічне середовище, в якому знаходиться система, майбутні потреби нових користувачів, підсистеми, які з’являються та йдуть за бажанням, внутрішні збої системи тощо. Ці невизначеності ставлять під сумнів розробку програмного забезпечення щодо забезпечення бажаних якісних властивостей, таких як продуктивність, надійність, відкритість, безпека та ін.

Самоадаптація заснована на проектному принципі поділу інтересів. Мета самоадаптації полягає в тому, щоб відокремити логіку, яка має справу з певними проблемами якості, що представляють інтерес, за допомогою механізму адаптації (так званої системи керування) від функціональності предметної області, наданої базовою програмою, специфічною для домену (керована система).

За останній час спільнота розробників програмного забезпечення привернула увагу до дослідження самоадаптивних систем і в результаті з'явилася низка підходів до проектування таких систем. Два недавніх дослідження дорожніх карт [11, 12] аналізують сучасний стан у галузі самоадаптивних систем і визначають відкриті проблеми для майбутніх досліджень. Двома особливими проблемами, які безпосередньо стосуються дослідження цієї дисертації, є розробка самоадаптивних систем та гарантії для самоадаптивних систем.

Що стосується інженерних аспектів, можна виділити два відомих підходи до самоадаптації. Перший підхід – самоадаптація на основі архітектури [13, 14]. Цей підхід зосереджений на компонентах програмного забезпечення для циклів зворотного зв’язку, моделях і функціях для адаптації та точках зв’язку між керованою та керуючою системами. Другий підхід – самоадаптація на основі контролю [15]. Цей підхід застосовує принципи теорії керування для проектування та аналізу контурів зворотного зв’язку для реалізації самоадаптації. Наша увага зосереджена на самоадаптації на основі архітектури.

Нижче ми коротко представляємо ряд відомих концептуальних архітектур для проектування та реалізації самоадаптивних систем.

1.2.1. Самоадаптація на основі архітектури

В роботі [16] введено термін самоадаптація на основі архітектури та представивлено основні принципи. Автори визнали необхідність створення програмних систем, які здатні автономно модифікуватися під час виконання, без необхідності перезавантаження.

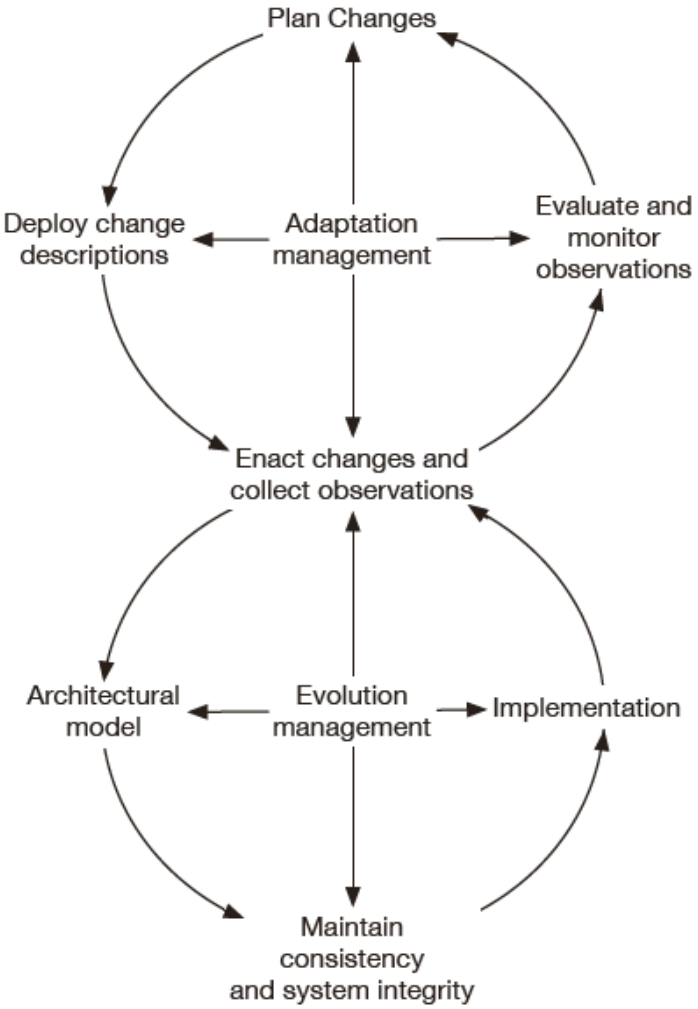


Рисунок 1.1 – Представлення про самоадаптивне програмне забезпечення на основі архітектури

Автори визначають два взаємодіючі життєві цикли, які відокремлюють проблеми управління адаптацією від проблем управління еволюцією (рис. 1.1). Життєвий цикл управління адаптацією відстежує та оцінює поведінку програми під час виконання та генерує плани впровадження адаптацій, коли це необхідно. Управління еволюцією «фокусується на механізмах, що використовуються для зміни прикладного програмного забезпечення» під час еволюції системи.

1.2.2. Еталонна модель MAPE-K

В 2003 році IBM [17] представила еталонну модель MAPE-K, яка була додатково вдосконалена в 2006 році. MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) містить п’ять компонентів, необхідних системі для самоадаптації.

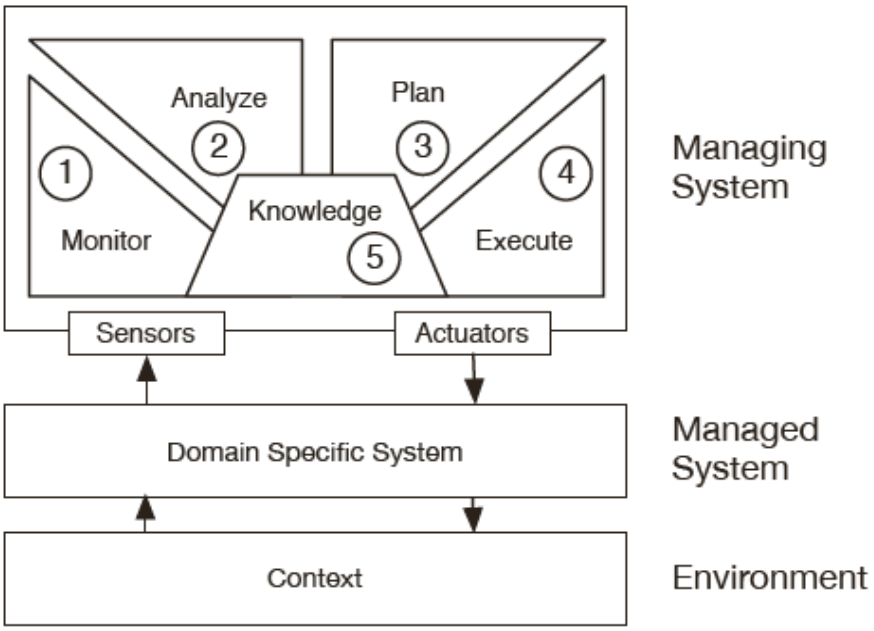


Рисунок 1.2 – Еталонна модель MAPE-K для самоадаптивних систем

Компонент моніторингу (позначений як 1 на рисунку 1.2) отримує інформацію від системи і середовища, у якому він працює, а також оновлює

знання, які передаються до компонента Analyze (2). Базуючись на базі знань (5), компонент Analyze відповідає за визначення правильності стану керованої системи щодо однієї (або кількох) бажаної цілі якості. Згодом компонент «Plan» (3) складає план, який містить дії і компонент Execute (4) виконує заплановані зміни в керованій системі з метою адаптації.

1.2.3. Контур управління зворотним зв'язком

Контур керування зі зворотним зв’язком [32] (позначений пунктирною лінією на рисунку 1.3) базується на принципах теорії керування.

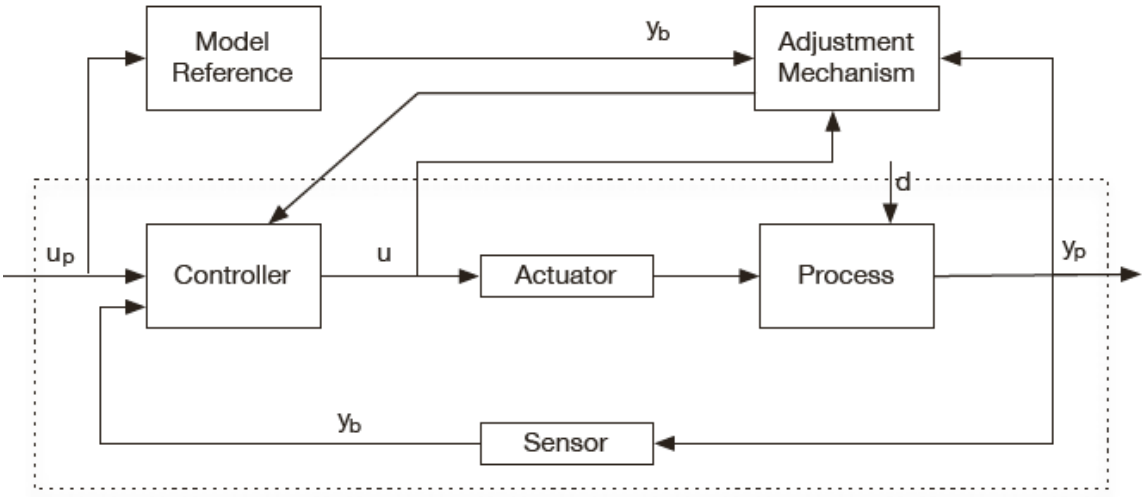


Рисунок 1.3 – Контур керування зворотним зв’язком

У теорії управління вихідні дані системи повертаються в систему, щоб змінити поведінку системи (через контролер). Контур керування зворотним зв'язком розрізняє процеси, що виконуються і оточення. Метою дизайну на основі контуру керування зворотним зв’язком є посилення видимості контуру керування на етапах проектування та розробки адаптивних систем для полегшення аналізу. Різними вимірами аналізу є системні вимоги, адаптивний дизайн і реалізація. Крім того, він надає механізми для перевірки моделі системи. Для цього додаткові компоненти (наведені на рисунку як еталонний механізм моделі та механізм налаштування) інтегровані в контур керування,

щоб змінити поведінку контролера. У системі, заснованій на підході контуру керування зворотним зв’язком, система отримує набір вхідних даних u_p і визначає необхідні адаптації в системі для досягнення бажаних властивостей системи y_p , незважаючи на потенційні збурення d. Механізм налаштування працює з інформацією, що описує поведінку системи (посилання на модель), дії контролера u та поточні якості системи y p.

Розглянемо архітектуру для моделювання архітектур управління зі зворотним зв'язком. Модель, показана на рисунку 1.4, використовує датчики для збору поточних даних з керованої системи (виконавча система) та середовища (операційне середовище), щоб створити оновлену модель поточного стану системи та навколишнього середовища, а також передбачення майбутнього стану. На основі моделей і враховуючи набір цілей системи, набір коригувальних дій планується та передається (ефект) назад у систему.

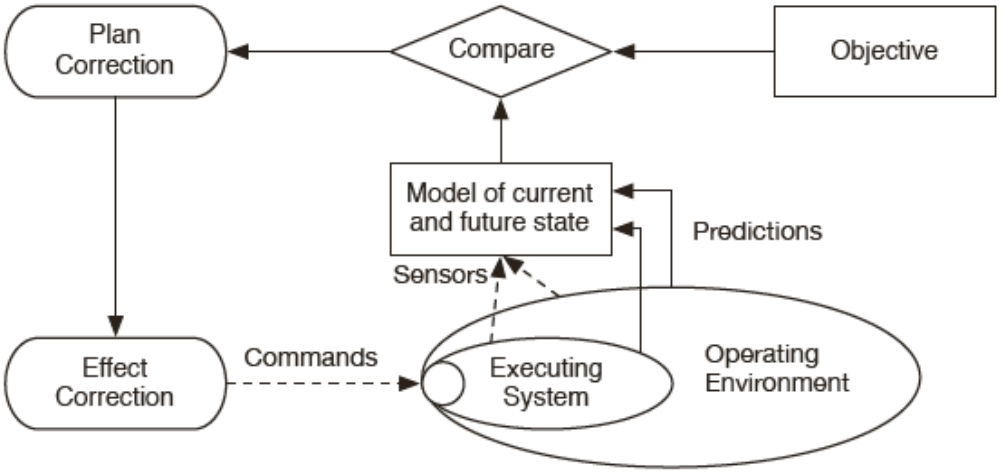


Рисунок 1.4 – Контур керування зворотним зв’язком для адаптивних систем

1.2.4. Структура “Rainbow” для адаптивних систем

Розглянемо структуру для самоадаптивних систем (рис. 1.5). Компоненти в Rainbow реалізують цикл зворотного зв’язку, який тісно пов’язаний з еталонною моделлю MAPE-K. Rainbow розділяє керовану та

керовану систему на два рівні, які називаються рівнем архітектури та рівнем системи відповідно. Архітектурна модель системи (Менеджер моделей) є частиною Rainbow і допомагає системі обґрунтовувати поточну поведінку системи (через засіб оцінки обмежень), щоб планувати (через механізм адаптації) і виконувати (виконавець адаптації) необхідні дії в система.

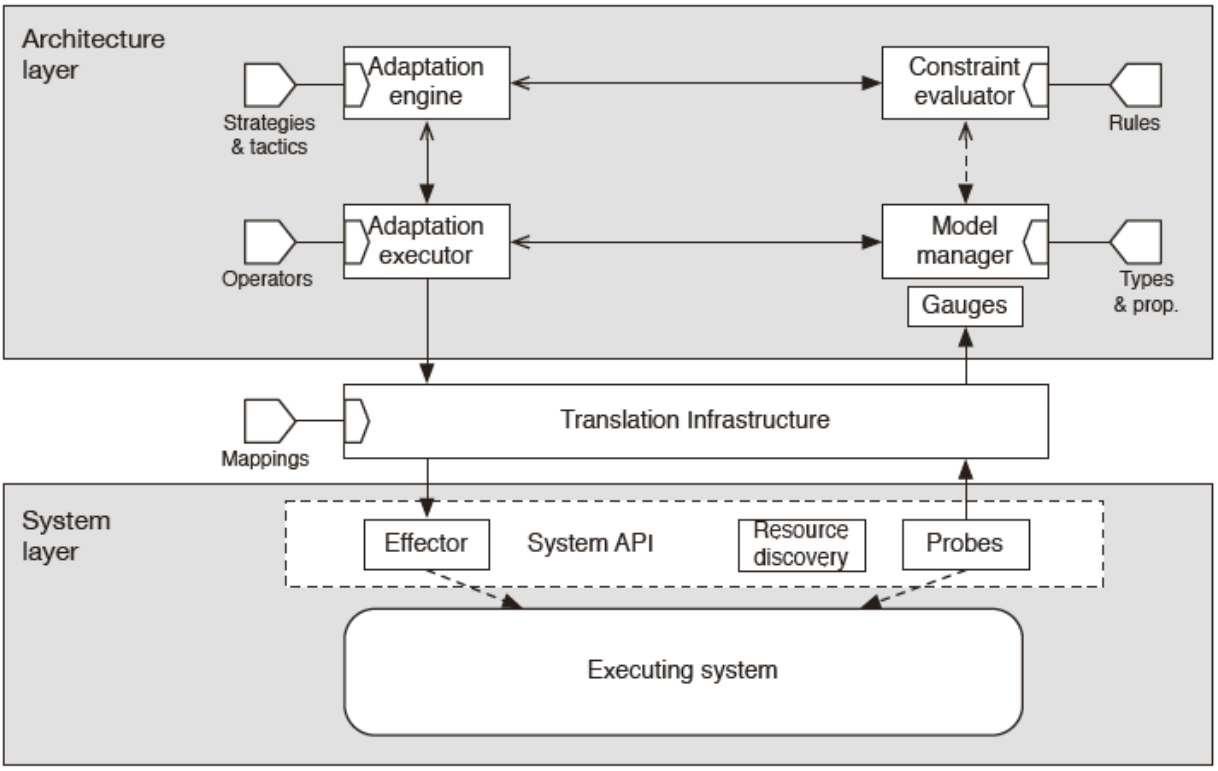


Рисунок 1.5 – Фреймворк Rainbow для самоадаптивних систем

Поєднання цих гарячих точок реалізує цикл самоадаптації. Rainbow підтримує моніторинг і адаптацію програмних систем, які розподілені в мережі. Проте контроль адаптації централізований.

1.2.5. Трирівнева модель архітектури

Ґрунтуючись на роботах [22, 23] було представлено структурний підхід до проектування самоадаптивних систем, який виступає за розділення цілей, змін і компонентів. Модель забезпечує високий рівень абстракції з точки зору

залучених компонентів, а не забезпечує логічну поведінку, яку можна знайти на кожному рівні. Це не слабкість моделі, навпаки, це її сильна сторона. Цей підхід пропонує належний рівень абстракції та загальності для проектування самоадаптивних систем.

Відповідно до їхнього підходу (рис. 1.6), рівень керування компонентами несе відповідальність за надання функціональних можливостей прикладної програми, специфічної для домену. На середньому рівні система відповідає за застосування планів адаптації на рівні компонентів на основі набору алгоритмів плану. Ці плани запускаються (за потреби) у відповідь на події на нижньому шарі. Верхній рівень, управління цілями, відповідає за підтримку набору планів, присутніх на нижньому рівні, щоб досягти набору цілей, які можуть динамічно змінюватися під час виконання.

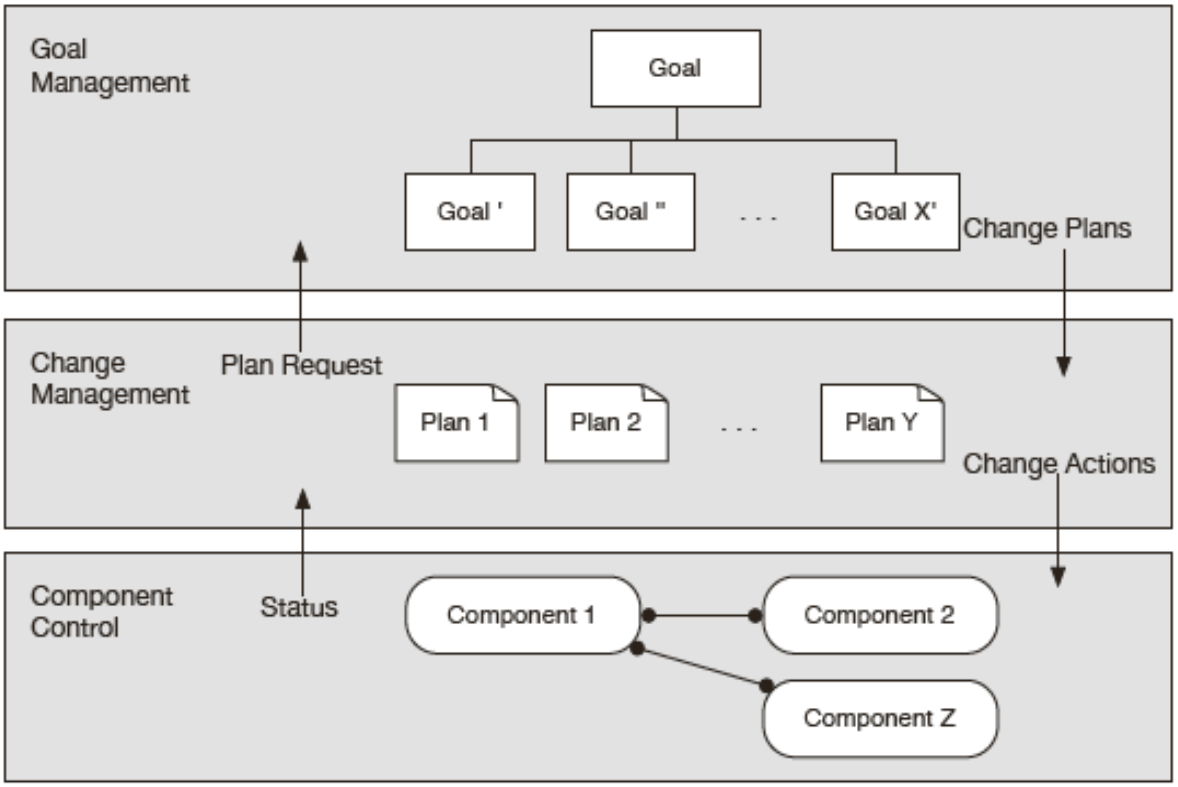


Рисунок 1.6 – Модель трирівневої архітектури

Представлені вище дослідження пропонують різні підходи до проектування самоадаптивних систем. Однак є певні аспекти, які поділяють ці дослідження.

Одним із важливих спільних знаменників є поділ проблем між керованою та керуючою системами. Цей принцип має на меті зменшити складність системи, дозволяючи розробникам програмного забезпечення зосередитися на конкретних проблемах і зменшити перешкоди, які можуть виникнути, якщо функціональні можливості домену та адаптивна поведінка переплітаються.

По-друге, дослідження пропонують використання рівнів як метод збереження поділу проблем і зменшення кількості точок взаємодії між керованою та керуючою системами.

Нарешті, дослідження включають структури знань, які включають один або більше з наступних аспектів: цілі, середовище, стан керованої системи та стан механізму адаптації. MAPE-K агрегує їх у представленні знань. У підході циклу зворотного зв’язку керування модель робить різницю між ціллю, що представляє цілі системи, та моделями, що пропонують абстракцію оточення та станів керованої системи. У Rainbow перевага знань явно не представлена в дизайні, але вона є частиною гарячих точок, які складають рівень адаптації. Модель трирівневої архітектури ділить моделі на дві частини. Управління цілями містить знання щодо цілей адаптації та стану системи управління; тоді як стан керованої системи та середовища є частиною управління змінами.

1.3 Дослідження якісних властивостей високої адаптивності програмних систем

Самоадаптація спрямована на додавання певних якісних властивостей програмній системі. Тому важливо дати чітке визначення щодо проблем якості, які ми прагнемо розглянути в цій роботі.

Стандартний глосарій термінології програмної інженерії IEEE визначає якість як:

1. Ступінь, до якого система, компонент або процес відповідає визначеним вимогам.

2. Ступінь, до якого система, компонент або процес задовольняє потреби або очікування клієнта чи користувача» [22].

З їх визначення ми розуміємо, що якості пов’язані з конкретними -вимогами, які користувач очікує від системи. У контексті цієї роботи ми прагнемо надати докази того, що певні проблеми щодо якості гарантовані. Конкретно потрібно зосередитися на стійкості (міцності) та відкритості. Тому важливо забезпечити розуміння визначення цих термінів. Нижче ми подаємо огляд концепцій і відповідних зусиль, які зосереджені на них.

Стандартний глосарій термінології програмної інженерії IEEE визначає стійкість як «ступінь, до якого система або компонент можуть правильно функціонувати за наявності недійсних вхідних даних або стресових умов навколишнього середовища».

Стійкість також зазвичай розглядається, коли йдеться про властивості безпеки системи. Конкретно, поведінкова безпека [34] відноситься до бажаної поведінки системи, яка не призведе до збою або зміни на небажану поведінку під час виконання під час протидії помилкам.

Стійкість системи є добре вивченою властивістю якості. Ґрунтуючись на роботі з тестування стійкості [35] було запропоновано підхід до оцінки та надання гарантій щодо надійності системи шляхом визначення відповідних тестів надійності для самоадаптивної системи та зміни поведінки зонда для імітації помилок, які потрібно вивчити. У [36] автори зосереджуються на питаннях надійності та порівнюють різні підходи до самоадаптації в розподілених системах. Конкретно, автори детально описують переваги самоадаптації з використанням мультиагентних систем щодо клієнт-серверного та сервіс-орієнтованого підходів.

Незважаючи на величезну кількість досліджень, забезпечення гарантій надійності системи залишається проблемою, зокрема в контексті самоадаптації. Існує потреба в розробці систем, які «забезпечують певну еластичність, щоб бути стійкими проти певних змін, або спроможні керувати змінами, генеруючи альтернативні рішення [27].

У цьому дослідженні ми вивчаємо стійкість у розподілених самоадаптивних системах. Наша особлива увага приділяється обробці збоїв ресурсів, необхідних для досягнення цілей керованої системи.

Відкритість зазвичай розглядається як тип гнучкості якості програмного забезпечення. Відкритість системи нещодавно стала бажаною властивістю системи завдяки високій динамічності системних середовищ і швидкому розвитку розробки програмного забезпечення. Відкритість було визначено як одну з чотирьох основних характеристик, якими на той час матимуть майбутні програмні системи. Визначено відкритість так: «програмні системи підлягають децентралізованому управлінню та можуть динамічно змінювати свою структуру» [16].

Інший більш загальний погляд на відкриті системи представлений в [34]: «системи програмного забезпечення є відкритими, якщо вони спеціально створені для забезпечення розширень. Відкриті системи зазвичай постачаються зі структурою для полегшення включення розширень». У цьому визначенні автори вважають систему відкритою, якщо вона допускає всілякі розширення, такі як компоненти програмного забезпечення, плагіни тощо, на додаток до фізичних ресурсів, які можуть з’явитися. Ще іншу точку зору висунули в [28]. Автори розглядають здатність системи сприймати нову поведінку, що надходить ззовні, як властивість відкритості для певних самоадаптивних систем.

У цій роботі ми зосереджуємося на проблемах відкритості в розподілених системах, щоб дозволити керувати ресурсами, які приходять і йдуть за бажанням.

Використання самоадаптації для боротьби з відкритістю все ще є проблемою [48]. Відкрита система передбачає високий рівень складності системи, оскільки об’єкти в системі повинні бути готові дозволити нові взаємодії з елементами, що надходять. Крім того, відкритість може вплинути на інші питання якості, такі як безпека та продуктивність. Таким чином, дуже актуальним є створення та дотримання належної практики для проектування відкритих самоадаптивних систем, які дозволяють їх аналіз.

В [31] виконали тематичне дослідження з метою вивчення переваг архітектурного підходу MAPE-K для забезпечення надійності та відкритості систем. MetaSelf [36] — це архітектура програмного забезпечення, заснована на циклах керування зворотним зв’язком, що дозволяє, серед інших цілей, відкритість системи через правила самоорганізації та надійності. В [37] детально описали переваги використання мультиагентних систем, як слабко пов’язаних рішень, щоб запропонувати відкритість системи для адаптації до мінливих потреб бізнесу. В [38] провели опитування щодо сфери самоадаптації та детально розповіли про використання систем, що самоорганізуються, щоб запропонувати, серед іншого, якість відкритості для системи. Автори в [39] представили самоадаптивне рішення на основі агента для пошуку, включення та спільного використання ресурсів у платформі з метою покращення продуктивності системи. У своєму дослідженні автори порівнюють це рішення з попередніми агентними рішеннями та заявляють про відповідні покращення порівняно з попередніми агентними дослідженнями. Однак це дослідження не застосовує поділ інтересів через самоадаптивний рівень, але інтегрує логіку самоадаптації в поведінку агента. Ця тенденція досить поширена в спільноті автономних агентів.

У своїй роботі ми працюємо над відкритістю через самоадаптацію, дотримуючись підходу на основі архітектури MAPE-K, щоб керувати складністю системи. Основною причиною, яка спонукає до впровадження самоадаптивного рівня в систему, є забезпечення бажаних якостей системи. Однак розробка самоадаптації для складних систем, таких як розподілені та

високоінтенсивні системи обробки, збільшує складність у розробці самоадаптивних рішень, оскільки адаптація може вимагати залучення багатьох вузлів та взаємодії їхньої поведінки. Як повторний у цьому випадку все ще є складним завданням забезпечити гарантії того, що в системі досягнуто бажаних властивостей якості [48].

У певних сферах, таких як електронна охорона здоров'я, транспортний транспорт та інші критично важливі для безпеки системи, дуже важливо мати можливість адаптувати та виправляти систему, коли трапляються збої, але так само важливо надати гарантії, що адаптації виконано правильно та що система задовольняє бажані цілі якості. Одним із загальноприйнятих підходів до надання гарантій щодо адаптації є використання формальних методів.

1.4 Формальні методи в процесах адаптації

Формальними методами ми називаємо методи проектування, які мають математичне підкріплення. Основна перевага застосування формальних методів полягає в тому, що вони надають засоби для точної специфікації поведінки програмного забезпечення та перевірки бажаних властивостей програмного забезпечення. З цієї причини формальні методи зазвичай використовуються для моделювання та міркування щодо поведінки систем, а також для перевірки моделі та підтвердження властивостей окремих систем [18]. Під час проектування зазвичай використовуються формальні моделі. Однак останнім часом зростає кількість досліджень, присвячених дослідженню застосування формальних методів під час виконання, щоб фіксувати поведінку систем і динамічно генерувати моделі систем для міркувань, наприклад [37, 38]. Поведінка систем може бути задана різними формальними мовами. Перехідні системи та автомати зазвичай використовуються спільнотою розробників програмного забезпечення для проектування самоадаптивних систем. У наших дослідженнях ми використовуємо часові автомати (TA - Timed Automaton) як мову для

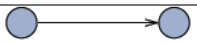
визначення поведінки системи, оскільки графічні представлення та переходи станів легко зрозумілі людям [47]. Ми використовуємо логіку дерева обчислень із синхронізацією (TCTL - Timed Computational Tree Logic) як формальну мову для визначення властивостей системи . Нижче ми пропонуємо огляд мов TA і TCTL.

1.4.1. Часові автомати

Часовий автомат (TA) — це кінцевий автомат, розширений часовими змінними, який моделює поведінку. Часові змінні використовуються для синхронізації поведінки. Крім того, автомати можуть спілкуватися через канали, де поведінка відправника x! синхронізується з поведінкою приймача x?. Формально TA визначається як кортеж

Таблиця 1.1

Умовні позначення в фігурах часових автоматів

Figure	Name	Description
	State	States of behaviors are represented by circles and (optionally) annotated in red on top of the associated *state*.
	Committed State	Committed states are represented with circles containing the *c* character. Committed states must be left without time consumption.
	Urgent State	Urgent states of behaviors are represented with circles containing the *u* character. Urgent states must be left as soon as exiting conditions are found (normally defined by conditions on outgoing transitions).
	Initial State	Initial states of behaviors are represented by double-lined circles. There must be one unique *Initial State* per automaton, specifying the behavior state when the system starts.
	Invariants	Invariants that need to be satisfied in certain states are annotated in purple under the related *state*.
	Transition	Transitions between two states are represented by directional arrows, showing the origin and destination of the transition.
	Conditions	Conditions to enable firing of transitions between states are annotated in green under the related *transition*.
	Signal	Signals used for communication between behaviors are annotated dark blue over the associated *transition*.
	Function	Functions associated with behaviors are annotated in light blue under the associated *transition*.

У цій роботі ми використовуємо UPPAAL [15] для моделювання ТА. В UPPAAL специфікації поведінки можуть бути доповнені виразами, заданими на С-подібній мові для визначення структур даних (концепції структури) і функцій. Ми дотримуємося умовних позначень UPPAAL для опису шаблонів поведінки, представлених у таблиці 1.1.

У автоматах із синхронізацією переходи між станами можуть здійснюватися за допомогою запуску події або часу. За допомогою механізму ініціювання подій один автомат запускає інший за допомогою сигналу, надісланого через канал. Цей випадок проілюстровано на рисунку 1.7 ліворуч, де автомат із поведінкою B1 запускає сигнал , щоб ініціювати перехід на поведінку B2. У цьому випадку ми говоримо, що друга поведінка B2 залежить від B1, оскільки вона не зможе виконувати переходи без відповідного сигналу від першої. За бажанням дані, створені в поведінці B1 , можуть бути передані в поведінку B2 за допомогою сховища знань.

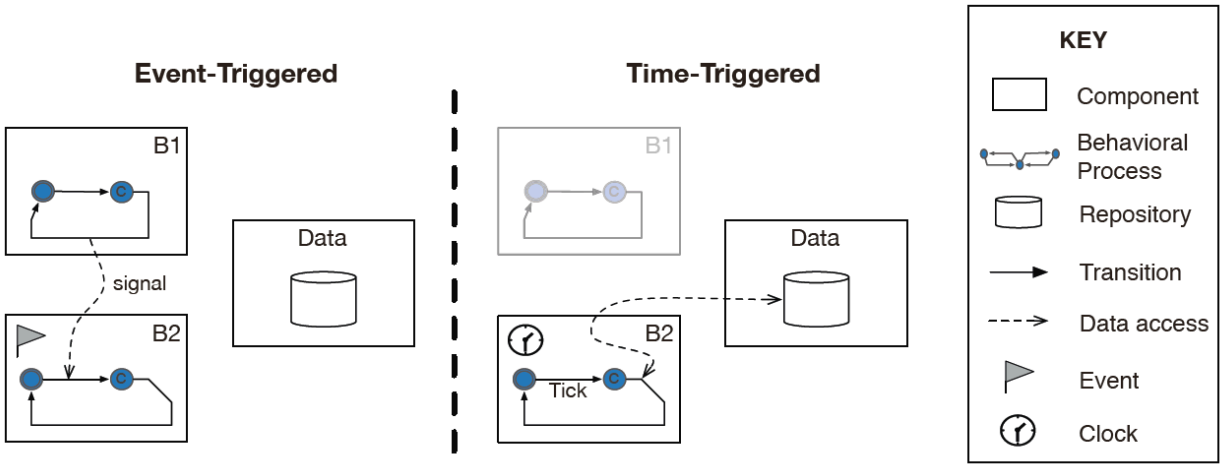


Рисунок 1.7 – Події (ліворуч) і час (праворуч) ініціювання підходів для компонента

З механізмами запуску часу перехід запускається на основі інваріантів стану та умов часу. Цей випадок показано на рисунку 1.7 праворуч, де поведінка B2 автономно виконує переходи в автоматі на основі підходу на основі часу (Tick). Завдяки автономності поведінки щодо іншої поведінки B1,

для запуску часу потрібне сховище даних для зберігання спільних знань щоразу, коли необхідно передавати інформацію між поведінками. Загалом, запуск за часом є менш ефективним з точки зору перевірки (оскільки передбачає більше потоків виконання).

Логіка обчислювального дерева з часовим поясненням (TCTL) — це формальна мова для специфікації властивостей на основі CTL, розширеної за допомогою змінних годинника. Вирази в TCTL описують формули стану та шляху, що дозволяють перевіряти такі властивості, як досяжність (система повинна/може/не може досягти певного стану або станів) і живучість (щось зрештою буде зберігатися) тощо.

Висновки до розділу 1

В даному розділі представлено основні поняття, які лежать в основі дослідження вискоадаптивних систем та їх особливостей. Розглянуто питання принципів самоадаптації, описані цілі якості, які ми націлюємо в цій роботі: надійність і відкритість. Нарешті, зроблено підсумок на основі часових автоматів і логіки дерева часових обчислень, які використовуються як формальні мови в представленому дослідженні.

РОЗДІЛ 2. ФОРМАЛЬНІ МЕТОДИ ОПТИМІЗАЦІЇ ТА МОДЕЛІ ВЕРИФІКАЦІЇ РІВНЯ АДАПТИВНОСТІ СИСТЕМ

2.1 Опис передумов використання самоадаптивного підходу та техніки MAPE-K

У цьому розділі ми вивчаємо використання формальних методів для специфікації та верифікації самоадаптивної програми. Зокрема, ми вивчаємо програму для спільного мобільного навчання, у якій ми застосовуємо самоадаптацію для виконання вимог до надійності. Основна увага приділяється навчальним заняттям, де групи з трьох-чотирьох студентів повинні вимірювати та обчислювати властивості геометричних фігур, таких як кола, прямокутники та трикутники. Для підтримки завдань кожен учень використовує мобільний пристрій із підтримкою GPS, який дозволяє обчислювати відстані. Залежно від діяльності групі можуть знадобитися дві або більше служб GPS, наприклад, три GPS необхідні для тріангуляції. Діяльність відбувається на відкритому повітрі, і через динамічні умови середовища точність GPS не завжди гарантується. А отже, це підриває результати навчальної діяльності. Щоб забезпечити надійність системи, ми додали два цикли MAPE, які мають справу з двома проблемами надійності: перший цикл стосується керування доступністю служби GPS на основі фактичної якості служби GPS; другий цикл займається керуванням необхідною кількістю GPS-сервісів поточного завдання для групи.

Цілями цього прикладу є:

- формально вказати програму мобільного навчання (тобто керовану систему в її середовищі);
- формально визначити поведінку самоадаптації, яка відповідає підходу циклу MAPE-K;

- формально вказати вимоги до стійкості щодо зниження точності GPS;
- перевірити та переконатися, що властивості зберігаються для моделі;
- зіставити моделі проектування з реалізацією.

Цей випадок сприяє відповіді на запитання дослідження:

- Як змоделювати поведінку компонентів MAPE-K, щоб відповідати вимогам надійності та відкритості?
- Які властивості необхідно перевірити, щоб забезпечити докази того, що необхідна поведінка самоадаптації забезпечена?

Щоб гарантувати необхідні вимоги до надійності, ми використовували TA, щоб визначити поведінку циклів MAPE і виразили вимоги до надійності у TCTL.

Ми використали явні модулі для кожної з функцій адаптації циклів MAPE у дизайні самоадаптивної програми та зіставили ці модулі один до одного з реалізацією Java.

Використання явних модулів для кожної функції адаптації циклів MAPE полегшує моделювання самоадаптивної поведінки, оскільки розробник:

- може зосереджуватися на одній діяльності за раз;
- моделювати взаємодію між керованою та керуючою системами, оскільки точки взаємодії чітко визначені;
- пояснювати поведінку всередині та між циклами MAPE в результаті чіткого розподілу проблем;
- міркувати про взаємодію між керованою та керуючою системою;
- вказати необхідні властивості, і ці властивості можна вказати на більш детальному рівні;
- визначити проблеми в дизайні;
- відобразити дизайн для реалізації.

Однак є й деякі компроміси. Деякі дії MAPE мають зрозумілу поведінку, що може викликати питання щодо корисності окремої специфікації; збільшується розмір конструкції; і вартість верифікації зростає. У цьому розділі міститься документація формального дизайну однієї з програм, що лежить в основі формальних шаблонів MAPE-K, які будуть представлені в наступному розділі.

2.2 Опис процесу імітаційного моделювання для дослідження надійності мобільного додатку

Мобільні навчальні програми підтримують традиційні лекції в приміщенні із заходами на свіжому повітрі за допомогою мобільних пристроїв. Прикладом сценарію є команда студентів, які використовують методи тріангуляції для вивчення властивостей геометричних фігур. Самоадаптивний рівень — це набір взаємодіючих циклів MAPE (монітор-аналіз-план-виконання), розподілених по телефонах. Щоб гарантувати вимоги до стійкості, ми формально вказуємо самоадаптивну поведінку за допомогою автоматів із синхронізацією, а необхідні властивості – за допомогою логіки дерева обчислень із синхронізацією. Ми використовуємо інструмент UPPAAL для моделювання самоадаптивної системи та перевірки вимог до надійності. Нарешті, ми обговорюємо, як формальний дизайн підтримував реалізацію самоадаптивного рівня поверх існуючої програми.

Майбутнє покоління програмних систем працюватиме у відкритих середовищах у тому сенсі, що вони лише частково відомі під час розробки. Зростаюча кількість цих систем складатиметься з слабко пов’язаних підсистем, які працюють на поширених пристроях і мережах, що надають послуги, які не є повністю передбачуваними. Одним із нових класів таких систем є мобільні навчальні програми. Мобільні навчальні програми підтримують традиційні лекції в приміщенні з заходами на свіжому повітрі

-за допомогою мобільних пристроїв. Основна увага приділяється групам із трьох-чотирьох учнів, які повинні виміряти та обчислити властивості геометричних фігур, таких як кола, прямі кути та трикутники. Для виконання завдань кожен учень використовує мобільний пристрій із підтримкою GPS. Мобільна навчальна програма надає учням навчальні послуги, які дозволяють їм виконувати завдання, наприклад служби для представлення завдань, вимірювання відстані між вибраними мобільними пристроями на основі їх поточного місцезнаходження GPS тощо.

Мобільна навчальна програма розроблена як розподілена система на основі агентів. На кожному мобільному пристрої розгортається агент пристрою, який забезпечує навчання. Агенти пристроїв групи студентів, які працюють над однаковими завданнями, утворюють організацію, яку ми називаємо мобільним віртуальним пристроєм (MVD - Mobile Virtual Device). MVD має структуру господар-підлеглий. Майстер отримує нові завдання від агента активності, який знаходиться на сервері, розгорнутому в школі, і звітує про результати після виконання завдання. Оскільки якість вимірювань безпосередньо залежить від точності GPS, система повинна гарантувати, що служби GPS забезпечують мінімальний рівень якості відносно точності вимірювань. Через зміну умов навколишнього середовища необхідний рівень точності GPS не завжди гарантується. На жаль, програма мобільного навчання не була розроблена для роботи з недостатньою якістю обслуговування GPS, що перешкоджає використанню програми, коли умови погіршуються.

Самоадаптація (SA - Self-adaptation) є загальновизнаним підходом для роботи з певними якостями часу виконання. SA використовується для додавання так званих самовластивостей (самовідновлення, самозахист, самооптимізація) до систем, що стосуються зміни умов роботи системи або її середовища. SA базується на проектному принципі поділу концернів. Зокрема, SA має на меті відокремити логіку, яка стосується питань якості, що цікавить, від функціональних можливостей домену, які надає базова керована

система. Одним із відомих підходів до реалізації SA є використання циклу зворотного зв'язку MAPE (монітор-аналіз-план-виконання). У той час як багато досліджень застосовували дизайн MAPE для реалізації самоадаптації, більшість із цих досліджень розглядають MAPE як концептуальну структуру, яка спрямовує процес розробки.

У цьому імітаційному експерименті ми показуємо, як ми розширили існуючу мобільну навчальну програму за допомогою рівня самоадаптації, зробивши систему стійкою до зниження точності GPS. Рівень самоадаптації задуманий як набір взаємодіючих циклів MAPE, розподілених по мобільних пристроях. Ми розробили окремі компоненти для різних функцій циклів MAPE, які пропонують переваги з точки зору моделювання, міркування та відображення дизайну в реалізації. Щоб гарантувати вимоги до надійності, ми формально визначаємо самоадаптивну поведінку та перевіряємо вимоги до надійності. Ми використовували офіційно перевірений дизайн для підтримки реалізації самоадаптивного рівня поверх існуючої програми.

2.3 Сценарій використання мобільної платформи з елементами високо адаптивності

У цьому розділі ми надаємо короткий опис мобільної навчальної програми, ми визначаємо проблему надійності, з якою ми зіткнулися через недостатню точність GPS, і ми описуємо, як ми вирішували цю проблему, розширюючи дизайн існуючої системи за допомогою рівня самоадаптації.

2.3.1. Опис платформи

Мобільний навчальний додаток підтримує навчальні, де учні використовують мобільні пристрої з підтримкою GPS. Навчальна діяльність відбувається в контексті лекції і складається з набору завдань (зазвичай від 4 до 8 завдань). Прикладом навчальної діяльності є вимірювання та обчислення

властивостей трикутників, а одне конкретне завдання полягає у використанні методів тріангуляції для знаходження місць на полі за трьома сторонами трикутника, а два з трикутників уже позначені на полі. поле. На рисунку 2.1 показано сценарій використання, де три групи студентів (представлені MVD) виконують завдання навчальної діяльності.

Програма задумана як розподілена система на основі агентів. Агент пристрою, розгорнутий на кожному мобільному телефоні, надає навчальні послуги студенту (місця збору, обчислення відстані тощо). Агенти пристроїв групи, які працюють над однаковими завданнями, утворюють MVD. У MVD один із агентів обирається основним, а інші є підпорядкованими. Наприклад, новий майстер обирається, коли головний телефон розряджається. Майстер спілкується через 4G із сервером за допомогою комунікаційної інфраструктури. Управління завданнями на сервері є відповідальністю Activity Agent. Майстер кожного MVD отримує нові завдання від агента активності на сервері та звітує про результати після виконання завдання.

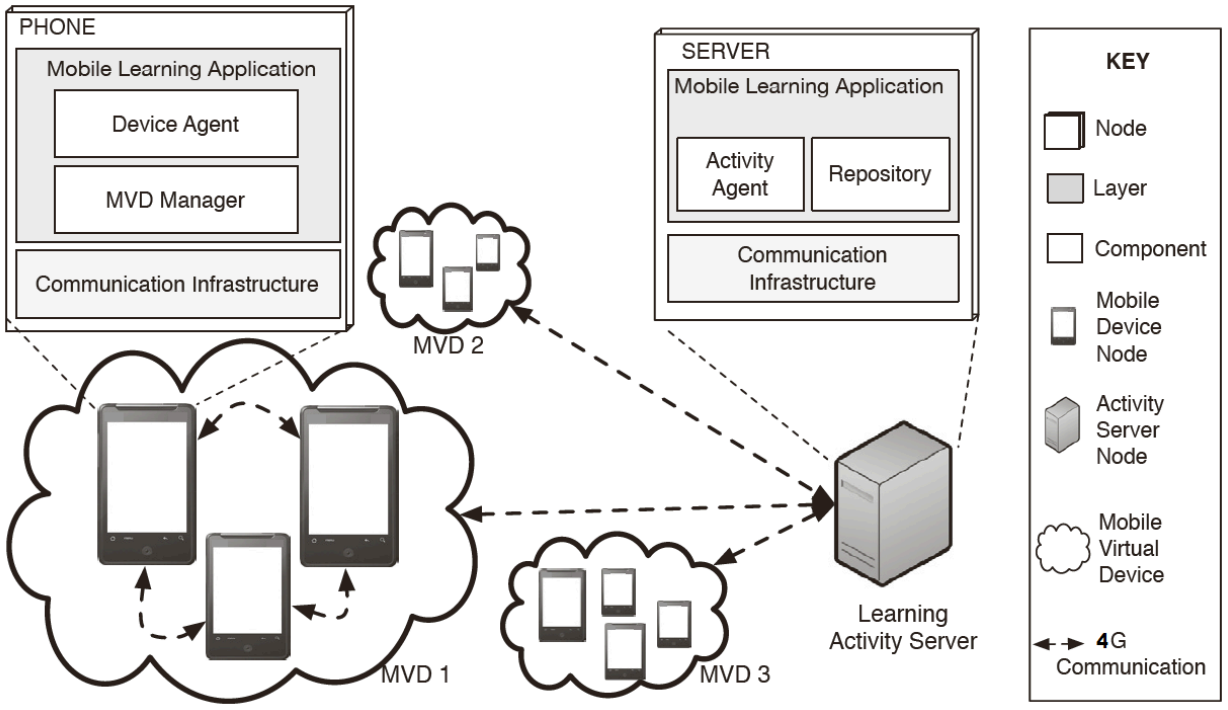


Рисунок 2.1 – Сценарій використання навчальної діяльності

Через зміну умов навколишнього середовища чутливість GPS може змінюватися з часом, що впливає на точність вимірювань і може перешкоджати використанню програми, коли умови погіршуються. Ми знали про те, що GPS надає не завжди точні дані. Однак це виявилося більшою проблемою, ніж наші початкові припущення, аж до того моменту, коли неточні вимірювання вводять студентів в оману.

Є дві основні змінні, які визначають необхідну якість вимірювань GPS під час навчальної діяльності: поточна точність GPS і необхідний рівень точності для поточного завдання. На рисунку 2.2 показано, як похибка точності GPS зазвичай змінюється з часом для мобільного пристрою.

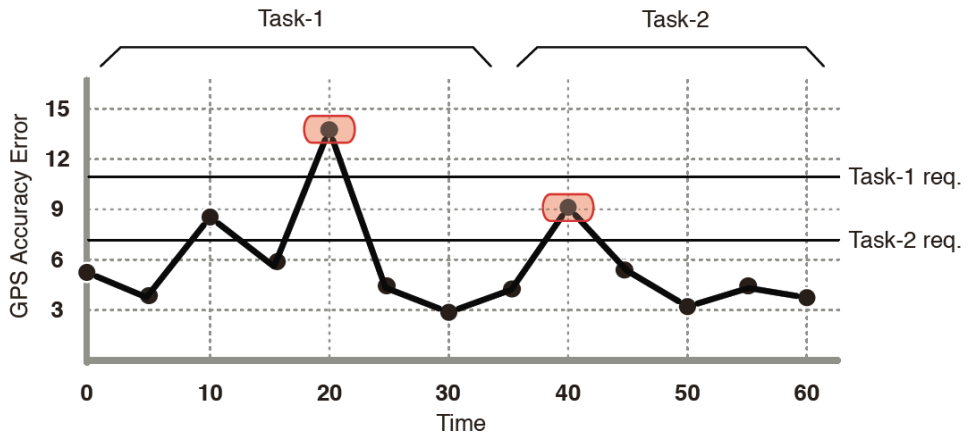


Рисунок 2.2 – Приклад точності GPS

Залежно від поставленого завдання допустимий рівень похибок точності GPS може бути різним. Наприклад, похибка точності 8 метрів у отриманні GPS має більший вплив, якщо використовується для обчислення відстані 20 метрів, ніж коли використовується для відстані 60 метрів. Тому існує потреба в динамічному оновленні необхідного рівня точності GPS для вимірювань кожного завдання. На рисунку 2.2 показано дві горизонтальні лінії, що представляють різні вимоги до точності для Завдання-1 і Завдання-2, де перше завдання вимагає нижчого рівня точності (похибка менше 11 метрів), ніж друге (похибка менше 7 метрів).

Комбінація якості GPS і вимог до певного завдання визначає, чи підходить мобільний пристрій для виконання вимірювань відстані для цього завдання. На рисунку 2.2 ми позначили два моменти часу, що представляють точки, де мобільний пристрій не забезпечує необхідної якості вимірювань. У мітці часу 20 під час виконання Завдання-1 повідомляється про помилку точності 14 метрів, яка не досягає точності 11 метрів, необхідної для завдання. Зауважте, що доступна якість GPS може відповідати вимогам для одного завдання, але не виконуватиме інше завдання (див., наприклад, позначку часу 40). З цього досвіду ми дізналися, що необхідна якість GPS для мобільних пристроїв суттєво відрізняється. Однак ми помітили, що під час навчальної діяльності не більше 20% модулів GPS не змогли забезпечити необхідний рівень якості, і це протягом менш ніж 20% часу навчальної діяльності. Як правило, ми можемо констатувати, що (найгірший випадок на практиці) менше 10% мобільних пристроїв одночасно знаходяться в небажаному стані.

Щоб мати справу з несправними пристроями, нам потрібно врахувати вимоги щодо необхідної кількості пристроїв для виконання завдань. Як пояснювалося раніше, навчальна діяльність складається з набору завдань. Однак для різних завдань може знадобитися різна кількість пристроїв. Наприклад, достатньо двох мобільних пристроїв для вимірювання діаметра кола, тоді як для тріангуляції потрібні три мобільні пристрої. Тому нам потрібно враховувати кількість необхідних GPS-пристроїв на групу (MVD) при роботі з несправними пристроями. Зазвичай доступно від 10 до 20% резервних телефонів.

Підводячи підсумок, можна сказати, що точність GPS телефонів може погіршуватися, роблячи їх недійсними для вимірювання відстані. У результаті кількість мобільних пристроїв у групі може бути недостатньою для успішного виконання завдань. Наразі додаток не підтримує студентів у виявленні недостатньої якості GPS-модуля та вирішенні проблеми шляхом динамічної інтеграції доступних телефонів. Щоб вирішити цю проблему, ми

прагнемо покращити поточну систему за допомогою механізмів самоадаптації, які гарантують надійність системи щодо зниження якості GPS мобільних пристроїв.

2.3.2. Додавання самоадаптивного шару

Щоб реалізувати необхідну надійність, ми додали самоадаптивний шар поверх вихідної системи. Ми реалізували самоадаптивний шар за допомогою циклів MAPE, як показано на рисунку 2.3. Конкретно, щоб забезпечити надійність системи, ми додали два цикли MAPE, які мають справу з двома проблемами надійності: перший цикл має справу з керуванням доступністю служби GPS на основі фактичної якості служби GPS (ліва сторона MAPE на рисунку 2.3).; другий цикл займається керуванням необхідною кількістю GPS-сервісів поточного завдання для MBC (права MAPE на рисунку 2.3).

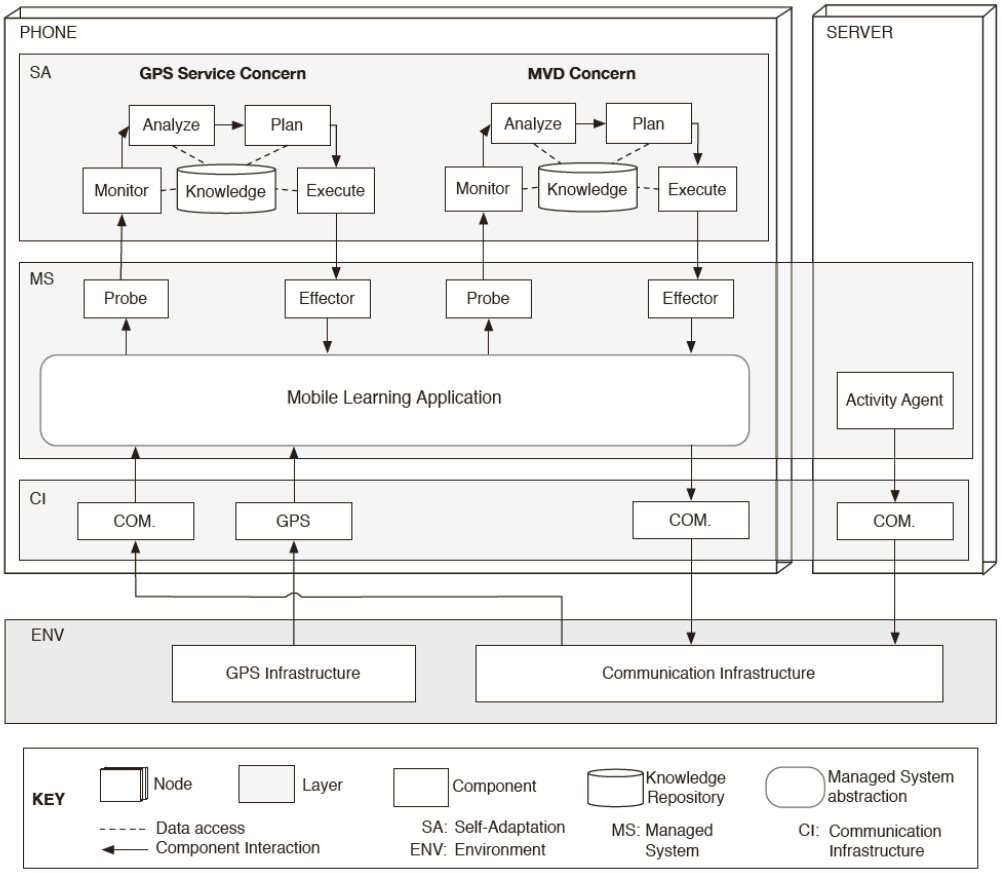


Рисунок 2.3 – Структурний вигляд самоадаптивної системи

Зонди та ефектори дозволяють циклам MAPE збирати релевантну інформацію про базову керовану систему та застосовувати заплановані адаптаційні дії.

Перша петля MAPE (GPS Service Concern) є локальною для кожного мобільного пристрою. Цей шлейф контролює якість GPS-модуля, порівнює її з необхідною якістю і на основі цього активує або вимикає послугу GPS. Коли службу GPS дезактивовано, вона може запустити другий цикл MAPE, щоб почати процес самовідновлення, тобто знайти новий пристрій і додати його до MVD. Кажемо, може спрацьовувати, тому що в MVD можуть бути зайві телефони, тому заміна не потрібна.

Другий шлейф MAPE (Концерн MVD) розподілений по пристроях MVD. У цьому циклі MAPE використовується шаблон головний-підлеглий. На рис. 2.4 показано розподіл компонентів MAPE для трьох телефонів.

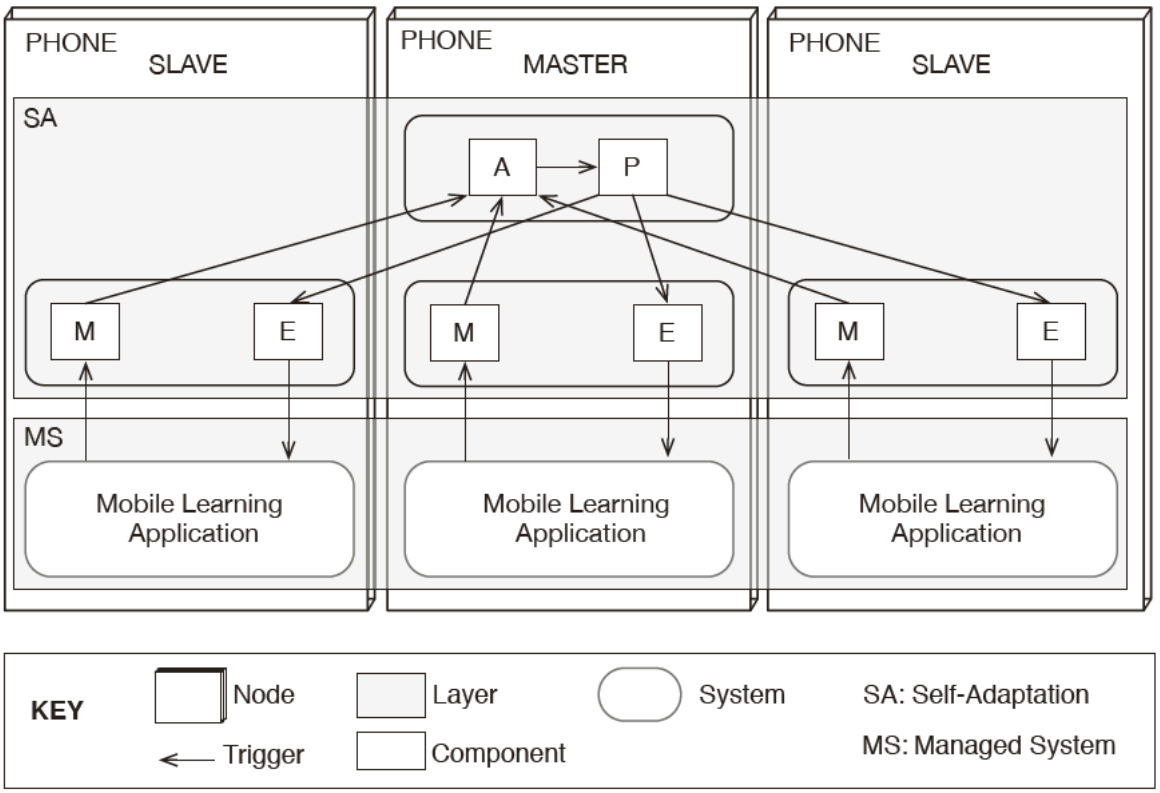


Рисунок 2.4 – Шаблон самовідновлення MVD «Головний/підлеглий» (для 3 телефонів)

Шаблон ведучий-підлеглий забезпечує координацію самоадаптації між вузлами в розподіленій системі. Пристрої мають подібні ролі (головний і підлеглий) як щодо адаптації у другому циклі MAPE, так і щодо функціональності, наданої мобільним додатком (тобто керованою системою). Усі пристрої MVD (master і slaves) контролюють мобільний навчальний додаток і виконують на ньому адаптаційні дії, але тільки master відповідає за аналіз і планування адаптацій. Якщо майстер виявляє, що кількості GPS-сервісів у MVD недостатньо для поточного завдання, він шукає додатковий сервіс. Якщо доступна безкоштовна послуга GPS, пристрій, що надає цю послугу, динамічно додається до MVD, якщо ні, то майстер періодично перевіряє. Головну роль може виконувати будь-який із телефонів у MVD, що робить організацію надійною у разі відмови головного.

2.4 Опис процесів самоадаптації з використанням верифікованих циклів MAPE-K

Структурні моделі самоадаптивного шару, описані в попередньому розділі, показують основні блоки циклів MAPE та їхні взаємодії. Ці моделі є корисними для пояснення механізмів адаптації на високому рівні абстракції та визначення модулів із зернистою структурою для реалізації самоадаптивного рівня. Однак, щоб гарантувати вимоги до надійності, нам потрібна сувора специфікація самоадаптивної поведінки разом із властивостями, які виражають вимоги до надійності. Ця специфікація дозволяє потім перевірити, чи самоадаптивна поведінка відповідає властивостям. З цією метою ми формально визначаємо поведінковий дизайн самоадаптивного рівня.

У цьому дослідженні ми використовуємо Uppaal [15], інструмент перевірки моделі, який підтримує моделювання поведінки (також званих процесами) за допомогою автоматизованих автоматів і перевірки властивостей стійкості, виражених у логіці дерева обчислень із

синхронізацією (TCTL). Часові автомати та TCTL забезпечують доступний формалізм. Автомати представляють стани, в яких можна знайти поведінку і визначають дії, які необхідно виконати при переході між станами. Ми починаємо з представлення процесів зовнішнього середовища. Потім ми представляємо поведінку першого циклу MAPE (послуги GPS) і закінчуємо поведінкою другого циклу MAPE (MVD). Для керованої системи ми моделюємо лише основні аспекти, необхідні для самоадаптації.

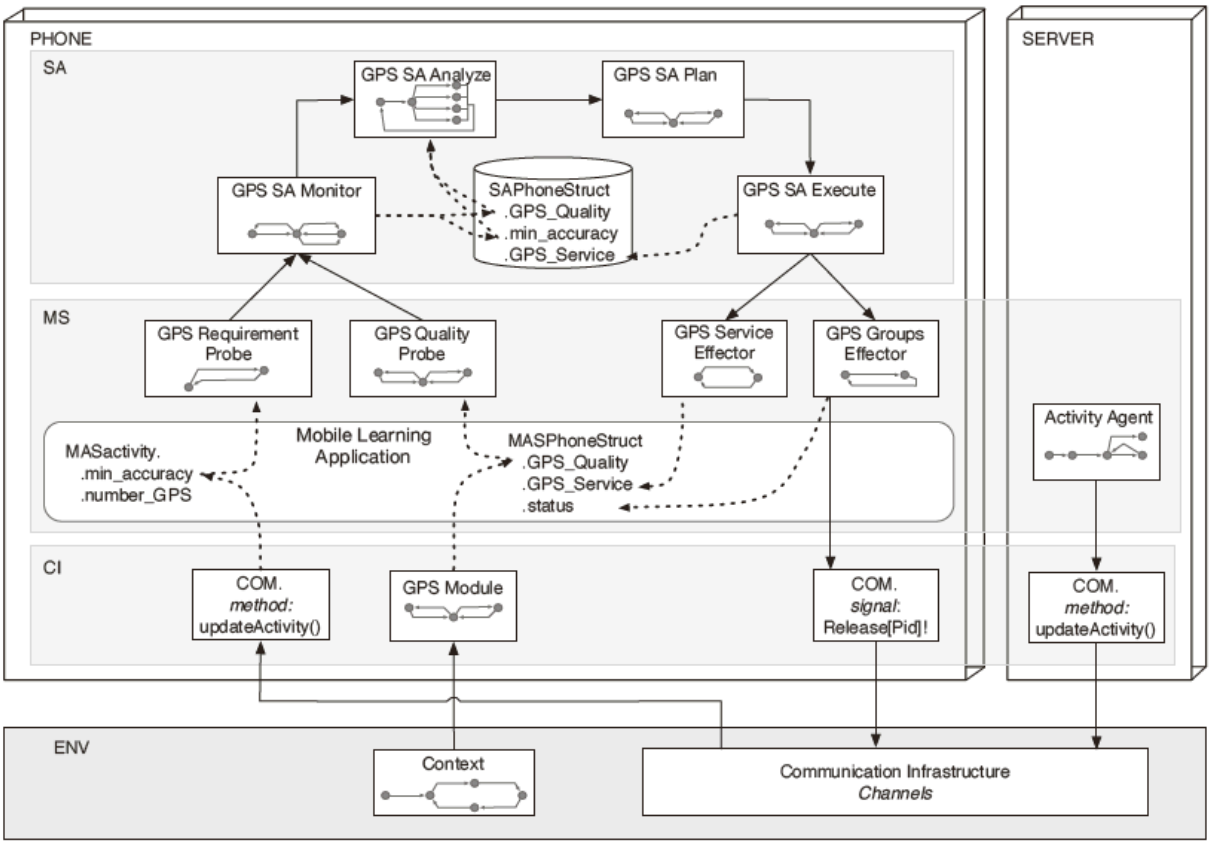


Рисунок 2.5 – Відображення процесу для самоадаптації служби GPS

Потреба в самоадаптації викликається змінами зовнішнього світу. Для цього необхідно формально визначити абстракцію зовнішнього світу. У нашому випадку зовнішній світ складається з трьох видів поведінки: агента активності, контексту та модуля GPS. На рисунку 2.5 показано поведінку по відношенню до циклу MAPE для самоадаптації служби GPS.

Агент активності, розташований на сервері активності, відповідає за встановлення имог до точності GPS для виконання завдань і кількості мобільних пристроїв, необхідних для групи. На рисунку 2.6 показаний автомат Activity Agent 1 . Перший крок ініціалізує розподілену програму, визначаючи початкове розгортання телефонів у MVD. Далі агент активності відповідає за контроль потоку активності.

Періодично 2 (Time_Activity), агент активності надсилає нові завдання в діяльності з новими вимогами (стан SubmitTask), доки завдання в діяльності (TotalLoops) не будуть виконані (стан Final). Вимоги до завдання визначають бажану мінімальну точність, необхідну для модулів GPS, і кількість модулів GPS у кожному MVD (представлені MASactivity.min_accuracy і MASactivity.number_GPS) (рис. 2.5).

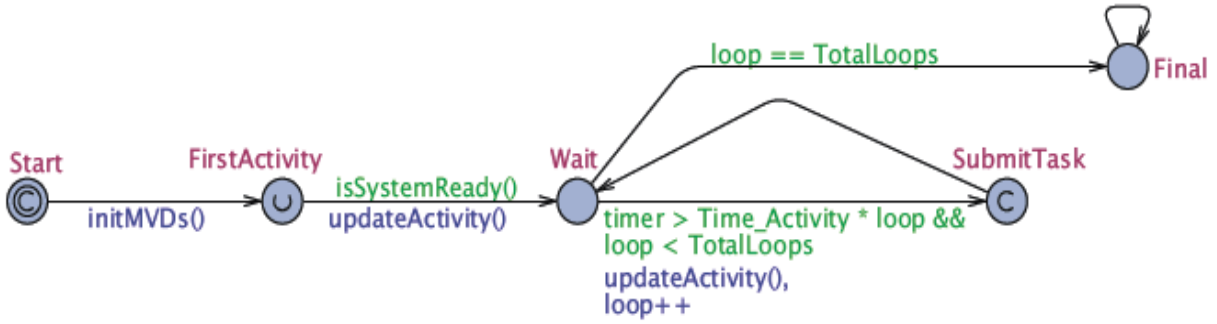


Рисунок 2.6 – Агент діяльності

Навколишнє середовище впливає на якість модуля GPS, потенційно переводячи службу GPS у небажаний стан. Середовище абстрактно моделюється контекстним автоматом (рис. 2.7), який описує стани середовища щодо перешкод GPS: Чистий або Шумний, і переходи в певні моменти часу (надаються функціями getNextNoise і getNextRecover). PID відноситься до ідентифікатора телефону. Один контекстний автомат створюється для кожного мобільного пристрою, що дозволяє нам моделювати вплив середовища на кожен модуль GPS.

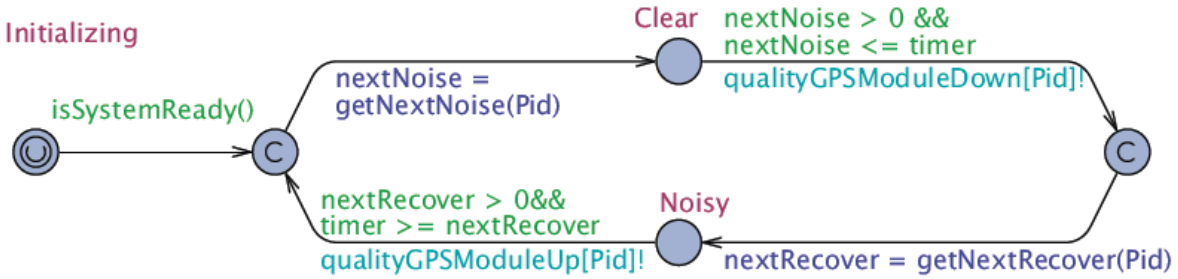


Рисунок 2.7 – Контекст

Поведінка GPS-модуля, який є частиною рівня комунікаційної інфраструктури, моделюється автоматом, показаним на рисунку 2.8. Ця поведінка отримує сигнали якості з контексту (через qualityGPSModuleUp і qualityGPSModulDown), які використовуються для оновлення представлення в системі, представленого MASPhoneStruct.GPS_Quality (рис. 2.5).

Переходи між станами спрацьовують на основі умов та/або отриманих сигналів (ми розміщуємо їх над стрілками переходів) і можуть виконувати дії чи надсилати сигнали іншим процесам (ми розміщуємо їх під стрілками переходів).

Модель абстрагує поведінку агента діяльності, надсилаючи нові вимоги на основі періоду. На практиці існує потік дій між сервером і агентами пристрою на основі призначення та виконання завдань.

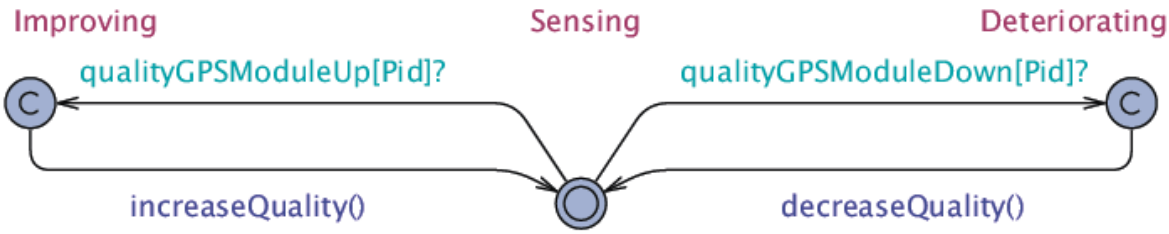


Рисунок 2.8 – Модуль GPS

Як згадувалося вище, ми абстрагували керовану систему до її основ, необхідних для роботи з самоадаптацією. Конкретно, керована система представлена за допомогою структур в UPPAAL. Рисунок 2.9 ілюструє, як

представлені різні елементи програми для мобільного навчання: поточна інформація щодо якості GPS (GPS_Quality) і стану служби (GPS_Service), участь в організаціях (status) і тимчасова інформація, яка використовується для визначення змін на Якість GPS (change_NotTreated, prev_quality).

```
struct{
  int GPS_Quality; // Undesired or OK
  int GPS_Service; // Deactivated or Active
  int status; // inMVD or Free
  bool change_NotTreated;  // internal (MAPE loop sync)
  int prev_quality; // internal (GPS quality)
}
```

Рисунок 2.9 – MASPhoneStruct

Маючи формальну модель зовнішнього світу та абстракцію керованої системи, ми можемо моделювати процеси самоадаптації.

Процеси самоадаптації служби GPS моделюють перший цикл MAPE, який займається активацією та дезактивацією послуг GPS на основі якості сигналів GPS. На рисунку 2.5 показано відображення поведінки циклу MAPE на компоненти циклу MAPE, показані на рисунку 2.3. Є дві змінні, які можуть вплинути на придатність модуля GPS: поточні вимоги до завдання та якість GPS. Тому ми моделюємо два процеси зондування, які збирають системну -інформацію. На рисунку 2.10 показано автомат, який представляє поведінку датчика якості GPS. Автомат містить стан, у якому визначається якість GPS (зондування) і два додаткові стани, у яких якість підвищується та знижується. Якщо якість GPS змінюється, GPS Service Monitor не повідомляється шляхом надсилання сигналу (SAqualityGPSIncreased і SAqualityGPSDecreased). Подібним чином GPS Requirement Probe фіксує зміни у вимогах до активності (MASactivity.min_accuracy), щоб повідомити компонент монітора (автомат не показаний).

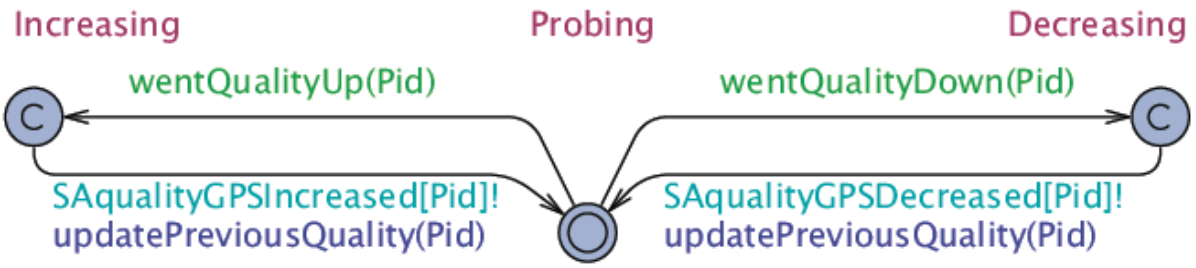


Рисунок 2.10 – Зонд якості GPS

Процес GPS SA Monitor (рис. 2.11) відповідає за моніторинг базової керованої системи та оновлення сховища знань (SAPhoneStruct на рисунку 2.5), підтримуючи аналіз і планування самоадаптивних дій. Автомат контролює дві окремі змінні. З лівого боку зміни на GPS вимоги (ініційовані агентом діяльності) обробляються (UpdateGPSReq). З правого боку обробляються зміни якості GPS (збільшення/зменшення - якість). Автомат сповіщає процес Analyze, коли виявляються зміни в знаннях (через канал SAGPSParametersChanged[Pid]).

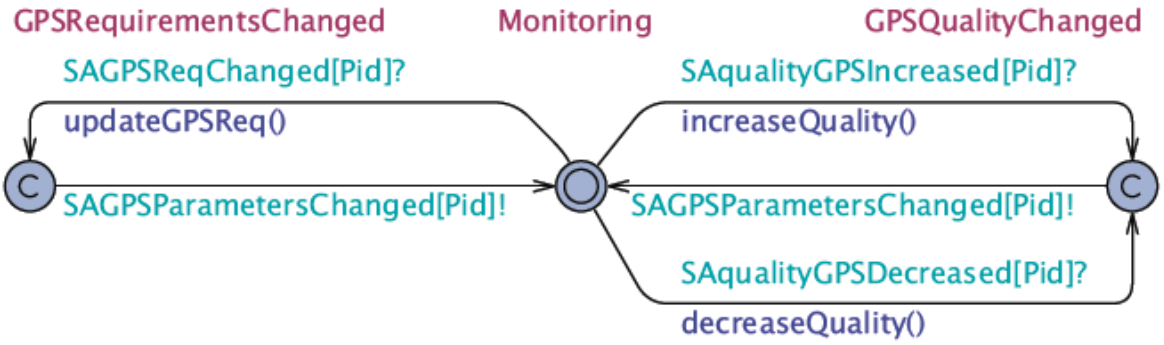


Рисунок 2.11 – Монітор GPS SA

Процес GPS SA Analyze (рис. 2.11) очікує в стані очікування тригера від процесу моніторингу, щоб здійснити перехід у стан аналізу. Можна досягти одного з чотирьох можливих станів (KeepGood, KeepBad, ChangedGood, ChangedBad), залежно від поточної якості GPS і вимог для виконання поточного завдання. У разі виявлення змін (ChangedGood, ChangedBad), процес планування сповіщається через сигнал

SA_GPS_degraded/recovered. Рисунок 2.12 ілюструє, як функції аналізу для визначення переходів до потенційно небажаних станів визначені в Uppaal.

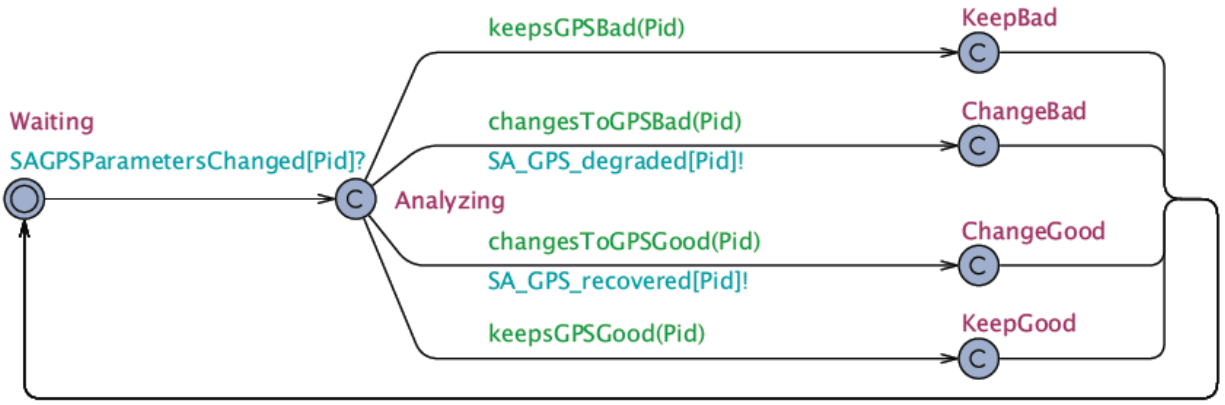


Рисунок 2.12 – Процес GPS SA Analyze

Процес GPS SA Plan (рис. 2.14) відповідає за планування адаптаційних дій щодо служби GPS. Тобто вирішити, чи вмикати чи вимикати службу GPS, що надається телефоном (ChangeToGood, ChangeToBad).

```
bool changesToGPSBad(phone_id Pid){
    if( SAphoneStruct[Pid].GPS_Quality <
        SAphoneStruct[Pid].activity1.min_accuracy &&
        SAphoneStruct[Pid].GPS_Service == 1){
       return true;
    }else{
       return false;
    }
```

Рисунок 2.13 – Функція changesToGPSBad

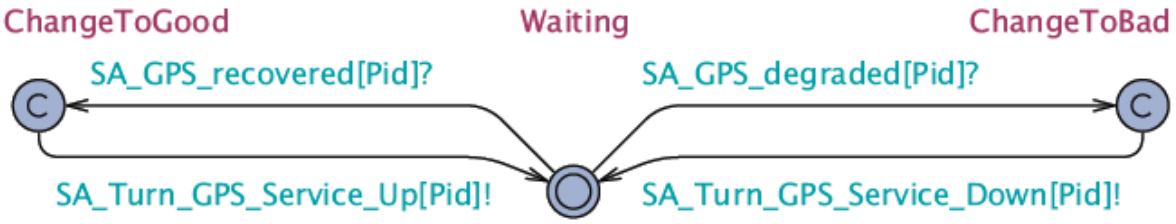


Рисунок 2.14 – План GPS SA

Процес GPS SA Execute (рис. 2.15) відповідає за застосування запланованих дій до керованої системи. У стані очікування план GPS SA

запускає перехід і змінює службу GPS через один із станів SetGPSServiceUp або SetGPSServiceDown.

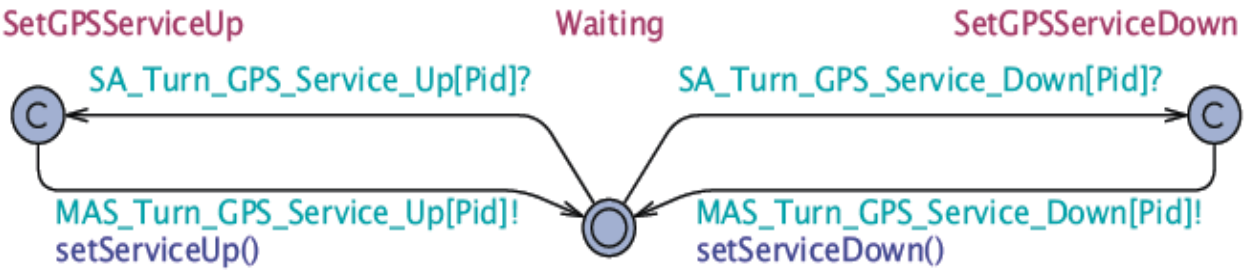


Рисунок 2.15 – Виконання GPS SA

Для підтримки процесу виконання передбачено два ефектори, які виконують фактичні адаптації до керованої системи. Процес GPS Service Effector (рис. 2.16) відповідає за активацію/дезактивацію служби GPS у керованій системі. Це представлено змінною MASPhoneStruct.GPS_Service (див. рис. 2.5). Крім того, процес GPS Group Effector призначений для видалення телефону з MVD у разі деактивації служби GPS (автомат не показаний).

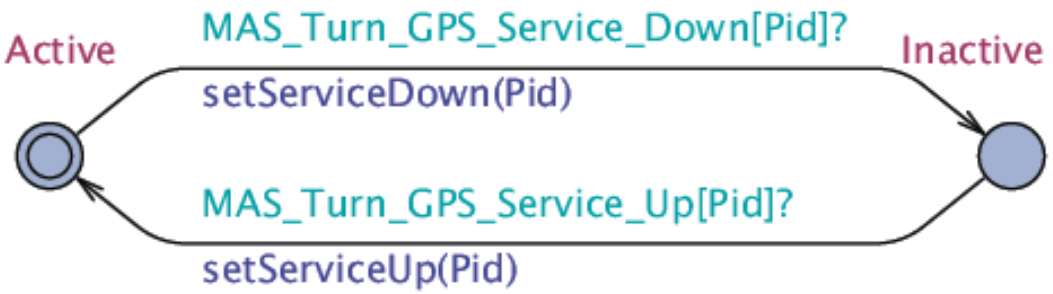


Рисунок 2.16 - Ефектор служби GPS

Процеси самовідновлення MVD моделюють другу петлю MAPE, яка займається відновленням небажаних станів MVD. На рисунку 2.17 показано відображення поведінки циклу MAPE на компоненти циклу MAPE, показані на рисунку 2.3.

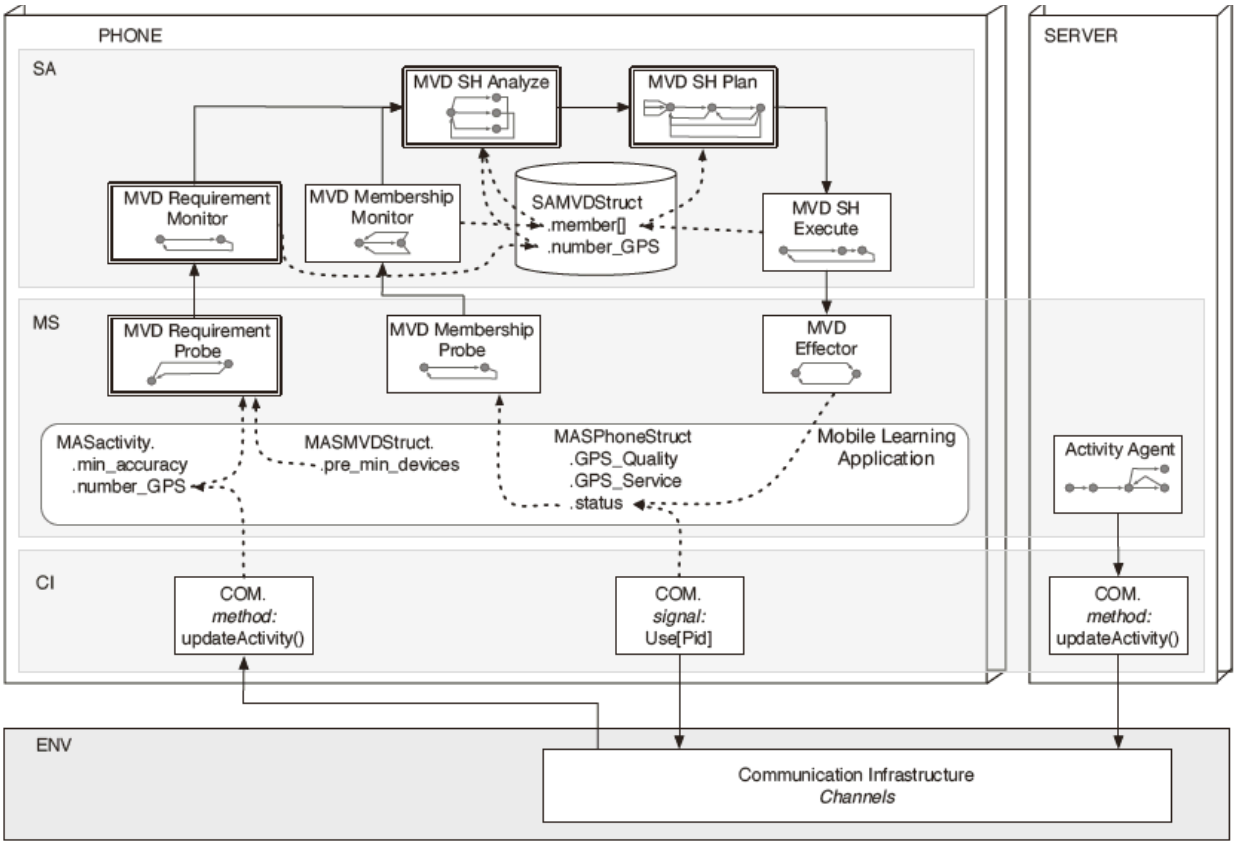


Рисунок 2.17 – Відображення процесу для самовідновлення MVD

Процес MVD Requirement Probe (рис. 2.18) показує поведінку, яка збирає інформацію про вимоги групи для поточного завдання (MASactivity.number_GPS, рисунку 2.16). Виявлені зміни повідомляються відповідному процесу Monitor. Середина відноситься до ідентифікатора MVD.

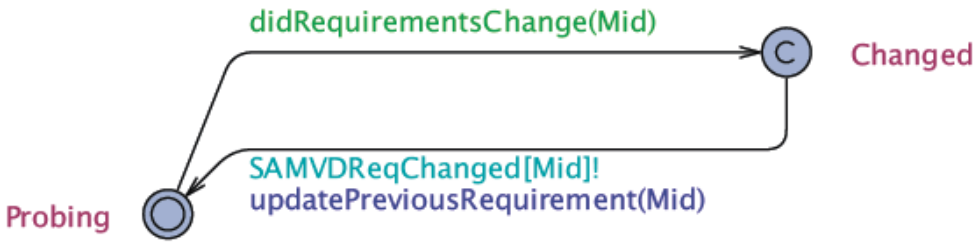


Рисунок 2.18 – Зонд вимоги MVD

Процес MVD Requirement Monitor (рис. 2.19) відповідає за моніторинг змін у вимогах завдання та, коли сигналізує зонд вимог, він оновлює сховище знань для процесу самовідновлення (змодельованого як SAMVDStruct, див.

рис. 2.16). Як показано на рисунку 2.17, процеси зондування та моніторингу, які відповідають вимогам MVD, створюються лише на головному пристрої.

Рисунок 2.19 – Монітор вимог MVD

Коли процес аналізу запускається процесом MVD SH Plan (рис. 2.20), ініціюється пошук телефону (found_Phone , який пропонує послугу GPS (LookForFreeGPS). Якщо ресурс не знайдено, процес залишається в стані NoFreeGPS і повторює пошук, доки не знайде послугу та не досягне своєї мети (AllFine). Телефон, який надає послугу (found_Phone), отримує сповіщення про інтеграцію в MVD. Лише головний телефон виконує процес планування.

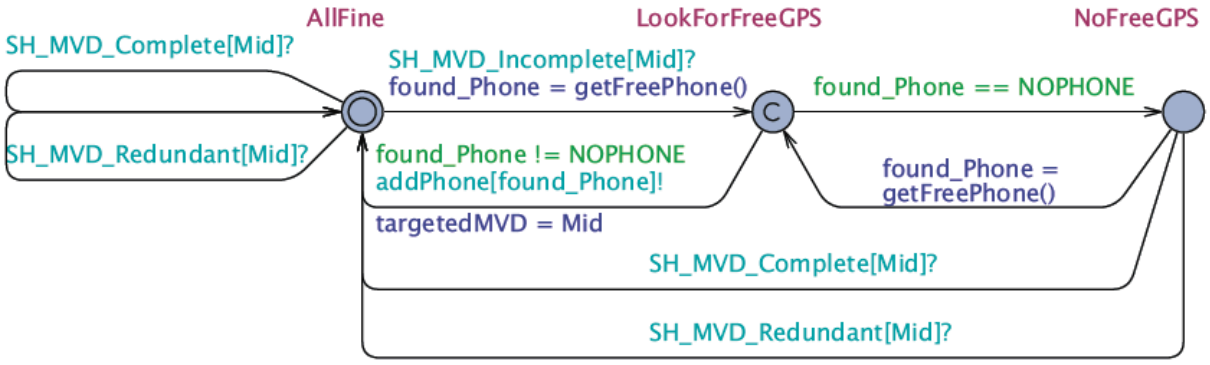


Рисунок 2.20 – Процеси MVD SH Plan

Ми створили різні сценарії зі збільшенням кількості телефонів і MVD. Навчальна діяльність складалася з 6 завдань. Вимірювання підтверджують, що вартість обробки для верифікації експоненціально зростає зі складністю сценаріїв (кількість телефонів і номер або MVD). У цьому конкретному домені сценарій із 3 телефонами та 1 MVD потребував 393 мс для перевірки властивості взаємоблокування (F1), у той час як 6 телефонів і 2 MVD

вимагали близько 21 хвилини, а 12 телефонів і 3 MVD вимагали кількох годин для аналізу. Результати аналізу показують лише невеликі відмінності щодо вартості перевірки інших властивостей для того самого сценарію. Наприклад, для сценарію з 1 MVD і 3 телефонами вартість перевірки коливалася від 167 мс до 178 мс для властивостей F2 і властивостей надійності від R1 до R9.

Оригінальний мобільний навчальний додаток було реалізовано за допомогою JADE [16]. JADE — це платформа, яка надає можливості для розробки розподілених багатоагентних систем, включаючи служби для передачі повідомлень, реєстрації та виявлення служб тощо. Ми коротко пояснюємо, як ми реалізували самоадаптивний рівень поверх застарілої системи. Ми відобразили один до одного поведінкові процеси, представлені в цьому розділі, на класи Java. Рисунок 2.21 ілюструє класи, які реалізують цикл MAPE служби GPS.

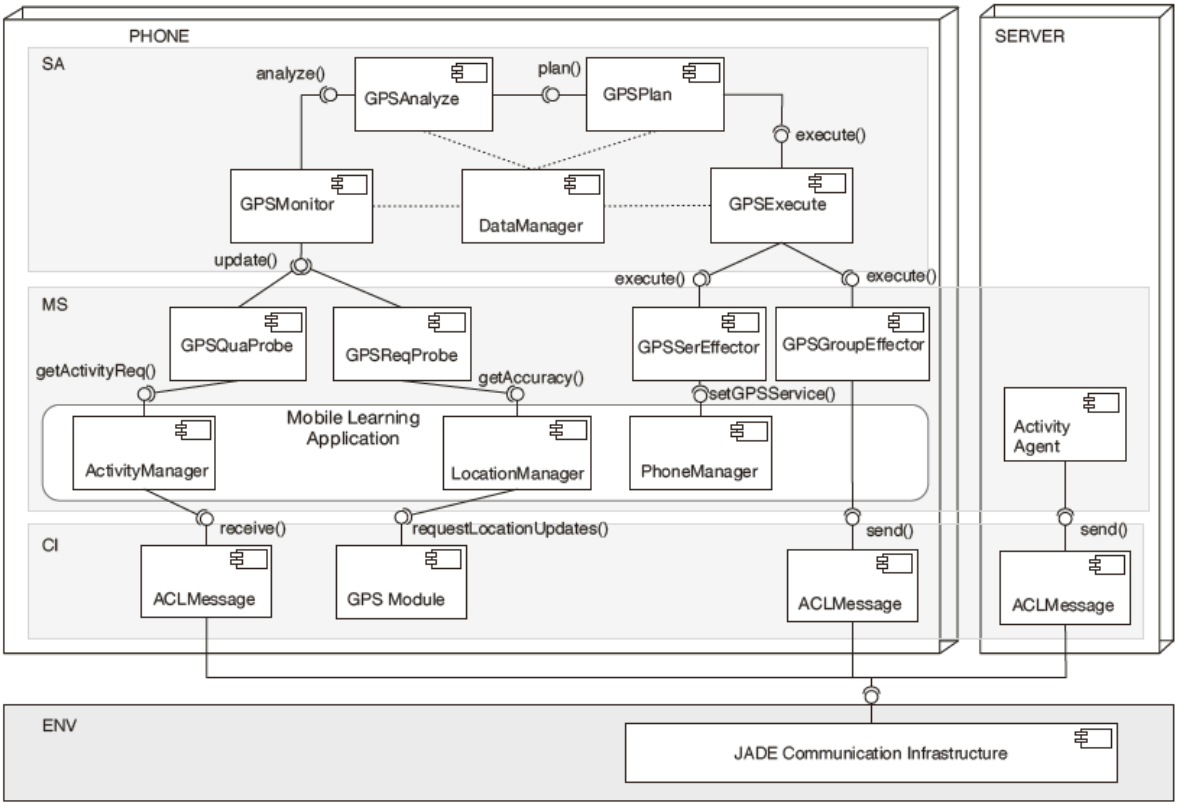


Рисунок 2.21 – Компонентна діаграма UML самоадаптації служби GPS

Процеси зондування реалізуються як поведінка Jade, яка виконується агентом на кожному мобільному пристрої.

Рисунок 2.22 показує реалізацію GPS Quality Probe. Код, який виконується на періодичній основі (TickerBehaviour), отримав доступ до поточних даних точності GPS через інтерфейс getAccuracy, який пропонує LocationManager. Класи ефекторів реалізують ефектори, які мають доступ до основної системи, щоб адаптувати її за потреби.

Наприклад, клас GPSServiceEffector реалізує ефектор, який використовує метод setGPSService() PhoneManager для активації (або деактивації) служби GPS за потреби.

```
public class GPSQualityProbeBehaviour extends TickerBehaviour{
[...]
  public GPSQualityProbeBehaviour(Agent agent, long period, float threshold)
      {
    super(agent, period);
  }

  @Override
  protected void onTick() {
    float accuracy = LocationManager.getInstance().getMyLocation().
        getAccuracy();
    myLogger.log(Logger.INFO, "Tick! Check GPS accuracy");
    if(accuracy - prev_accuracy > Threshold){
      myLogger.log(Logger.WARNING, "GPS accuracy changed");
      GPSMonitor.getInstance().update("accuracy",accuracy);
    }
    pre_accuracy = accuracy;
  }
}
```

Рисунок 2.22 – Клас GPSQualityProbleBehaviour

Аналогічно, ми відобразили процеси циклу самовідновлення MVD на класи Java. Для зв'язку між майстрами та агентом активності, з одного боку, та між агентами MVD, з іншого боку, ми використовували ACLMessages (мовні повідомлення агента), надані JADE.

Ці повідомлення пропонують користувачу комунікаційні примітиви високого рівня та допоміжні протоколи, такі як запит-підтвердження, повідомлення тощо.

Висновки до розділу 2

В даному розділі представлена інформація про MAPE, розглянуто програму для мобільного навчання, описано проблему та описує архітектуру самоадаптивного рішення. Детально досліджено поведінкові моделі самоадаптивної системи та необхідні властивості їх перевірки. Було проведено імітаційне моделювання яке дозволило розширили існуючу мобільну навчальну програму за допомогою рівня самоадаптації, зробивши систему стійкою до зниження точності GPS. Ми розробили самоадаптивний рівень як набір взаємодіючих верифікованих циклів MAPE. Щоб гарантувати необхідні вимоги до надійності, використано автомати з синхронізацією для визначення поведінки циклів MAPE і виражали вимоги до надійності як формальні властивості в TCTL. Інструмент Uppaal дозволив перевірити властивості, поведінковий дизайн було зіставлено з реалізацією Java.

РОЗДІЛ 3. ПОБУДОВА ФОРМАЛЬНИХ ШАБЛОНІВ ЗАБЕЗПЕЧЕННЯ АДАПТИВНОСТІ РОЗПОДІЛЕНОЇ СИСТЕМИ

3.1 Опис проблеми побудови формальних шаблонів адаптивних систем

Щоб допомогти розробникам ефективно та правильно визначити моделі та властивості, у цьому розділі вивчається консолідація формальних проектів сімейства самоадаптивних систем. Ми задокументували одну з конкретних програм у попередньому розділі. Сімейство систем визначає цільовий домен, який характеризується наступним чином:

- системи містять програмне забезпечення, розгорнуте на розподілених вузлах;
- вузли мають чітке положення в середовищі (і можуть бути мобільними);
- вузли мають постійний доступ до зв'язку;
- динаміка в системі та середовищі на порядки нижча за швидкість зв'язку та виконання програмного забезпечення.

Результати дослідження являють собою набір формально заданих шаблонів для проектування самоадаптивних систем на основі MAPE-K і перевірки властивостей стійкості та відкритості. Ми називаємо ці шаблони MAPE-K Formal Templates. Шаблони відповідають дослідницьким питанням:

- Як змоделювати поведінку компонентів MAPE-K, щоб відповідати вимогам надійності та відкритості ?
- Які властивості необхідно перевірити, щоб забезпечити докази того, що необхідна поведінка самоадаптації забезпечена".

Формальні шаблони MAPE-K включають:

1) шаблони поведінки, які можна багаторазово використовувати для моделювання різних компонентів циклу зворотного зв’язку MAPE-K (на основі мереж синхронізованих автоматів);

2) шаблони специфікації властивостей, які підтримують перевірку правильності поведінка адаптації системи (на основі логіки дерева обчислень у часі).

Самоадаптивна система, як правило, складається з керованої системи та контуру зворотного зв’язку. Керована система має справу з питаннями домену для користувачів, тоді як цикл зворотного зв’язку має справу з проблемами адаптації керованої системи (наприклад, оптимізація керованої системи для різних робочих умов, відновлення керованої системи при виявленні несправності тощо).

Одним із важливих завдань у розробці самоадаптивних систем є надання доказів того, що цілі системи задовольняються щодо умов експлуатації, що динамічно змінюються [48]. З цією метою сучасний рівень самоадаптації на основі архітектури підтримує використання формальних методів.

Існує мало систематичного закріплення знань про дизайн для майбутнього використання. Лише кілька зусиль було зроблено для консолідації досвіду проектування самоадаптивних систем, включаючи [48], який документує дванадцять шаблонів для проектування самоадаптивних систем, і який представляє набір шаблонів для децентралізованого керування MAPE-K петлі зворотного зв'язку. Однак основна увага зосереджена на структурних аспектах, і бракує чіткого підкріплення шаблонів. Інші дослідники продемонстрували цінність формально визначених шаблонів, наприклад формально визначених шаблонів моделювання та визначених шаблонів специфікації для імовірнісних вимог. В [43] вказують на те, що визначення таких шаблонів є важливою дослідницькою проблемою для самоадаптивних систем.

3.1.1. Шаблони проектування

Загальний підхід до консолідації знань про дизайн полягає в документуванні шаблонів проектування. У сфері самоадаптивних систем на основі архітектури було зроблено лише кілька зусиль щодо документування шаблонів проектування.

Робота [35] представляє декілька шаблонів реконфігурації програмного забезпечення для динамічної еволюції програмних архітектур. Автори визначають шаблон реконфігурації, що складається з повторюваних послідовностей кроків адаптації (наприклад, зупинка/запуск, відзв'язування, додавання/видалення), необхідних для забезпечення послідовної адаптації програмної системи. У [36] автори використовують шаблони реконфігурації в контексті самокерованих сервіс-орієнтованих програмних систем, тоді як у [28] продемонструвати їх корисність у розробці рішень проміжного програмного забезпечення на основі архітектури.

Дослідження [37] пропонує адаптивний стиль архітектури програмного забезпечення, що складається з:

1) основного нижнього рівня з компонентами програми, які керують роботом;

2) одного або кількох мета-рівнів з компонентами мета-рівня, які реалізують відмовостійкість, динамічне оновлення, виявлення ресурсів, перерозподіл тощо.

У запропонованій архітектурі кожен рівень може адаптувати рівень r під ним. Автори використовують підхід для проектування та реалізації самоадаптивної поведінки в програмному забезпеченні робототехніки.

Робота [22] представляє набір із 12 шаблонів, заснованих на ряді досліджень самоадаптивних систем. Шаблони документуються за класичним шаблоном, який описує призначення шаблону, контекст, структуру за допомогою діаграм класів UML і наслідки шаблону серед інших елементів опису. Представлені шаблони зосереджені на рівні розробки програмного

забезпечення. Вони спрямовані на полегшення проектування та побудови самоадаптивної системи програмного забезпечення, надаючи альтернативні рішення для адаптації в реалізації систем.

В іншому дослідженні [41] автори представляють набір шаблонів для децентралізованого керування в самоадаптивних системах. У дослідженні задокументовано п’ять шаблонів для координації та взаємодії кількох контурів керування MAPE-K, включаючи шаблон MAPE головний-підлеглий, шаблон координації та шаблон обміну інформацією для великих самоадаптивних систем. Основна увага шаблонів зосереджена на структурних аспектах архітектурного дизайну логіки самоадаптації. Шаблони ілюструються конкретними прикладами програм, з яких вони були отримані.

Представлені зусилля забезпечують ряд задокументованих шаблонів для самоадаптивних систем на основі архітектури. Шаблони варіюються від архітектури високого рівня до рівня конкретного дизайну. Основна увага приділяється структурним аспектам. Робота, представлена в цьому документі, доповнює ці зусилля шаблонами проектування, щоб точно визначити поведінку взаємодіючих компонентів MAPE-K.

3.1.2. Шаблони специфікації властивості

Друга частина цієї роботи зосереджена на шаблонах специфікації властивостей, які дозволяють сформулювати та перевірити необхідні властивості поведінки MAPE-K. Наскільки нам відомо, не існує пов’язаної роботи щодо шаблонів властивостей, присвячених самоадаптивним системам. Ми обговорюємо низку загальних підходів, які не зосереджені на самоадаптивних системах, а потім наближаємося до початкових ідей щодо шаблонів специфікації властивостей самоадаптивних систем.

Дослідження [33] вводить специфікації властивостей для перевірки кінцевих автоматів. Специфікації дозволяють виражати повторювані властивості в узагальненій формі, дозволяючи моделювати та перевіряти

системні вимоги в суворий спосіб. Інші приклади специфікацій властивостей були запропоновані для систем реального часу і систем з імовірнісними вимогами до якості. Робота [39] представляє результати дослідження шаблонів специфікацій для додатків на основі послуг. Автори провели широкий аналіз використання документованих властивостей специфікації для великої кількості специфікацій конкретної банківської компанії. Поточні шаблони властивостей не визначені для застосування в самоадаптивних системах, а останні зусилля щодо специфікації властивостей для самоадаптивних систем не були консолідовані для створення шаблонів для цієї галузі.

Оскільки перевірка моделі є одним із відомих підходів до надання доказів властивостей системи, ми проаналізували існуючу роботу, яка використовує методи перевірки моделі для перевірки самоадаптивних систем. З цього аналізу ми вивели цікаву модель, яка відображає різні типи поведінки самоадаптивних систем у зонах простору станів: нормальна поведінка, небажана поведінка, адаптаційна поведінка та неприпустима поведінка. Цікаві властивості щодо самоадаптації зазвичай відображаються на переходах між різними зонами. Наприклад, властивість може виражати , чи можна досягти неприпустимих станів із нормальної поведінки системи, або вона може виражати, чи правильно система адаптується від небажаної поведінки за допомогою поведінки адаптації назад до нормальної поведінки. Ця модель зони забезпечує потенційну основу для систематичної документації властивостей специфікації для самоадаптивних систем.

3.2 Цільовий домен для застосування шаблонів високо адаптивних систем

Шаблони втілюють знання, які ми отримали в результаті формалізації адаптивної поведінки для ряду самоадаптивних систем у мобільному

навчанні, управлінні трафіком і робототехніці. Цільова область, яку ми досліджували, представляє собою самоадаптивні системи.

Оскільки ми розглядаємо розподілені самоадаптивні системи, керована система зазвичай складається з кількох частин (розгорнутих на різних вузлах), які ми позначаємо локальними керованими системами. Ці частини можуть бути адаптовані за допомогою одного або кількох контурів зворотного зв'язку. Зокрема, ми вивчаємо самоадаптивні системи, засновані на циклах зворотного зв'язку MAPE-K, щоб відповідати вимогам надійності та відкритості . Важливо не помічати, що наша особлива увага приділяється адаптаціям, які потребують додавання та видалення ресурсів в системі. Під ресурсами ми маємо на увазі контрольовані частини керованої системи, такі як вузли, підсистеми та компоненти. Що стосується надійності, ми вивчаємо самоадаптацію, щоб мати справу з частинами системи, які виходять з ладу. Що стосується відкритості, ми вивчаємо самоадаптацію, щоб мати справу з частинами, які виникають і йдуть динамічно. З цією метою один або більше циклів зворотного зв'язку MAPE-K можуть спостерігати за локальними керованими системами та контекстом їх виконання та адаптувати локальні керовані системи шляхом додавання та видалення ресурсів.

Нижче ми описуємо два приклади програм, які ми використовуємо для ілюстрації різних шаблонів. Ці програми були екземплярами програм з яких були отримані шаблони. Ми використовуємо TA і TCTL для формалізації шаблонів MAPE-K. Для специфікації моделей і перевірки властивостей ми використовували інструмент Uppaal [15].

3.2.1. Мобільний навчальний додаток

Один із мобільних навчальних додатків, який ми вивчали, підтримує навчальну діяльність за допомогою мобільних пристроїв із підтримкою GPS [18]. Служби GPS є ресурсами в цьому домені. Заходи вимагають роботи в групах, тобто два мобільні пристрої потрібні для обчислення відстані, три

пристрої для тріангуляції та більше пристроїв для інших більш складних завдань. Дві петлі зворотного зв'язку MAPE-K були розроблені, щоб забезпечити надійність щодо наступних двох проблем. По-перше, умови навколишнього середовища можуть вплинути на надійність GPS і, як наслідок, на якість вимірювань на основі GPS. Ми визначили першу петлю MAPE-K, щоб гарантувати використання лише послуг GPS, які забезпечують мінімальний рівень точності своїх вимірювань. По-друге, кількість доступних служб GPS у групі може не задовольняти необхідних послуг для діяльності. Ми визначили другу петлю MAPE-K, щоб переконатися, що групи володіють необхідними службами GPS для виконання необхідної діяльності. На рисунку 3.1 угорі показаний автомат із синхронізацією, який визначає поведінку середовища, в якому відбувається навчальна діяльність. Поведінка забезпечує абстракцію умов неба, за яких небо може бути ясним або хмарним. Переходи між цими станами були змодельовані за допомогою спеціальних функцій (get-NextCloudy, getNextClearing, isValid) і умов на таймері.

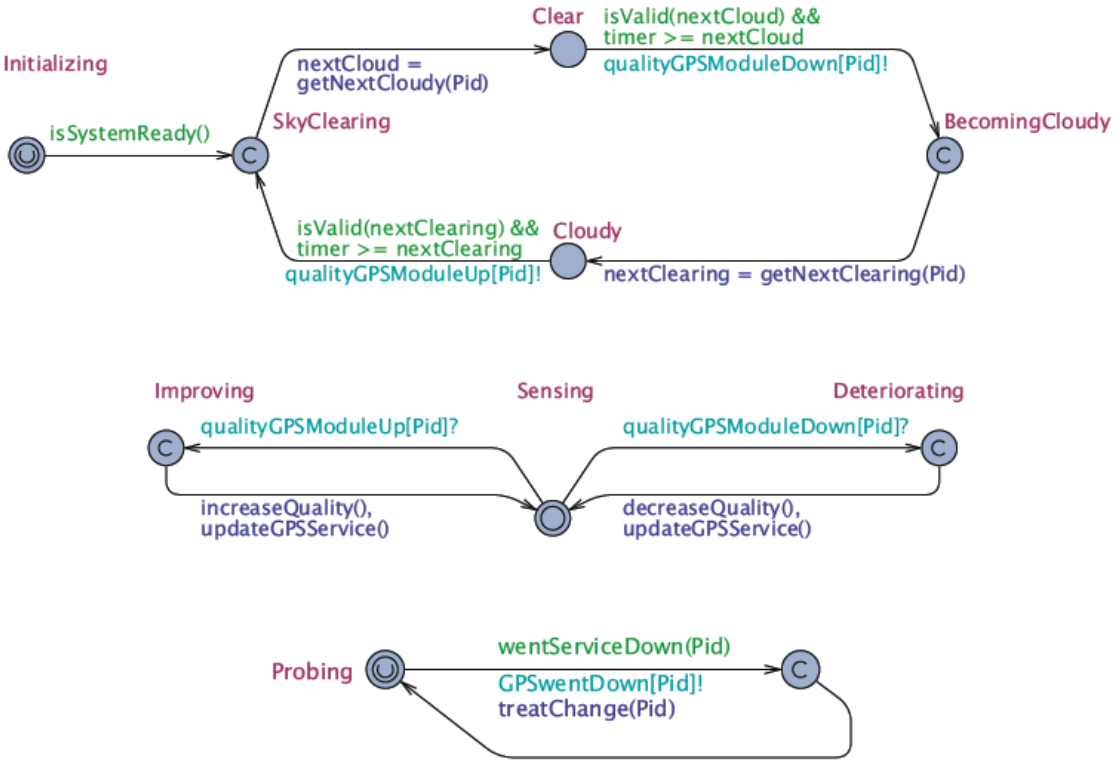


Рисунок 3.1 – Поведінка середовища (вгорі), GPS-модуля (посередині) і GPS-зонда для мобільного навчального додатка

Автомат, показаний на рисунку 3.1 посередині, визначає ту частину -поведінки компонента GPS, яка стосується якості отриманих місцезнаходження. На поведінку GPS впливають умови середовища, тобто послуга GPS або активується, або дезактивується (автомати спілкуються через сигнали qualityGPSModule*). Функція updateGPSService() дозволяє активувати або деактивувати службу GPS.

Поведінка зонда, показана на рисунку 3.1 внизу, визначає стан служби GPS і за потреби сповіщає пов’язану самоадаптивну петлю (через сигнал GPSwentDown).

Комбінація автоматів визначає поведінку локальної керованої системи, яка, залежно від умов, може активувати або деактивувати служби GPS. Виходячи з такої поведінки, група телефонів може стати неповною, коли служби дезактивовано. Ми зосереджуємося на повноті необхідних послуг GPS у групі як на надійності. З цією метою ми розширили логіку домену мобільного навчального додатку за допомогою циклів зворотного зв’язку MAPE-K, щоб мати змогу працювати з цими властивостями надійності.

3.2.2. Логістична роботизована система

Ми вивчали різні сценарії адаптації, коли роботи повинні виконувати транспортні завдання на складі, дотримуючись макета маршруту на основі графів. Щоб керувати складом, роботи зберігають представлення макета маршруту. Для цієї програми елементи карти (такі як смуги, розташування на макеті тощо) представляють ресурси в розподіленій системі. У цьому документі ми вивчаємо сценарій, коли роботам наказують вимкнути або ввімкнути певні смуги макета (наприклад, для технічного обслуговування на складі), змінити макет, додаючи та видаляючи його частини, де роботи можуть їздити (наприклад, додавши нове місце для доставки вантажів). Ми вказали макет, який кожен робот використовує для руху на складі (рис. 3.2) і

поведінку набору інтерфейсів, які дозволяють менеджеру маніпулювати елементами карти в макеті (рис. 3.3 ліворуч).

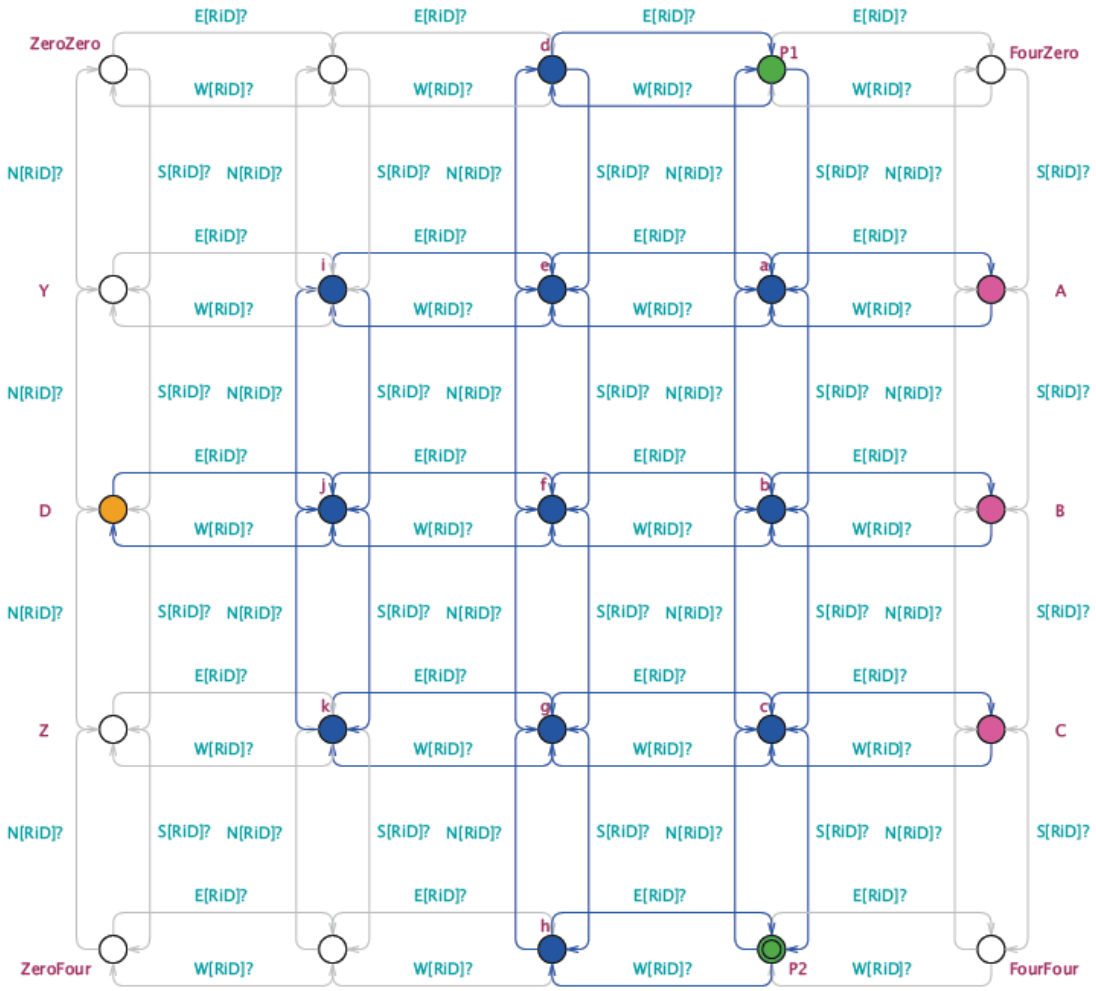


Рисунок 3.2 – Автомат, що описує макет карти робота

Ми визначили набір можливих маніпуляцій дій, які менеджер може виконувати з макетом під час виконання: додавання (bAdd-) і видалення (bRem-) місць розташування (-Loc), країв (-Edge) і місць призначення (-Dest) на карті. Наприклад, bRemLoc? на рисунку 3.3 ліворуч збирає запити від менеджера на видалення розташування з макета та повідомляє певного робота RiD , щоб він оновив свої знання щодо бажаних змін у макеті. На рисунку 3.2 показано зображення макета. Місцезнаходження та пункти призначення, які зараз недоступні для робота, позначені білим кольором, а

вимкнені краї – сірим. Робот не знає про існування цих конкретних місць і смуг.

Маніпулювання макетом карти може бути обмежено поточними умовами, наприклад, робот не може вимкнути смугу, якою він їде. Рисунок 3.3 праворуч визначає поведінку, яка контролює поточне розташування робота та цільовий пункт призначення в макеті.

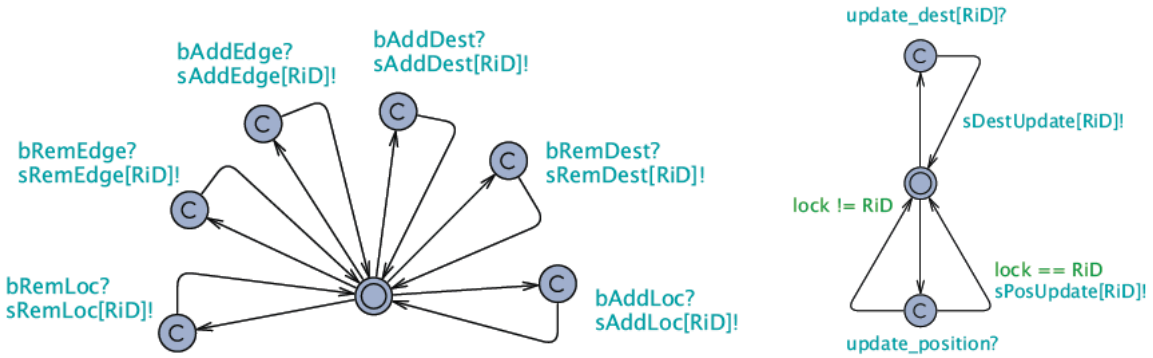


Рисунок 3.3 – Інтерфейси визначення поведінки, які менеджери використовують для оновлення

Щоб впоратися з маніпуляціями з макетом, ми розширили логіку домену роботів за допомогою циклу зворотного зв’язку MAPE-K.

3.3 Розробка шаблонів специфікації поведінки для моделювання контурів зворотного зв’язку для реалізації адаптивності

MAPE-K надає еталонну модель для реалізації самоадаптації, яка містить контур керування, що складається з чотирьох основних компонентів, які реалізують дії адаптації до керованої системи. На рисунку 1.2 (розділ 1) показана модель високого рівня самоадаптивної системи, де керована система розширена циклом зворотного зв’язку, реалізованим через компоненти MAPE-K. Компонент моніторингу на рисунку 1.2 отримує інформацію від керованої системи та її середовища та відповідно оновлює знання (1). Компонент аналізу (3) відповідає за визначення потреби в адаптації керованої

системи щодо цілей адаптації. Analyze отримує доступ до сховища знань для підтримки процесу аналізу. Якщо потрібна адаптація, компонент Plan (4) складає план, що складається з дій адаптації, які виконує Execute (5) для адаптації керованої системи за потреби.

Зараз ми представляємо набір формально заданих шаблонів для моделювання поведінки компонентів циклів зворотного зв'язку MAPE-K, які ми отримали з попереднього досвіду проектування самоадаптивних систем. Для кожного шаблону ми починаємо з представлення конкретних екземплярів конкретного компонента MAPE для двох прикладних сценаріїв. Потім ми вивчаємо загальні характеристики формалізованих компонентів і визначаємо моделі поведінки, які надають багаторазові специфікації для проектування різних поведінок MAPE.

Через обмеження простору ми не можемо надати детальну специфікацію всіх аспектів шаблонів, включаючи механізми запуску часу та подій, специфікацію структур даних, специфікацію всіх функцій тощо.

3.3.1. Знання

Для реалізації самоадаптації компоненти Monitor, Analyze, Plan and Execute використовують моделі (знання), які забезпечують абстрактне представлення відповідних аспектів керованої системи, її середовища та цілей самоадаптації. Ми ділимо знання на чотири частини (які технічно визначені як чотири визначення структури в Uppaal) наступним чином:

- ManagedSystemKnowledge надає абстракцію керованої системи; ця частина знань представляє відповідну інформацію щодо ресурсів у керованій системі. Ця інформація може містити дані про ресурси, про те, чи використовуються ресурси чи ні, та інші аспекти, такі як залежності ресурсів або властивості якості.

- EnvironmentKnowledge забезпечує абстракцію середовища, тобто ці знання представляють інформацію про контекст, у якому розташована та працює самоадаптивна система.
- ConcernKnowledge описує знання стосовно проблем адаптації, що цікавить; ця частина визначає необхідні ресурси для реалізації цілей самоадаптивної системи.
- AdaptationKnowledge представляє дані під час виконання, які спільно використовуються між поведінками MAPE; ми розрізняємо стани, які використовуються для непрямої синхронізації поведінки і робочі дані, які представляють дані, які поведінка використовує для реалізації своїх функцій (наприклад, історичні дані для аналізу, робочі процеси для адаптації тощо).

3.3.2. Монітор

Монітор збирає інформацію (через зонди) з керованої системи та, можливо, середовища, щоб оновити знання циклу зворотного зв’язку. Перед оновленням Знань Монітор може попередньо обробити зібрані дані. Прикладами попередньої обробки є стандартизація даних, фільтрація та агрегування даних.

Сценарій для навчання. У сценарії мобільного навчання було визначено автомат для опису поведінки GPS-монітора. GPS-монітор відповідає за оновлення інформації про приналежність телефонів до груп.

Зокрема, поведінка монітора:

- визначає, чи не працюють служби GPS;
- перевіряє, чи вже було повідомлено про збій GPS;
- визначає групу, у якій використовувався мобільний пристрій служби GPS;
- оновлює знання щодо визначеної групи.

Чотири кроки представлені чотирма станами автомата Monitor (рис. 3.4). У стані моніторингу поведінка очікує на сповіщення від зонда

(GPSwentDown[Pid]?). Якщо це нове сповіщення (isNew(caselD)), ідентифікація групи починається з CollectingGroup (determinMyMVD (Pid)), і нарешті, після GroupIdentified, поведінка оновлює знання щодо членства телефону за допомогою функції remove_Phone(myMVD).

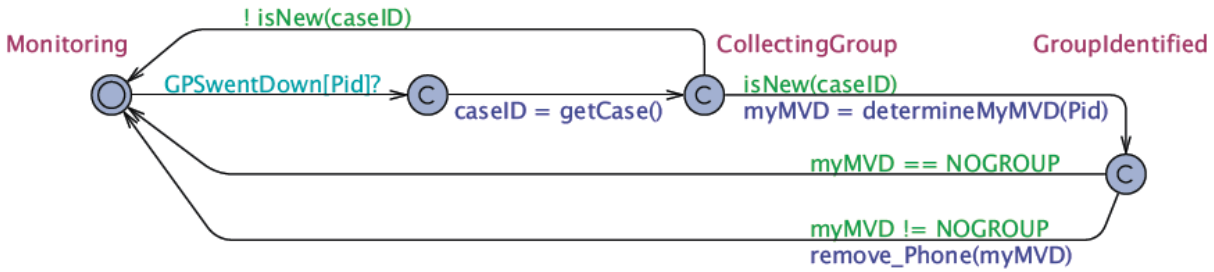


Рисунок 3.4 – Поведінка GPS-монітора мобільного навчання

Сценарій для транспортування робота. У сценарії транспортування робота ми розробили поведінку монітора, яка оновлює знання щодо:

- бажаних змін у макеті;
- поточного положення;
- пункту призначення робота.

Таким чином, поведінка відстежує два джерела даних, які потрібні логіці адаптації. По-перше, поведінка фіксує сигнали, які ідентифікують різні типи модифікацій, які слід застосувати до макета (сигнали можна надсилати через канали sAddEdge, sRemEdge, sAddLoc, sRemLoc, sAd-dDest і sRemDest, показані у верхніх автоматах на рисунку 3.5).

По-друге (рис. 3.5 внизу), поведінка відстежує контекстну інформацію робота, необхідну для визначення того, чи може відбутися адаптація, чи її слід відкласти, доки не будуть задоволені необхідні умови (наприклад, робот їде до місця призначення, яке потрібно видалити).

Це було зазначено в поведінці через два канали, які ідентифікували оновлення щодо поточного розташування робота (sPosUpdate) і цільового призначення (sDestUpdate). Поведінка монітора запускає процес аналізу за допомогою аналізу [RiD]! сигнал.

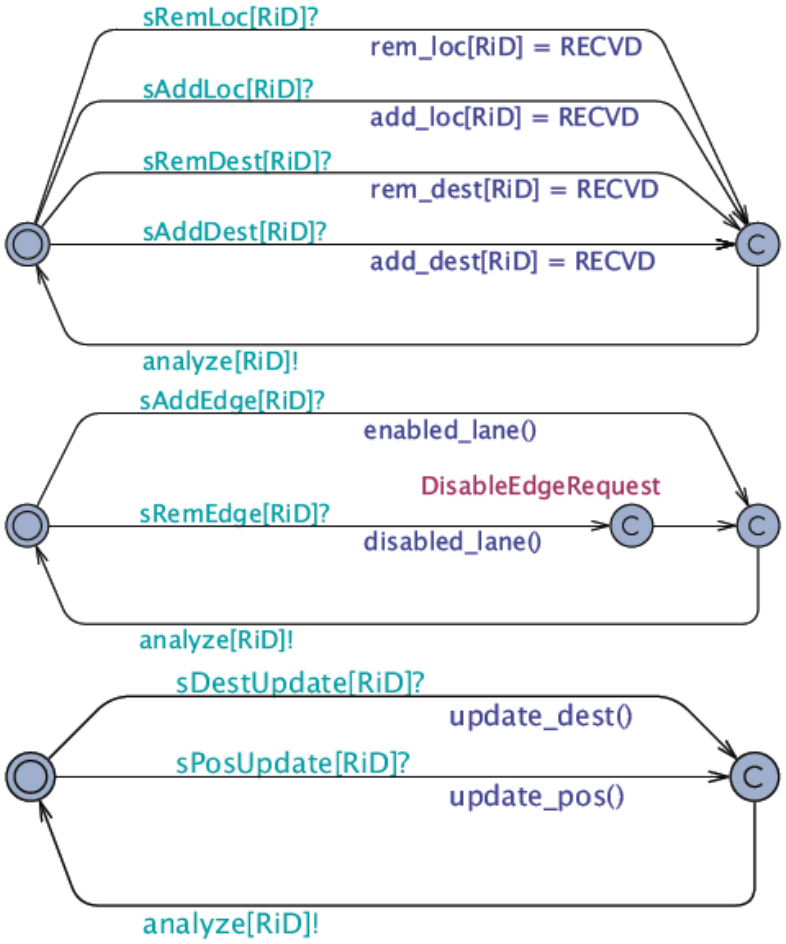


Рисунок 3.5 – Транспортний інтерфейс робота. Монітор поведінки

Шаблон дизайну монітора. Зараз ми представляємо шаблон проектування монітора, який консолідує наші знання щодо розробки поведінки монітора.

Активація подій, наприклад GPSwentDown[Pid]? у сценарії мобільного навчання та sDestUpdate[Rid]? у сценарії транспортування робота узагальнюються в механізм запуску, представлений для ініціювання збору даних. На рисунку 3.6 показано механізм запуску події, визначений за допомогою Monitor[Node_ID]? сигнал. У мобільному навчальному додатку це було вказано через функцію determinMyMVD(); у програмі робота дані для моніторингу містилися в сигналах. Ми узагальнюємо цей крок у функції getData(), яка відповідає за збір значень елементів даних, які поведінка повинна контролювати.

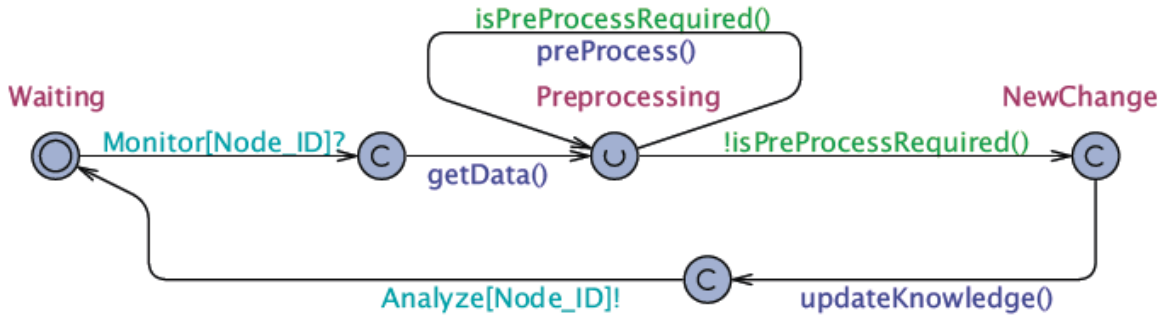


Рисунок 3.6 – Модель поведінки монітора з активацією події

Ми включаємо етап попередньої обробки в монітор для такої поведінки, яка може вимагати стандартизації даних, агрегації тощо (якщо isPreprocessingRequired(), то preprocess()). У випадку мобільного навчання необхідний був етап попередньої обробки, щоб уникнути звітування про повторні запити.

Нарешті, після того, як знайдено нові значення (NewChange), поведінка монітора оновлює знання та повідомляє відповідні компоненти Analyze. У сценарії мобільного навчання це було змодельовано за допомогою функції remove_phone() , яка оновлювала конкретні знання групи та встановлювала умови для аналізу поведінки для обробки нових даних. Для транспортної програми робота це було змодельовано за допомогою різних функцій оновлення (rem_loc[Rid] = RECVD, update_dest() тощо), а потім аналіз[Rid]! сигнал. Ми вказуємо цей процес у шаблоні розробки монітора за допомогою функції updateKnowledge(), а потім сигналу, який запускає аналіз (Analyze[Node_ID]!).

Підсумовуючи, поведінка монітора поділяється на наступні етапи: ініціювання моніторингу, збір даних, попередня обробка даних, оновлення робочих даних і поведінки аналізу сигналів.

Крім того, ми вказуємо шаблони, що запускаються за часом (усі варіанти шаблонів задокументовані на веб-сайті проекту). Тут ми представляємо монітор із запуском часу (рис. 3.7), який періодично ініціює свою поведінку для збору отриманих даних через get-Data(). Частота

монітора визначається умовою часу (t==Період) і інваріантом стану (t<=Період).

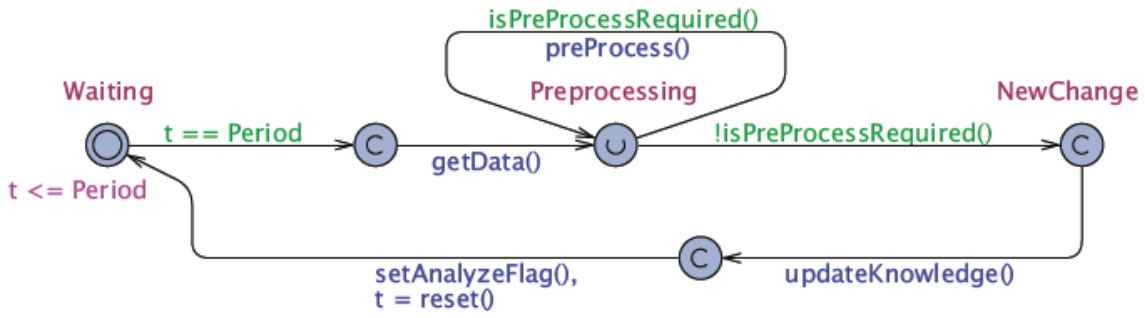


Рисунок 3.7 – Модель поведінки монітора з ініціюванням часу

3.3.3. Аналіз

Аналіз відповідає за визначення того, чи потрібні адаптаційні дії на основі стану керованої системи, навколишнього середовища та інтересу адаптації.

Сценарій для навчання. Для сценарію мобільного навчання ми розробили поведінку аналізу, яка визначає, чи мають групи достатньо послуг GPS для виконання навчальної діяльності. На рисунку 3.8 показано автомат поведінки аналізу із запуском часу (на основі частоти 5 одиниць часу). Поведінка використовує вимоги до активності та кількість використаних ресурсів GPS у групі для виконання аналізу. Функції getRequired() і getUsed() надають цю інформацію, звертаючись до ConcernKnowledge і ManagedSystemKnowledge. Потім ці знання використовуються для оцінки того, чи достатньо доступних послуг GPS у групі для реалізації цілей діяльності (за допомогою функцій порівняння). Результати аналізу використовуються для координації з Планом для вжиття дій у разі потреби (наприклад , SH_MVD_Incomplete[Mid]!).

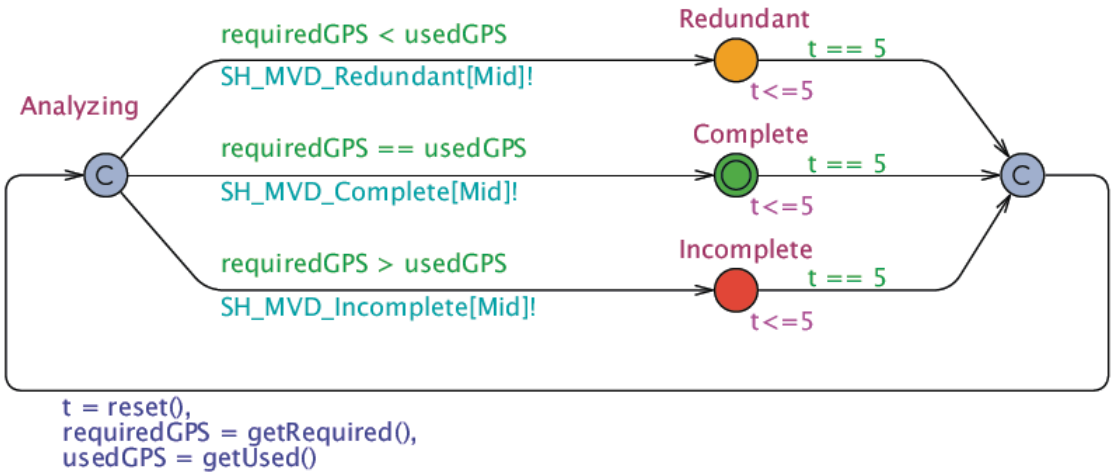


Рисунок 3.8 – Аналіз поведінки програми для мобільного навчання

Сценарій для транспортування робота. Для випадку транспортування робота поведінка Analyze повинна визначити, чи відповідає представлення макета в роботі справжньому макету, визначеному адміністратором. Ми вказали три функції, які відповідають за визначення правильності поточного представлення макета (noNeedForAdaptation()), чи потрібні нові смуги, місця чи інші ресурси (needAdaptationAdd()), чи джерела ресурсів слід видалити з макета (needAdaptationRemove()).

На рисунку 3.9 показано поведінку аналізу, викликаного подією, для сценарію робота.

На основі функцій поведінка може мати три різні переходи від стану очікування. Перехід через ElementsMissing виконується, якщо елементи на карті потрібно ввімкнути або додати. Перехід через ElementsExtra виконується, коли один (чи більше) елементів на карті потрібно вимкнути або видалити. У цих двох випадках поведінка Analyze зв’язувалася з поведінкою Plan у того самого робота (RiD) через сигналізацію (planning*[RiD]!).

У решті випадків поведінка аналізу безпосередньо повертається до стану очікування та чекає, доки не знадобиться новий аналіз.

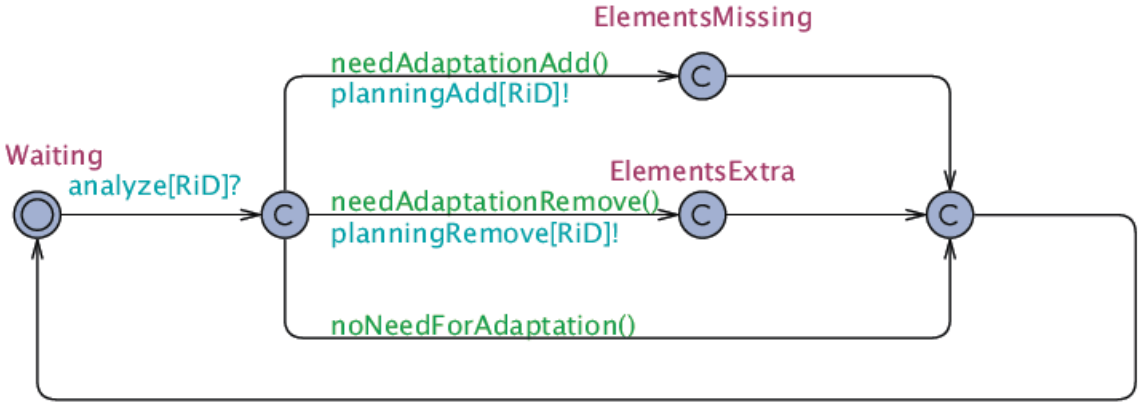


Рисунок 3.9 – Аналіз поведінки сценарію транспортування робота

Тепер ми представляємо шаблон Analyze (рис. 3.10), який консолідує досвід, який ми отримали під час проектування аналізу поведінки для різних програм, як показано вище.

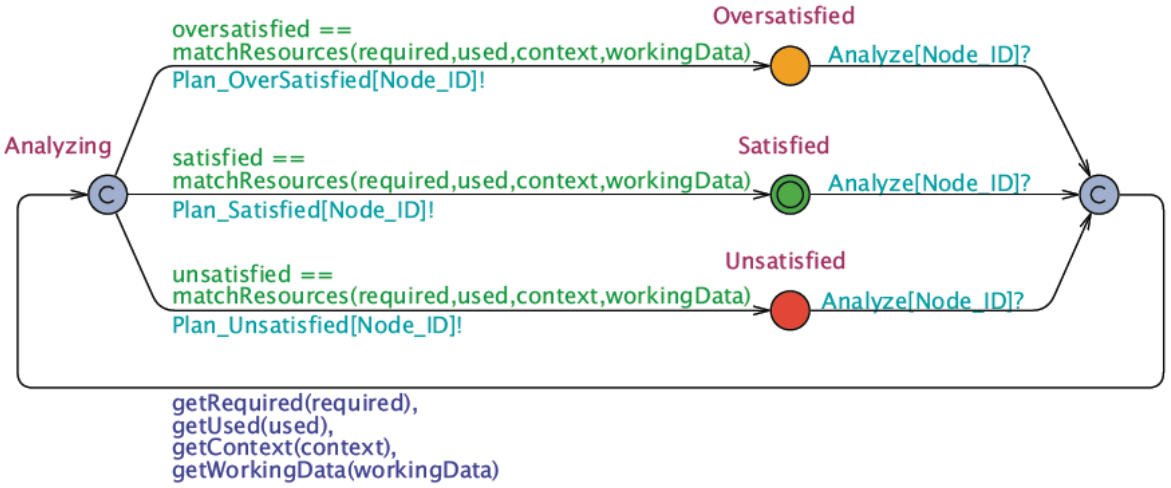


Рисунок 3.10 – Аналіз моделі поведінки за допомогою запуску події

Поведінка Analyze зіставляє необхідні ресурси з ресурсами, що використовуються, враховуючи поточний контекст і робочі дані (наприклад, служби GPS або смуги в макеті робота). Ми оснастили аналіз поведінки методами аналізу потреб в адаптації. Такі функції, як порівняння кількості послуг GPS і перевірка того, що робот працює з оновленим макетом (needAdaptationAdd()), узагальнено у функції matchResources(). Ми підтримуємо аналіз за допомогою набору функцій, які відповідають за збір

даних із різних представлень знань (функції get*()). Наприклад, ми спостерігали цей тип функцій у getRequired() у програмі для мобільного навчання, і подібні функції були внутрішньо визначені в трьох функціях аналізу, розроблених у програмі robots. Ми надаємо приклад (needAdaptationAdd) на рисунку 3.11.

```
1 bool needAdaptationAdd(){
2     return add_loc[RiD] == RECVD || enable_lane[RiD] == RECVD ||
          add_dest[RiD] == RECVD;
3 }
```

Рисунок 3.11 – Функція Analyze.needAdaptationAdd для трафіку роботів

Аналіз складається з трьох основних станів. Результат задоволений, коли керована система має ресурси, необхідні для реалізації своїх цілей. Повний стан у мобільному навчанні та вільний перехід у роботах (noNeedForAdaptation()) показують цю частину поведінки. Результат аналізу має статус «Незадоволений», якщо системі бракує ресурсів або вони не відповідають поточному контексту та цілям і «Надмірно задоволений», коли система має надлишкові ресурси. Ці декларації безпосередньо збігаються з Redundant/Incomplete і ElementsExtra/ElementsMissing з наших ілюстративних сценаріїв. Результати аналізу потім повідомляються поведінці плану (наприклад, Plan_UnSatisfied[Node_ID]!). Зауважте, що Analyze не може повідомляти стани Satisfied для поведінки плану, якщо плани адаптації можуть бути скасовані.

На рисунку 3.10 показано загальний шаблон для поведінки аналізу із запуском події (Analyze[Node_ID]?).

Підводячи підсумок, ми визначаємо такі кроки в поведінці аналізу: аналіз ініціювання, збір даних, процес аналізу та сигналізація пов’язаної поведінки плану.

3.3.4. Виконання шаблону дизайну

Рисунок 3.12 внизу показує поведінку виконання для видалення та вимкнення елементів представлення макета. Автомат визначає тип дій, які потрібно виконати, координує роботу з керованою системою для підготовки до дій адаптації та застосовує дії адаптації до керованої системи. У цих випадках потрібен другий крок, щоб переконатися, що жодні елементи карти не видаляються, хоча робот все одно потребує їх (наприклад, рух по смузі, яка буде вимкнена).

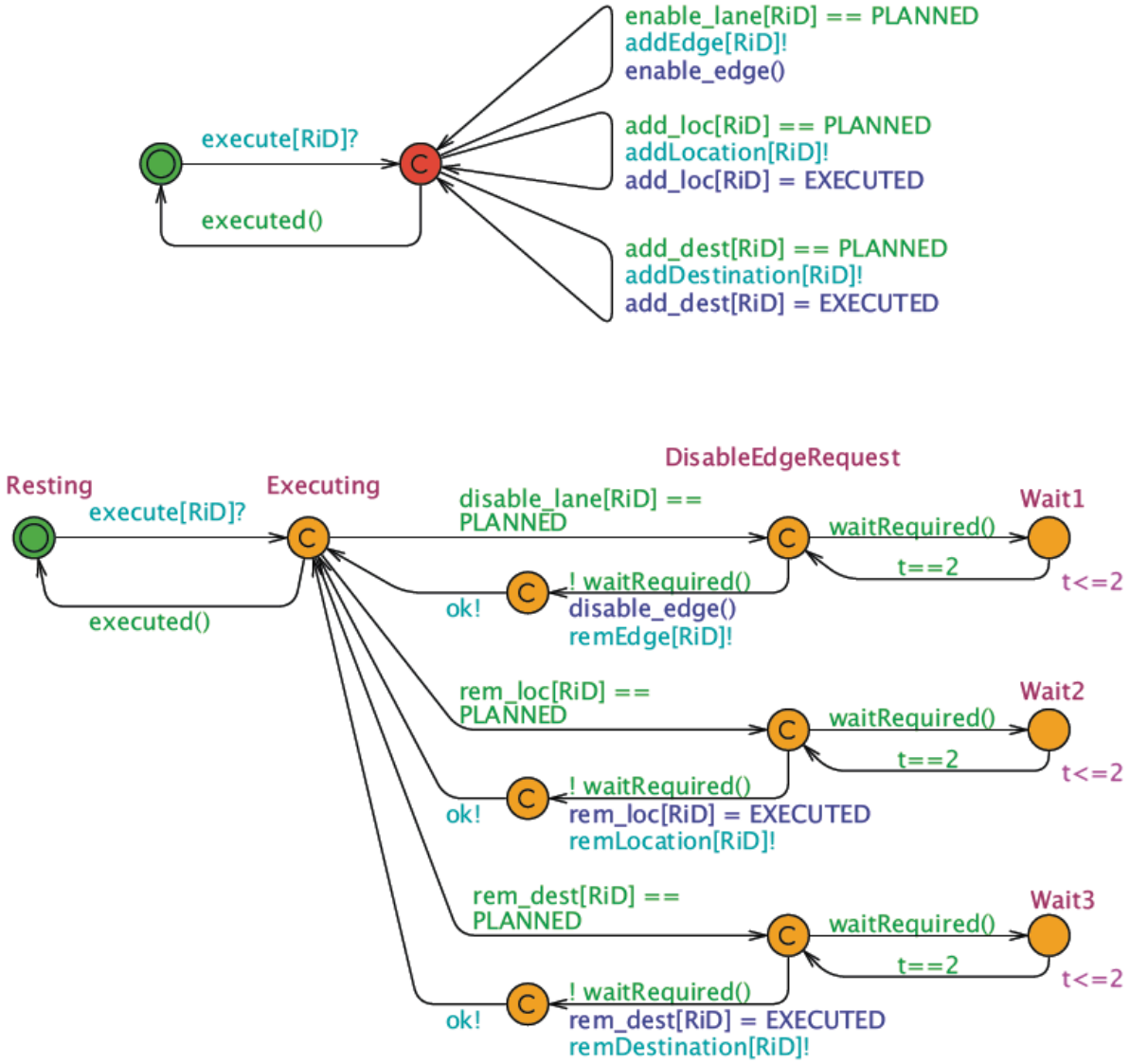


Рисунок 3.12 – Процес транспортування робота

Тепер ми представляємо шаблони Execute, які консолідують наш досвід розробки поведінки Execute для різних самоадаптивних систем. Ми визначили потребу у двох гілках поведінки «Виконати»: одна гілка займається додаванням ресурсів до керованої системи; інша гілка, щоб звільнити ресурси. На рисунку 3.12 показано «гілку додавання» шаблону поведінки Execute. «Видалити гілку» слідує еквівалентній логіці.

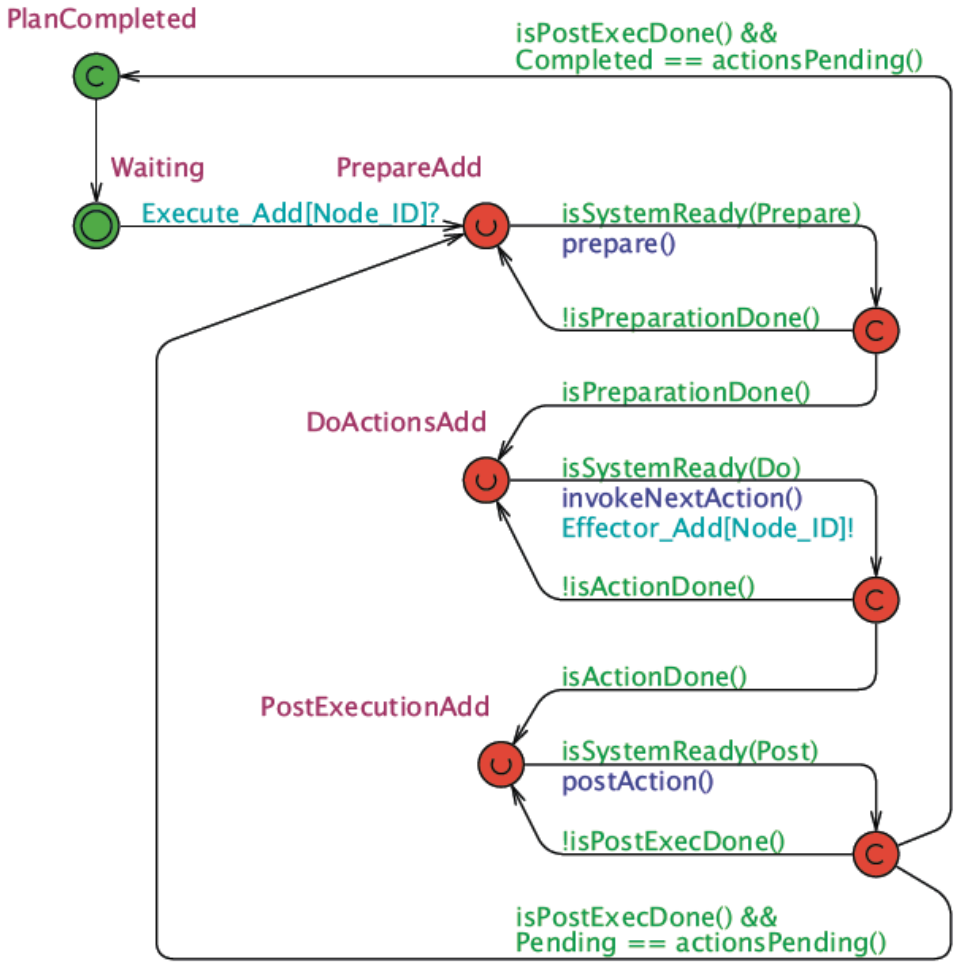


Рисунок 3.12 – Шаблон поведінки виконання із запуском часу

Ми визначили три фази в поведінці Execute: підготовка, виконання (doAction) і після виконання. По-перше, поведінка Execute запускається для виконання дії (Execute_Add[Node_ID]?). Execute перевіряє, чи керована система готова викликати заплановані дії (isSystemReady(Prepare)), як ми спостерігали за допомогою waitRequired() у випадку робота. Якщо це так,

Execute виконує підготовчі завдання (стан PrepareAdd) перед виконанням дії для додавання повторного джерела до керованої системи. Підготовка зазвичай вимагає кроків, необхідних для застосування дій адаптації, наприклад, блокування певних ресурсів (як у сценарії мобільного навчання), забезпечення того, що керована система перебуває в безпечному (неактивному) стані тощо. Така підготовка може вимагати багаторазової підготовки кроки.

Після завершення підготовки поведінка переходить до DoActionsAdd, де такі дії, як інтеграція нового телефону, додавання місцезнаходження та активація смуги, виконуються за допомогою функції (invokeNextAction() і сигналу Effector_Add[Node_ID]!).

Для додавання ресурсів можуть знадобитися завдання PostExecutionAdd, які виконуються за допомогою postAction(). Прикладами є розблокування ресурсів керованої системи та оновлення внутрішньої інформації в сховищах знань. Нарешті, поведінка Execute be перевіряє, чи виконано план (actionsPending()). Якщо інші дії очікують, поведінка повертається до стану PrepareAdd, звідки повторюється процес для виконання наступних дій у керованій системі.

Висновки до розділу 3

Отже, в цьому розділі запропоновано формально визначені шаблони MAPE-K, які документують досвід проектування сімейства самоадаптивних систем. Шаблони містять: шаблони специфікації поведінки для моделювання різних компонентів циклу зворотного зв’язку MAPE-K на основі мереж синхронізованих автоматів і шаблони специфікації властивостей, які підтримують перевірку правильності поведінки адаптації на основі логіки дерева обчислень із синхронізацією.

ВИСНОВКИ

В кваліфікаційній роботі розглянуто моделі, методи та засоби побудови високо адаптивних розподілених систем. Для реалізації поставленої мети використано набір багаторазово використовуваних формальних моделей для визначення самоадаптивної поведінки та набір багаторазово використовуваних властивостей для перевірки бажаних властивостей, які має мати самоадаптивна поведінка. У недавньому дослідженні використання формальних методів для самоадаптивних систем були визначені як одна з найбільш використовуваних мов у цій галузі. Ми використовуємо часові автомати як формальну мову для специфікації самоадаптивної поведінки. Для визначення бажаних самоадаптивних властивостей використовується логіка обчислювального дерева (TCTL). Представлено формальні методи оптимізації та моделі верифікації рівня адаптивності систем, здійснено опис передумов використання самоадаптивного підходу та техніки MAPE-K. Виконано опис процесу імітаційного моделювання для дослідження надійності мобільного додатку, розглянуто сценарій використання мобільної платформи з елементами високо адаптивності з додаванням самоадаптивного шару. Запропоновані процеси самоадаптації з використанням верифікованих циклів MAPE-K.

У цій роботі запропоновані формальні шаблони в автономному режимі як під час початкової розробки системи, так і під час редизайну. Ця робота не зосереджена на використанні шаблонів під час виконання, але початкова робота показала, що шаблони також можна використовувати як основу для надання доказів для гарантій самоадаптивних систем під час виконання.

Особлива увага приділяється принципам проектування самоадаптації на основі архітектури, властивостям якості: надійність і відкритість, а також формальним мовам, які ми використовуємо для суворої специфікації та верифікації поведінки самоадаптивних системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. L. Adelman. “Experiments, quasi-experiments, and case studies: a review of empirical methods for evaluating decision support systems”. In: Systems, Man and Cybernetics, IEEE Transactions on 21.2 1991.

2. J. Almeida et al. “Resource management in the autonomic service-oriented architecture”. In: International Conference on Autonomic Computing. 2006.

3. R. Alur, C. Courcoubetis, and D. Dill. “Model-checking for real-time systems”. In: Logic in Computer Science, 1990. LICS ’90, Proceedings., Fifth Annual IEEE Symposium on e. 1990.

4. K. Angelopoulos, V.E.S. Souza, and J. Pimentel. “Requirements and architectural approaches to adaptive software systems: a comparative study”. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and SelfManaging Systems. IEEE Press. 2013.

5. C. Baier and J. Katoen. Principles of Model Checking. MIT Press, 2008.

6. D. Balasubramaniam et al. “Support for feedback and change in self-adaptive systems”. In: Workshop on Self-Healing systems. 2004.

7. L. Baresi and L. Pasquale. “Live goals for adaptive service compositions”. In: Software Engineering for Adaptive and Self-Managing Systems. 2010.

8. B. Bartels and M. Kleine. “A CSP-based framework for the specification, verification, and implementation of adaptive systems”. In: International Symposium on Software Engineering for Adaptive and Self-Managing Systems. 2011.

9. V.R. Basili, G. Caldiera, and H.D. Rombach. “Goal Question Metric Approach”. In: Encyclopedia of Soft. Engineering. 1994.

10. B. Becker et al. “Symbolic invariant verification for systems with dynamic structural adaptation”. In: 28th International Conference on Software Engineering. 2006.

11. M. Beer, M. Fasli, and D. Richards. Multi-Agent Systems for Education and Interactive Entertainment: Design, Use and Experience. IGI Global, 2010.

12. F. Bellifemine. “JADE: What it is and what it is next.” In: Workshop on Models and Methods of Analysis for Agent Based Systems (MMAABS). Vol. 2564. Lecture Notes in Computer Science. Genova: Springer Berlin / Heidelber, 2001.

13. J. Bengtsson and W. Yi. “Timed Automata: Semantics, Algorithms and Tools”. In: In Lecture Notes on Concurrency and Petri Nets. Ed. by W. Reisig and G. Rozenberg. LNCS 3098. Springer-Verlag, 2004.

14. D. Bianculli et al. “Specification patterns from research to industry: a case study in service-based applications”. In: Proc. of ICSE. Zurich, Switzerland: IEEE, 2012, pp. 968–976. ISBN: 978-1-4673-1067-3.

15. J. Bisbal and B. Cheng. “Resource-based approach to feature interaction in adaptive software”. In: Workshop on Self-Healing systems. 2004.

16. K. Biyani and S. Kulkarni. “Mixed-Mode Adaptation in Distributed Systems: A Case Study”. In: Software Engineering for Adaptive and Self-Managing Systems. 2007.

17. L. Bollen, M. Jansen, and S.C. Eimler. “Towards a Multichannel Input Dimension in Learning Scenarios with Mobile Devices”. In: WMUTE ’12. 2012.

18. R. Borges, A. d’Avila Garcez, and L. Lamb. “Integrating model verification and self-adaptation”. In: International Conference on Automated Software Engineering. 2010.

19. J. Bradbury et al. “A survey of self-management in dynamic software architecture specifications”. In: Workshop on Self-Managed Systems. 2004.

20. D. Breitgand, E. Henis, and O. Shehory. “Automated and adaptive threshold setting: Enabling technology for autonomy and self-management”. In: International Conference on Automatic Computing. 2005.

21. G. Brown et al. “Goal-oriented specification of adaptation requirements engineering in adaptive systems”. In: Software Engineering for Adaptive and SelfManaging Systems. 2006.

22. Y. Brun et al. “Engineering Self-Adaptive Systems through Feedback Loops”. In: Software Engineering for Self-Adaptive Systems. Springer-Verlag, 2009.

23. Bucchiarone et al. “Self-repairing systems modeling and verification using AGG”. In: Working IEEE/IFIP Conference on Software Architecture. 2009.

24. J.F. Calderón, M. Nussbaum, et al. “A Single-Display groupware Collaborative Language Laboratory”. Accepted in Interactive Learning Environments. 2014.

25. R. Calinescu and M. Kwiatkowska. “Using quantitative analysis to implement autonomic IT systems”. In: 31st International Conference on Software Engineering. 2009.

26. J. Cámara, C. Canal, and G. Salaun. “Behavioural self-adaptation of services in ubiquitous computing environments”. In: Software Engineering for Adaptive and Self-Managing Systems. 2009.

27. J. Cámara and R. de Lemos. “Evaluation of resilience in self-adaptive systems using probabilistic model-checking”. In: Software Engineering for Adaptive and Self-Managing Systems. 2012.

28. L. Cavallaro and E. di Nitto. “An approach to adapt service requests to actual service interfaces”. In: Software Engineering for Adaptive and Self-Managing Systems. 2008.

29. L. Cavallaro et al. “Synthesizing adapters for conversational web-services from their WSDL interface”. In: Software Engineering for Adaptive and Self-Managing Systems. 2010.

30. B.H.C. Cheng et al. “Software engineering for self-adaptive systems: A research roadmap”. In: Software Engineering for Self-Adaptive Systems. Vol. 5525. Lecture Notes in Computer Science. Springer-Verlag Berlin, Heidelberg, 2009.

31. P. Ciancarini, A. Omicini, and F. Zambonelli. “Multiagent System Engineering: The Coordination Viewpoint”. In: Intelligent Agents VI. Agent Theories, Architectures, and Languages. Vol. 1757. LNCS. Springer, 2000.

32. T.D. Cook, D.T. Campbell, and A. Day. Quasi-experimentation: Design & analysis issues for field settings. Boston: Houghton Mifflin College Div, 1979.

33. G. Di Marzo Serugendo, J. Fitzgerald, and Al. Romanovsky. “MetaSelf: an architecture and a development method for dependable self-* systems”. In: Proceedings of the 2010 ACM Symposium on Applied Computing. ACM. 2010.

34. Ebnenasir. “Designing run-time fault-tolerance using dynamic updates”. In: Software Engineering for Adaptive and Self-Managing Systems. 2007.

35. G. Edwards et al. “Architecture-driven self-adaptation and self-management in robotics systems”. In: Proc. of SEAMS. 2009. DOI: 10.1109/SEAMS.2009.

36. Epifani et al. “Model evolution by run-time parameter adaptation”. In: 31st International Conference on Software Engineering. 2009.

37. N. Esfahani, E. Kouroshfar, and S. Malek. “Taming uncertainty in self-adaptive software”. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ESEC/FSE ’11. Szeged, Hungary: ACM, 2011.

38. N. Esfahani and S. Malek. “On the role of architectural styles in improving the adaptation support of middleware platforms”. In: Proceedings of the 4th European conference on Software architecture. ECSA’10. Copenhagen, Denmark: SpringerVerlag, 2010.

39. J. Ferber. Multi-agent systems: an introduction to distributed artificial intelligence. Vol. 1. Addison-Wesley Reading, 1999.

40. Filieri et al. “Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements”. In: International Conference on Automated Software Engineering. 2011.

41. J. Fox. “A formal orchestration model for dynamically adaptable services with COWS”. In: International Conference on Adaptive and Self-Adaptive Systems and Applications. 2011.

42. O. Garcia et al. “CAFCLA, a framework to ease design, development and deployment AmI-based collaborative learning applications”. In: CISTI ’12. IEEE. 2012.

43. D. Garlan and B. Schmerl. “Model-based adaptation for self-healing systems”. In: Workshop on Self-Healing systems. 2002.

44. Georgiadis, J. Magee, and J. Kramer. “Self-organizing software architectures for distributed systems”. In: 1st Workshop on Self-healing Systems. ACM, 2002.

45. Ghezzi et al. “Managing non-functional uncertainty via model-driven adaptivity”. In: International Conference on Software Engineering. 2013.

46. H. Giese and W. Schäfer. “Model-Driven Development of Safe Self-optimizing Mechatronic Systems with MechatronicUML”. In: Assurances for Self-Adaptive Systems. Vol. 7740. LNCS. Springer, 2013.

47. Gil de la Iglesia. “Designing a Decentralized Distributed Self-Adaptive System in M-Learning Activities”. In: WMUTE ’12. 2012.

48. Gil de la Iglesia. Uncertainties in Mobile Learning applications: Software Architecture Challenges, (Licentiate Thesis). Linnaeus University, 2012.

49. Gil de la Iglesia, J. Andersson, and M. Milrad. “Enhancing Mobile Learning Activities by the Use of Mobile Virtual Devices – Some Design and Implementation Issues”. In: Proceedings of the 2nd International Conference on Intelligent Networking and Collaborative Systems (INCoS 2010). IEEE Computer Society, Nov. 2010.

50. D. Gil de la Iglesia, J. Andersson, and M. Milrad. “Mobile Virtual Devices for Collaborative M-Learning”. In: Proceedings of the 18th International Conference on Computers in Education (ICCE2010). Ed. by S. L. Wong. Putrajaya, Malaysia: Asia-Pacific Society for Computers in Education, 2010.

51. D. Gil de la Iglesia, M.U. Iftikhar, and D. Weyns. “Reusable Templates to Support the Design of MAPE-K Loops for Self-Adaptive Systems”. In: homepage.lnu.se/ staff/daweaa/MAPE-K-Templates.htm. 2009.

52. D. Gil de la Iglesia, M. Milrad, and J. Andersson. “Software Requirements to Support QoS in Collaborative M-Learning Activities”. In: Collaboration and Technology. Ed. by Valeria Herskovic et al. Vol. 7493. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.

53. D. Gil de la Iglesia and D. Weyns. “Guaranteeing robustness in a mobile learning application using formally verified MAPE loops”. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. 2013.

метадані

Заголовок

Імплементація формальних моделей оптимізації методів побудови адаптивних розподілених систем

Автор

Сак В.І. Науковий керівник / Експерт

підрозділ

King Danylo University

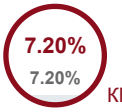
Тривога

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

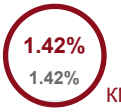
Заміна букв	ß	13
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	53

Обсяг знайдених подібностей

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



КП 1



КП 2

25
Довжина фрази для коефіцієнта подібності 2

14415
Кількість слів

110783
Кількість символів

Подібності за списком джерел

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

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	http://repository.ukd.edu.ua:8080/bitstream/handle/123456789/379/%D0%91%D0%B0%D0%BB%D0%B0%D0%BD%D1%8E%D0%BA_%D0%86_%D0%86_%D0%9A%D0%92%D0%90%D0%9B%D0%86%D0%A4%D0%86%D0%9A%D0%90%D0%A6%D0%86%D0%98%CC%86%D0%9D%D0%90_%D0%A0%D0%9E%D0%91%D0%9E%D0%A2%D0%90.pdf?sequence=1	42	0.29 %
2	http://repository.ukd.edu.ua/bitstream/handle/123456789/397/%D0%94%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%D0%BD%D0%B0_%D0%A2%D0%B0%D1%82%D0%B0%D1%80%D1%87%D1%83%D0%BA.pdf?sequence=1	39	0.27 %
3	http://repository.ukd.edu.ua/bitstream/handle/123456789/397/%D0%94%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%D0%BD%D0%B0_%D0%A2%D0%B0%D1%82%D0%B0%D1%80%D1%87%D1%83%D0%BA.pdf?sequence=1	35	0.24 %