МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 03.00.00.000 ПЗ

Група ІПм-23-3

Смачило Михайло

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Смачило Михайло Степанович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Оптимізація моделей виконання програмних додатків

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Смачило М.С.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Мельник Віталій Дмитрович, к.т.н., доцент

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

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

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

Нормоконтроль

доц. Вовк Р.Б.

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

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

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

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Смачилу Михайлу Степановичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Оптимізація моделей виконання програмних додатків ”

керівник проекту (роботи) Мельник Віталій Дмитрович, к.т.н., доцент

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

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

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

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

1. Аналіз предметної області розробки моделей програмних додатків
2. Методологія побудови моделі домену на основі артефактів
3. Реалізація моделі рівня обчислювальної платформи
4. Застосування параметричної моделі для оптимізації виконання програмних додатків

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

1. Архітектура програмного забезпечення для літографічних систем (рис. 1.1)
2. Підхід з використанням Y-діаграми (рис. 1.2)
3. Додаток, платформа та відображення – абстракції моделювання (рис. 1.3)
4. Підхід побудови моделі на основі парадигми Y-діаграми (рис. 1.4)
5. Представлення підходу для реконструкції моделей домену (рис. 2.1)

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз предметної області розробки моделей програмних додатків	29.09.2024	виконано
3	Методологія побудови моделі домену на основі артефактів	15.10.2024	виконано
4	Реалізація моделі рівня обчислювальної платформи	08.11.2024	виконано
5	Застосування параметричної моделі для оптимізації виконання програмних додатків	20.11.2024	виконано
6	Реалізація функціональності запропонованої інформаційної технології	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 77 с., 33 рис., 2 табл., 48 джерел.

Тема: Оптимізація моделей виконання програмних додатків

Об'єкт дослідження: процеси побудови та оптимізації моделей виконання програмних додатків для складних спеціалізованих систем.

Мета роботи: розробка та оптимізація методів моделювання програмних додатків для складних систем із застосуванням модельно-керованої архітектури (MDA), а також аналіз часових характеристик високотехнологічних систем для покращення їхньої продуктивності та конфігурації.

Предмет дослідження: методи моделювання та оптимізації програмних систем, зокрема використання модельно-керованої архітектури (MDA) для збору та трансформації даних.

Результати дослідження:

В роботі виконано розробку методику побудови моделей для складних систем, що передбачає використання модельно-керованого підходу з акцентом на взаємодію компонентів та функціональні характеристики системи

Висновок

Представлено формальний підхід до параметричного аналізу, який використовувався для визначення оптимальних параметрів та конфігурацій системи. Розглянуто теоретичні основи моделі та формалізовано алгоритми для обчислення ключових параметрів.

ПРОГРАМНІ ДОДАТКИ, ОПТИМІЗАЦІЯ КОНФІГУРАЦІЙ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, ПРОДУКТИВНІСТЬ СИСТЕМ, ПАРАМЕТРИЧНИЙ АНАЛІЗ, АНАЛІТИЧНИЙ ПЛАНУВАЛЬНИК, АЛГОРИТМИ МОДЕЛЮВАННЯ

ABSTRACT

Master Thesis: 77 pp., 33 fig., 2 tab., 48 sources.

Thesis Subject: Optimization of software application execution models

Research object: processes of building and optimizing software application execution models for complex specialized systems.

The purpose of the work: development and optimization of software application modeling methods for complex systems using model-driven architecture (MDA), as well as analysis of time characteristics of high-tech systems to improve their performance and configuration.

Research subject: methods of modeling and optimization of software systems, in particular, the use of model-driven architecture (MDA) for data collection and transformation.

Research results:

The work developed a methodology for building models for complex systems, which involves the use of a model-driven approach with an emphasis on the interaction of components and functional characteristics of the system

Conclusion

A formal approach to parametric analysis is presented, which was used to determine optimal system parameters and configurations. The theoretical foundations of the model are considered and the algorithms for calculating key parameters are formalized.

SOFTWARE APPLICATIONS, CONFIGURATION OPTIMIZATION, PARALLEL CALCULATIONS, SYSTEMS PERFORMANCE, PARAMETRIC ANALYSIS, ANALYTICAL PLANNER, SIMULATION ALGORITHMS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ASML - Advanced Semiconductor Materials Lithography

EMF - Eclipse Modeling Framework

LTTng - Linux Trace Toolkit next generating

RTT - Real-Time Tracing

ITAT - Timing Analysis Tool

SM - System Manager

HSI - Hardware-Software Interface

XSD - XML schema denition

FCN - Function completion notication

CFS - Completely Fair Scheduler

ВСТУП

Актуальність теми.

Сучасні високотехнологічні системи стають дедалі складнішими та вимагають ефективних підходів до їхнього моделювання та оптимізації. Традиційні методи побудови та управління програмними додатками не завжди здатні забезпечити необхідний рівень продуктивності та адаптивності до змінних умов експлуатації. Зростає необхідність у застосуванні інноваційних методів, які дозволяють ефективно аналізувати та прогнозувати поведінку складних систем, забезпечуючи їхню оптимізацію за ключовими параметрами, такими як швидкодія, ресурсоємність і надійність.

Запропоновані в даному дослідженні методи моделювання, засновані на модельно-керованій архітектурі (MDA), дозволяють збирати та аналізувати дані з реальних працюючих систем, що значно підвищує точність прогнозування та планування їхньої роботи. Крім того, розробка і впровадження алгоритмів параметричного аналізу дозволяє оптимізувати конфігурації складних програмних систем, знижуючи накладні витрати і підвищуючи ефективність використання ресурсів.

Актуальність роботи також обумовлена необхідністю підвищення продуктивності багатопроцесорних систем, які відіграють ключову роль у сучасних високотехнологічних рішеннях. Оптимізація конфігурацій багатопроцесорних систем за допомогою розроблених моделей і алгоритмів дозволить значно покращити їхню роботу в умовах реального часу.

Сучасні системи літографії швидко розвиваються з точки зору технології та складності. Динамічна природа цих високотехнологічних систем призводить до труднощів у прогнозуванні продуктивності. На відміну від традиційних систем, системи літографії є складними та важко прогнозувати їх продуктивність. Для прогнозування продуктивності системи важливо точно витягувати відповідну інформацію з системи.

Артефакти, присутні в системі, схильні до частих змін, що призводить до її невідповідності. Динамічну поведінку можна досягти шляхом вилучення відповідної інформації з працюючої машини. Далі інформація реконструюється та реалізується у вигляді доменних моделей. Реконструйовані моделі можна використовувати для аналізу та покращення часової продуктивності.

Метою даної роботи є прогнозування часової продуктивності та дослідження можливості альтернативних конфігурацій. Продуктивність системи можна прогнозувати шляхом вилучення відповідної інформації. Підхід на основі моделювання розроблений для отримання артефактів з системи та їх класифікації як різних доменних моделей. Крім того, розроблено аналітичний планувальник, який представляє поведінку працюючої машини та імітує планувальник Linux. Як результат, створюються доменні мови для кожного рівня, щоб витягувати артефакти з працюючої машини. За допомогою артефактів прогнознувальна здатність аналітичного планувальника визначається шляхом валідації щодо вимірювання в реальному часі. Аналітичний планувальник може бути використаний для дослідження майбутнього шляхом визначення найкращих конфігурацій щодо кількості ядер у процесорі та пріоритету застосунків. Розроблений планувальник може бути застосований до систем літографії, а прогнозованість може бути розширена шляхом консолідації архітектурного типу процесора для застосування на подібних платформах.

Таким чином, дослідження спрямоване на вирішення нагальних завдань, пов'язаних з підвищенням продуктивності та оптимізацією складних програмних систем, що є вкрай важливим у контексті розвитку сучасних технологій та інновацій у різних галузях промисловості.

Мета дослідження - розробка та оптимізація методів моделювання програмних додатків для складних систем із застосуванням модельно-керованої архітектури (MDA), а також аналіз часових характеристик

високотехнологічних систем для покращення їхньої продуктивності та конфігурації.

Об'єкт дослідження - процеси побудови та оптимізації моделей виконання програмних додатків для складних спеціалізованих систем.

Предмет дослідження - методи моделювання та оптимізації програмних систем, зокрема використання модельно-керованої архітектури (MDA) для збору та трансформації даних.

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

1. Провести огляд наукових джерел та сучасних підходів до моделювання програмних додатків для складних систем, зокрема дослідити методи модульного підходу та модельно-керованої архітектури (MDA).

2. Розробити методику побудови моделей для складних систем, що передбачає використання модельно-керованого підходу з акцентом на взаємодію компонентів та функціональні характеристики системи.

3. Формалізувати алгоритми для параметричного аналізу, спрямовані на визначення оптимальних параметрів конфігурації програмних додатків.

4. Дослідити аналітичний планувальник для оцінки продуктивності реконструйованих моделей та провести порівняння отриманих результатів з емпіричними даними для підтвердження точності моделей.

5. Підготувати рекомендації щодо використання модельно-керованої архітектури та розроблених методів у практичних додатках для оптимізації програмних систем.

Методи дослідження.

В роботі застосована модельно-керована архітектура (MDA) для побудови та аналізу моделей програмних систем; алгоритмічний аналіз для оптимізації конфігурації систем; параметричний аналіз для визначення оптимальних характеристик системи; верифікаційні методи для порівняння прогнозованих та емпіричних даних; огляд наукової літератури для вивчення сучасних підходів до моделювання.

Наукова новизна отриманих результатів полягає в розробці нових підходів до моделювання складних систем із застосуванням модельно-керованої архітектури (MDA), що дозволяють ефективно збирати дані з працюючих систем та перетворювати їх для подальшого аналізу

Практичне значення магістерської роботи полягає в розробці методики аналізу часових характеристик систем, що дозволяє підвищити точність прогнозування та покращити продуктивність систем за рахунок оптимального розподілу ресурсів.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 77 сторінок, і містить 33 рисунки, 2 таблиці, список використаних джерел із 48 найменувань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ МОДЕЛЕЙ ПРОГРАМНИХ ДОДАТКІВ

1.1. Опис архітектури та особливостей програмного забезпечення для складних спеціалізованих систем

Складні спеціалізовані системи – це інженерні системи, розроблені для виконання конкретних завдань у складних і часто мінливих умовах. Вони характеризуються високим рівнем інтеграції різних компонентів, як програмних, так і апаратних, і мають високу ступінь автоматизації.

Від інших систем вони відрізняються наступними характеристиками:

- Спеціалізація. На відміну від універсальних систем, спеціалізовані системи оптимізовані для виконання конкретного набору задач.

- Складність. Вони складаються з великої кількості взаємопов'язаних компонентів, що ускладнюють їх проектування, розробку та експлуатацію.

- Інтеграція. Об'єднують різні фізичні системи, програмне забезпечення та мережі.

- Автоматизація. Багато функцій виконуються автоматично, з мінімальною участю людини.

- Адаптивність. Здатні адаптуватися до змін у навколишньому середовищі та виконувати свої функції в динамічних умовах.

Системи літографії – це високотехнологічні інструменти, які використовуються для створення надзвичайно маленьких візерунків на поверхні матеріалу, зазвичай на кремнієвих пластинах. Ці візерунки є основою для виробництва мікросхем, які використовуються в комп'ютерах, смартфонах, та інших електронних пристроях.

ASML є провідним виробником літографічних систем, які відповідають за виробництво мікросхем, таких як процесори, мікросхеми пам’яті та індивідуальні мікросхеми. Це складні системи з інтеграцією кількох підсистем, які забезпечують такі функції, як сканування, експонування та

калібрування пластин. Функціональні можливості системи реалізуються спеціальними програмними додатками, розгорнутими на обчислювальній платформі.

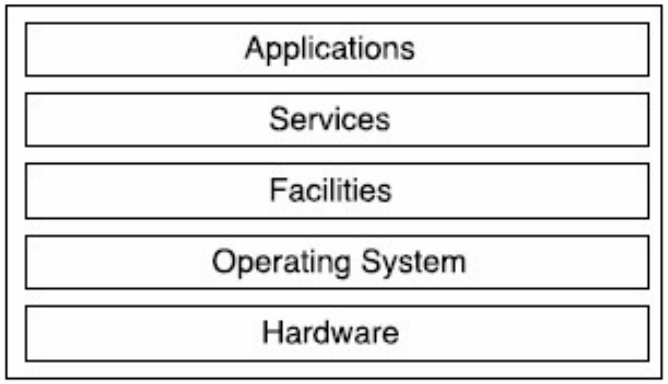


Рис. 1.1. Архітектура програмного забезпечення для літографічних систем

Архітектура програмного забезпечення для сучасних систем літографії складається з п’яти рівнів, показаних на рисунку 1.1. Верхній рівень на рисунку представляє прикладний рівень, який відповідає за взаємодію людини з машиною та керування системою. Сервісний рівень реалізує набір багаторазово використовуваних функцій. Рівень засобів діє як проміжне програмне забезпечення, яке є мостом між рівнем програми та рівнем операційної системи. Рівень операційної системи забезпечує управління та планування доступних обчислювальних ресурсів. Щоб зрозуміти систему, потрібно зрозуміти її рівні. Ці шари взаємопов’язані один з одним і впливають на загальну поведінку. Ці спостереження дають розуміння для аналізу та прогнозування ефективності.

Розподілений характер обчислювальної платформи збільшує складність прогнозування продуктивності. Обчислювальні платформи складаються з хостів, і кожен хост відповідає за такі функції, як сканування, експозиція, калібрування та вирівнювання. Ці хости з'єднані з іншими хостами через Ethernet. Хости складаються з процесорів, з'єднань і пам'яті. Процесори

можуть мати кілька ядер і можливості гіперпотокової обробки. З кількома обчислювальними блоками збільшується простір для розгортання програми. Наприклад, N ресурсів і M програм призводять до N M можливих розгортань. Тому визначити оптимальну конфігурацію стає все важче.

Взаємодія між додатком і рівнем обчислювальної платформи аналізується, щоб зрозуміти розподілену систему. Процес призначення додатків обчислювальній платформі називається відображенням. Коли кількість програм збільшується, вони можуть бути зіставлені з тією самою обчислювальною платформою, що називається мультиплексуванням.

Підвищення точності та продуктивності літографічних систем призводить до збільшення кількості додатків корекції та контролю, а також до скорочення бюджету часу. Отже, стає важче відповідати вимогам щодо часу та передбачити продуктивність системи.

Обчислювальна платформа має бути адаптована до нових вимог. Адаптація передбачає перепроектування обчислювальної платформи та оптимальне відображення програми на обчислювальній платформі. Щоб знизити витрати, спостерігається збільшення мультиплексування на дефіцитних ресурсах платформи. Мультиплексування ще більше збільшує загальну складність і ускладнює прийняття проектних рішень при зміні вимог.

Щоб спрогнозувати продуктивність літографічної системи, враховуючи альтернативне розгортання та конфігурації платформи, розробники повинні відповісти на наступні запитання.

1. Як розгортати додаток на платформі?

2. Які типи процесорів необхідно використовувати та скільки процесорів потрібно?

3. Які типи накопичувачів слід використовувати?

4. Які типи з’єднань потрібні та яка пропускна здатність потрібна для складної системи ?

1.2. Методика побудови моделі для складної системи

Поведінка системи надто складна, щоб створювати моделі вручну. Це пов’язано зі збільшенням кількості заявок із обмеженим бюджетом. Тому нам потрібен автоматизований реконструктор, щоб ефективно розробляти моделі з активної машини. Також необхідно розробити модель аналізу продуктивності, щоб передбачити часові характеристики альтернативних конфігурацій і визначити оптимальну. Дві важливі цілі, які необхідно досягти:

1. За допомогою інтегрованого середовища розробки (IDE) розробляється автоматизований реконструктор для отримання таких моделей домену:

- Формальна модель предметної області для компонентної архітектури логіки програми. Модель включає в себе взаємодію різних компонентів усередині моделі.
- Формальна модель предметної області для ресурсів обчислювальної платформи. Моделі включають багатопроцесорні, багатоядерні та гіперпотокові можливості. Фізичні з’єднання та блоки зберігання також присутні.
- Формальна модель для розгортання (включаючи планування) програми на ресурсах обчислювальної платформи.

2. Модель аналізу продуктивності Модель аналізу продуктивності використовується для аналізу властивостей синхронізації для різних зіставлень програми з різними обчислювальними платформами. Модель аналізу продуктивності використовується для дослідження альтернативних конфігурацій проекту. Модель також дає уявлення про потенційні вузькі місця та ефективне використання ресурсів платформи.

Вимоги до часу досліджуються шляхом розуміння проблем обчислювальної платформи, тобто програми, платформи та відображення. Проблеми обчислювальної платформи можна розділити та зрозуміти за

допомогою підходу Y-діаграми [1]. Як зазначено в [1], « підхід Y-діаграми — це методологія, яка надає розробникам кількісні дані, отримані шляхом аналізу продуктивності архітектури для заданого набору програм. ” Підхід Y-діаграми показано на рисунку 1.2.

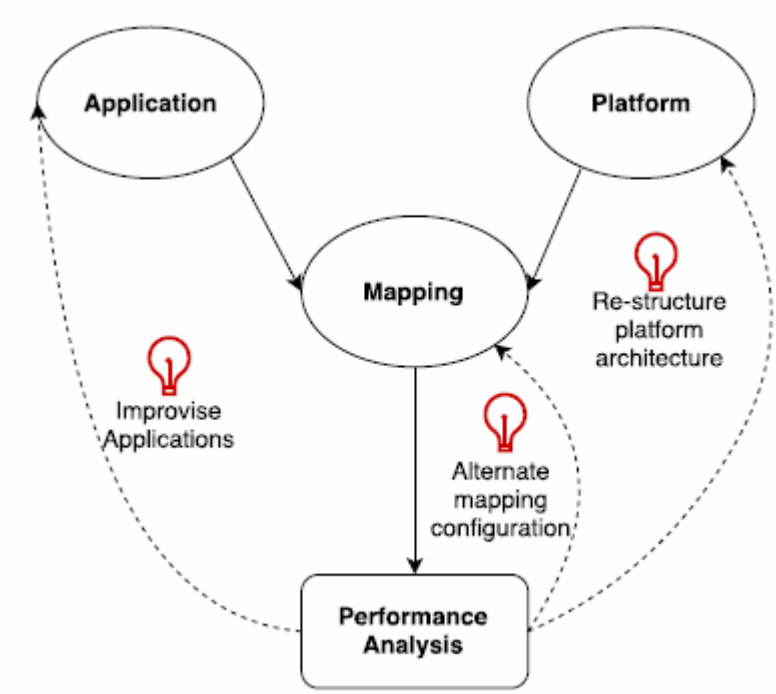


Рис. 1.2. Підхід з використанням Y-діаграми [1]

Цей підхід потребує програми та моделі платформи, а також моделі відображення для їх явного планування. Вихід складається з даних продуктивності, таких як пропускна здатність, час і використання ресурсів. Дизайнери можуть інтерпретувати результати та покращувати продуктивність за допомогою серії ітерацій.

Y-діаграма - це візуальний інструмент, запропонований Бартом Кієнхюйсом, який використовується для моделювання та аналізу складних систем, особливо в контексті вбудованих систем. Вона дозволяє представити систему як сукупність трьох взаємопов'язаних аспектів:

- Data: Дані, які обробляє система.

- Behavior: Поведінка системи, тобто те, як вона трансформує дані.

- Environment: Оточення системи, яке впливає на її роботу і з яким система взаємодіє.

Y-діаграма має форму літери "Y". Кожна гілка відповідає одному з трьох аспектів системи:

- Ліва гілка (Data): Тут описуються типи даних, які обробляє система, їх формат, структура та джерела.

- Середня гілка (Behavior): Ця гілка містить інформацію про алгоритми, які виконуються системою, логіку прийняття рішень, а також про взаємодію між різними компонентами системи.

- Права гілка (Environment): Тут описуються зовнішні фактори, які впливають на систему, такі як обмеження за ресурсами, фізичні закони, користувачі та інші системи.

Переваги використання Y-діаграм:

- Візуалізація: Y-діаграми дозволяють наочно представити складні системи, що полегшує розуміння їх структури та функціонування.

- Системний підхід: Заохочують до системного мислення, враховуючи всі аспекти системи.

- Комунікація: Сприяють ефективній комунікації між різними зацікавленими сторонами проекту.

- Аналіз: Допомагають виявити потенційні проблеми та вузькі місця в системі.

- Дизайн: Можуть використовуватися для розробки нових систем або модернізації існуючих.

Одним із способів зрозуміти продуктивність активної машини є абстрагування архітектури програмного забезпечення в термінах абстракції Y-діаграми. На рисунку 1.3 показано абстракцію Y-діаграми архітектури програмного забезпечення. Нижня частина показує архітектуру програмного забезпечення, як пояснюється в попередньому розділі. Три верхні частини – це необхідні абстракції Y-діаграми для аналізу продуктивності та оптимізації.

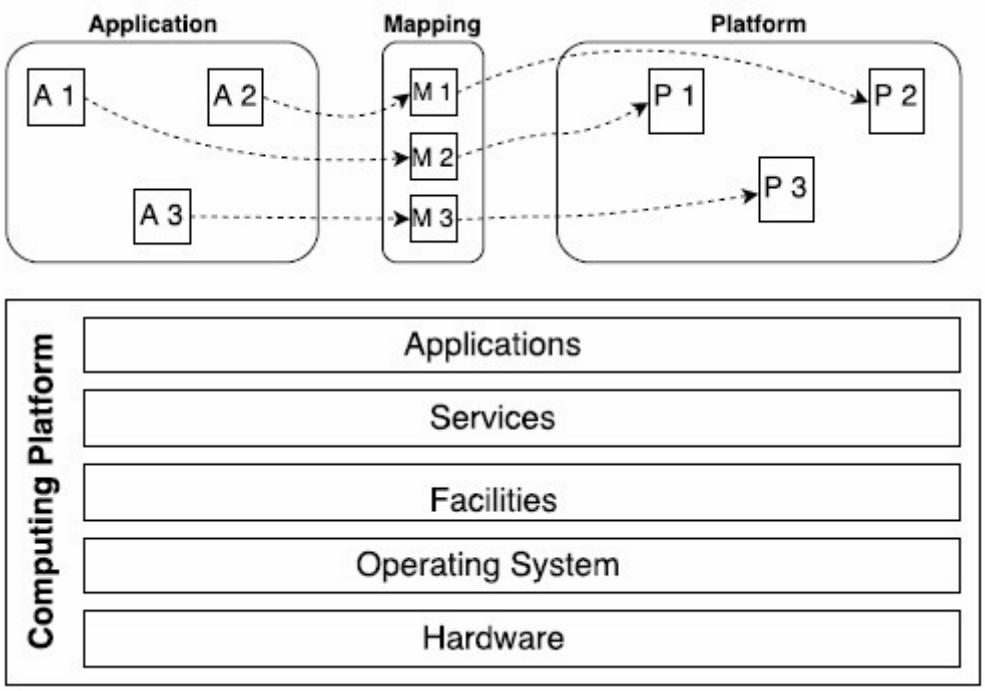


Рис. 1.3. Додаток, платформа та відображення – абстракції моделювання

Верхня ліва частина представляє модель додатка, що складається з взаємодіючих компонентів додатка. Права верхня частина представляє модель платформи, що складається з фізичних компонентів, таких як процесори та комунікаційна мережа. Верхня середня частина представляє модель відображення, яка забезпечує явне відображення програми в моделі платформи.

Компанія ASML працює над проектом під назвою Concerto. Метою цього проекту є прогнозування продуктивності синхронізації за допомогою модельного підходу на основі парадигми Y-діаграми. Детальний підхід до проекту Concerto показано на рисунку 1.4. Першим кроком є отримання відповідних артефактів, таких як сліди виконання та виконувані файли. Вони отримані з працюючої машини, а також з архіву програмного забезпечення. Інформація про запуск і завершення процесів витягується з файлів конфігурації системи. Поведінку процесу можна відстежити за допомогою інструментів профілювання системи. Показаний підхід дає уявлення про

розгортання процесу на обчислювальній платформі та використання її ресурсів.

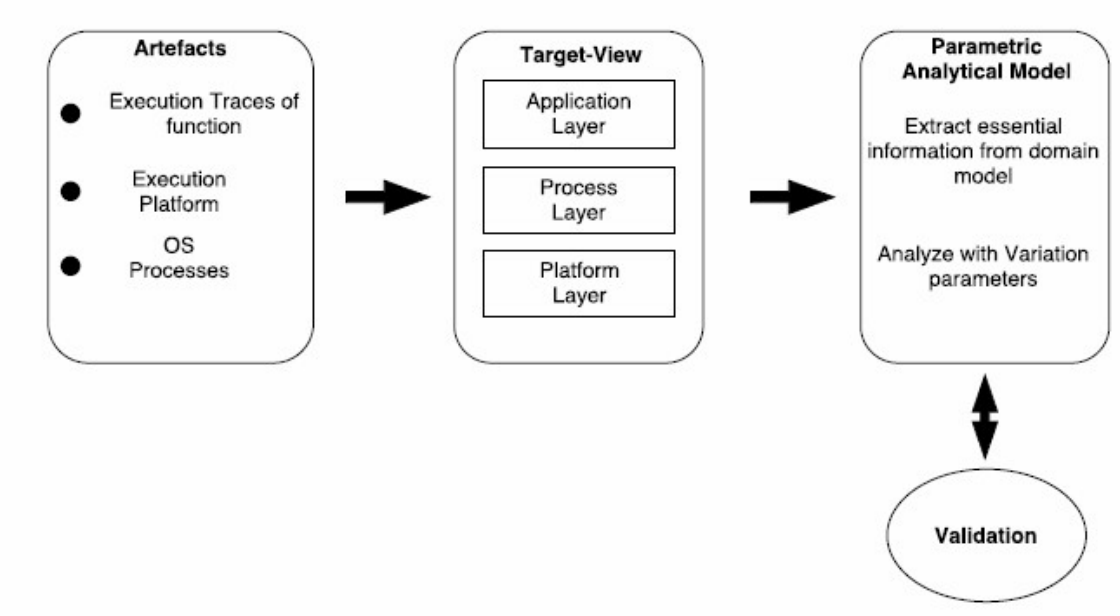


Рис. 1.4. Підхід побудови моделі на основі парадигми Y-діаграми

Після визначення артефактів цільове зображення реконструюється для формалізації артефактів у різних шарах. Цільове представлення представляє формалізовані моделі домену прикладного рівня, рівня процесу та рівня обчислювальної платформи. Прикладний рівень містить високорівневий опис додатків. Рівень процесу описує залежності та взаємодію між процесами. Операційна система (ОС) і рівень апаратного забезпечення описують ОС поверх апаратних компонентів, таких як багатопроцесорні процесори, з'єднання та пристрої зберігання.

Спостереження цільового перегляду використовуються для отримання необхідної інформації для параметричної аналітичної моделі та її перевірки. Декомпозиція виконується для поділу проблем, як зазначено в Y-діаграмі. Ця модель розроблена з варіюванням параметрів, тобто кількості ядер, тактової частоти та пріоритету процесів. Перевірка, виконана на наборах тестів, допомагає визначити його прогностичну силу. Після перевірки параметри можна налаштувати для вивчення різних альтернатив розгортання та

платформи. Отримані результати використовуються для оптимізації функціональності програми, відображення та платформи на основі підходу Y-діаграми.

1.3. Аналіз наукових джерел по темі дослідження

Аналіз розподілених систем та їх оптимізація є активною сферою досліджень. Добре відома парадигма Y-діаграми [1], запропонована Bart Kienhuis et. al — це метод проектування програмованих вбудованих систем. Він займається оптимізацією розподілених систем шляхом аналізу конфігурацій і визначення альтернативних конфігурацій шляхом зміни програми, відображення та обчислювальної платформи. Крім того, підхід підтримує дослідження простору проектування та визначення компромісів між продуктивністю, вартістю та точністю. Парадигма Y-діаграми дотримується інженерної структури, керованої моделлю [2], явно створюючи моделі шляхом поділу проблем, тобто модель додатків, модель відображення та модель платформи, і створюючи моделі для кожної проблеми, щоб зрозуміти поведінку на абстрактному рівні . У цій дисертації ми взяли [1] і [2], щоб розділити питання літографічних систем і передбачити оптимальну конфігурацію розгортання.

Хоча вже було проведено значні дослідження аналізу розподілених систем, лише деякі дослідження зосереджені на отриманні конкретної інформації про відповідні проблеми. Наприклад, зібрати артефакти програмного забезпечення є складною справою, оскільки архітектура програмного забезпечення з часом змінюється в будь-якій комп’ютерній системі, а з розподіленими системами стає все складніше. Вилучення артефактів з активної машини є формою зворотного проектування. Початкові дослідження в цій галузі [3] пропонують підхід до отримання архітектури програмного забезпечення з точки зору логічної архітектури та фізичної архітектури. [4] пропонує генерацію програмних артефактів, керовану

моделлю функцій, шляхом створення моделей для загальних і змінних функцій. [3] було адаптовано на початковому етапі дослідження шляхом розуміння обчислювальної платформи працюючої машини як логічної та фізичної моделі. Однак вони не змогли розглянути тонкий зв’язок між артефактами та похідною архітектурою на наступних етапах.

У [5] van Deursen та ін. запропонував методологію реконструкції під назвою Symphony, яка забезпечує реконструкцію архітектури програмного забезпечення на основі перегляду. Цей підхід відповідає стандарту IEEE 1471, створюючи репрезентацію системи з точки зору відповідного набору проблем. У документі також обговорюється реконструкція архітектури шляхом виявлення проблеми від зацікавлених сторін, визначення вихідного подання та перетворення їх у цільове подання за допомогою доступних інструментів моделювання, таких як UML та Eclipse Modeling Framework (EMF). Документ адаптує підхід шляхом збору відповідних даних для реалізації вихідного вигляду з системи та виконує реконструкцію за допомогою доступних інструментів. Модель логічного висновку знань допомагає витягувати та комбінувати інформацію для отримання цільового уявлення. Цільове подання далі використовується параметричною аналітичною моделлю для аналізу продуктивності синхронізації. У цій дипломній роботі ми дотримуємося методології Symphony, визначаючи джерело та перетворюючи його на цільовий вигляд.

Поєднання Y-діаграм та штучного інтелекту (ШІ) відкриває нові горизонти в оптимізації процесів літографії. Це потужний тандем, який дозволяє досягти високої точності, ефективності та адаптивності у виробництві мікросхем.

Y-діаграма забезпечує структурований опис процесу літографії, визначаючи взаємозв'язки між даними, поведінкою та оточенням. Ця модель стає фундаментом для подальшої роботи зі ШІ. Алгоритми машинного навчання, такі як нейронні мережі, можуть аналізувати великі обсяги даних, отриманих з Y-діаграми, виявляти закономірності та прогнозувати поведінку

системи. На основі отриманих прогнозів ШІ може розраховувати оптимальні параметри процесу, мінімізуючи вартість, максимізуючи врожайність і покращуючи якість продукції.

Переваги такого підходу:

- Висока точність і ефективність: ШІ дозволяє приймати рішення на основі великого обсягу даних, що забезпечує високу точність і ефективність процесів.

- Адаптивність: Системи, побудовані на основі ШІ, можуть адаптуватися до змін умов виробництва і виявляти нові закономірності.

- Скорочення часу розробки: Автоматизація багатьох рутинних операцій дозволяє значно скоротити час розробки нових продуктів.

Очікується, що використання ШІ в літографії буде розширюватися, охоплюючи все більше процесів і етапів виробництва. Зараз будуть розроблятися нові алгоритми машинного навчання, спеціально адаптовані для задач літографії. ШІ буде інтегруватися з іншими технологіями, такими як Інтернет речей, хмарні обчислення та цифровими двійниками. Використання Y-діаграм у поєднанні зі штучним інтелектом відкриває нові можливості для оптимізації процесів літографії та створення більш досконалих мікросхем.

Висновки до розділу

В даному розділі проведено всебічний аналіз предметної області розробки моделей для складних програмних систем. Детально проаналізовано архітектуру програмного забезпечення для складних спеціалізованих систем. Визначено ключові компоненти, що впливають на ефективність та масштабованість систем, а також специфічні вимоги, зумовлені природою складних програмних рішень.

РОЗДІЛ 2. МЕТОДОЛОГІЯ ПОБУДОВА МОДЕЛІ ДОМЕНУ НА ОСНОВІ АРТЕФАКТІВ

2.1. Процес збору даних для побудови моделі

Хороша модель абстрагується від нерелевантної поведінки системи. Наша мета полягає в тому, щоб автоматично побудувати таку модель шляхом вилучення інформації з працюючої машини та її аналізу. Ми реконструюємо артефакти, з яких походить модель, у різних видах. На рисунку 2.1 показано підхід до реконструкції моделей домену з артефактів. Вихідний вигляд складається з артефактів, отриманих від працюючої машини. Моделі цільового перегляду формалізовані та відокремлені на різних рівнях як моделі домену.

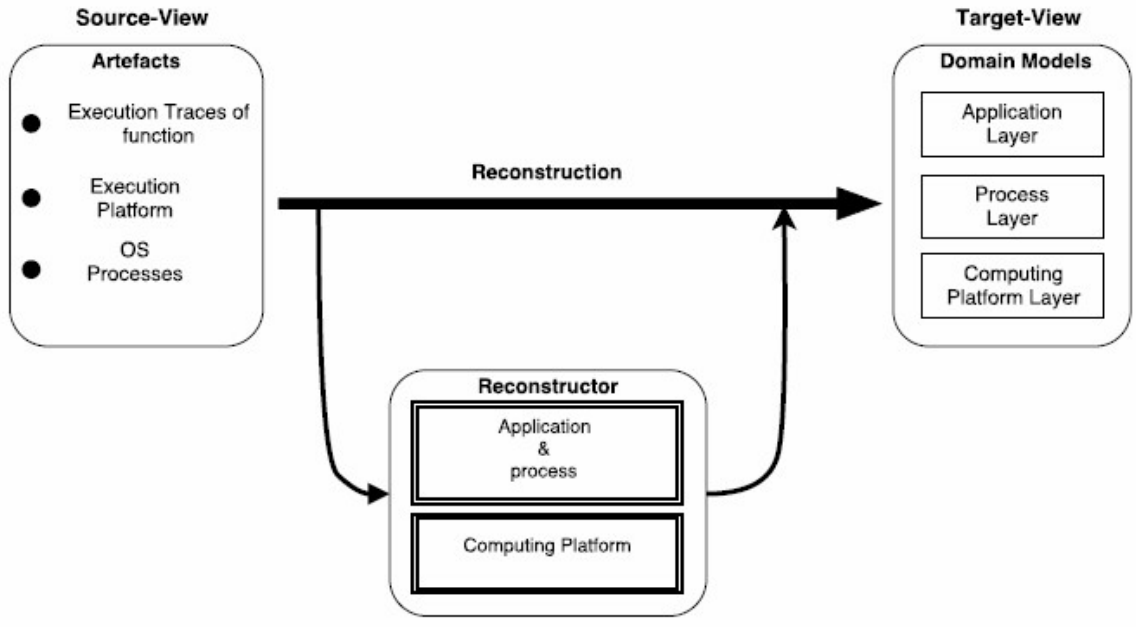


Рис. 2.1. Представлення підходу для реконструкції моделей домену

Щоб формалізувати дані, отримані від працюючої машини, у моделі домену, спочатку потрібно визначити артефакти, які відповідають машині. Програмне забезпечення сучасних систем літографії схильне до частих змін.

Ці зміни не часто оновлюються в існуючій архітектурі програмного забезпечення, залишаючи артефакти несумісними з поведінкою в реальному часі. Артефакти допомагають описати функції, архітектуру та дизайн програмного забезпечення, які зрештою визначають загальну поведінку системи. Для цієї дисертації відповідними артефактами, які витягуються з машини, є сліди виконання, файли конфігурації системи та файли системного журналу.

Поведінку обчислювальної платформи можна частково зрозуміти з послідовностей виконання та відображення завдань. У Linux користувач може керувати послідовністю виконання за допомогою спеціальних файлів, відомих як файли конфігурації системи. Файли конфігурації системи містять таку інформацію:

• Інформація про запуск і завершення завдань.

• Список завдань.

• Послідовність завдань і сценарій відображення.

Ця інформація може бути використана для визначення поведінки програми та процесу.

Моніторинг, керований подіями, є найпоширенішим підходом для спостереження за поведінкою розподілених систем і відстежується з точки зору набору простих подій, що представляють найнижчу спостережувану активність системи, а саме часові позначки потоків і подій. Відстеження виконання збирає час виконання від запущеної машини, який можна використовувати для аналізу поведінки та відповідної оптимізації. Спостереження за системою може здійснюватися на рівні процесу або на рівні ядра.

Linux Trace Toolkit next generating (LTTng) — це інструмент, який записує взаємодію різних модулів ядра в журнал подій. LTTng використовується для моніторингу ядра Linux у цьому проекті. Крім того, цікаву функцію можна відстежувати, записуючи буфери трасування до та після використання трасування в реальному часі (RTT). ASML також

створила спеціальний інструмент відстеження [6] для розуміння поведінки процесів та їх комунікацій, відомий як Inter-process Timing Analysis Tool (ITAT). ITAT створено за допомогою LTTng, RTT і Trace Compass. Вони розроблені як плагіни в інтегрованому середовищі розробки eclipse, які візуалізують результат трасування функції та LTTng, які надаються як вхідні дані для отримання візуалізації. Функціональне відстеження отримується з програмного середовища моделювання ASML (dev bench) шляхом увімкнення компонентів, які беруть участь у цікавій послідовності. Крім того, LTTng використовується для відстеження планувальника Linux , який включає події щодо процесів. Нарешті, результати додаються до спеціального інструменту аналізу платформи ASML для візуалізації графа викликів, який забезпечує розуміння взаємозв'язку між функціями.

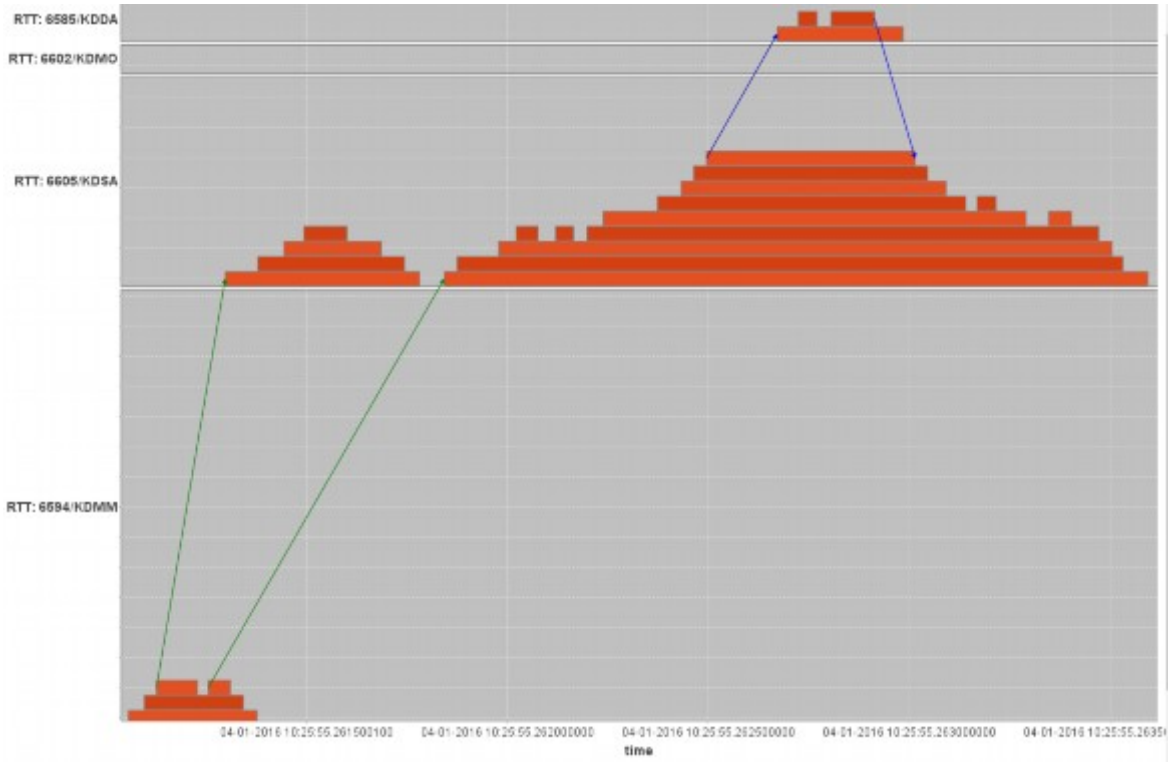


Рис. 2.2. Візуалізація процесу ITAT

На рисунку 2.2 показано візуалізацію в ITAT. Графік викликів показує віддалений виклик процедури від клієнта, який викликає функцію на сервері. Щоразу, коли запит було оброблено, сервер відповідає клієнту відповіддю. У

ITAT виклик функції з синьою стрілкою вважається блокуючим, а зелена стрілка — неблокуючим.

Системи літографії складаються з різних подій, які відбуваються в послідовності. Ці складні системи працюють у реальному часі завдяки стохастичній поведінці. Ці події можуть відбуватися послідовно або паралельно для виконання певного завдання. Виклики функцій реєструються в журналах під час входу та виходу з них. Інформація також містить повідомлення про запуск, системні зміни та завершення процесів. Події записуються у файл журналу з позначкою часу, відомі як файли системного журналу.

2.2. Дослідження процесу трансформації моделі

Реконструкція надає абстраговану інформацію про працюючу машину з артефактів. Надто детальна інформація не допоможе ні для загального аналізу, ні для визначення оптимального рішення. Тому артефакти формалізовані у вигляді моделей домену. Як показано на рисунку 2.1, початковий підхід передбачає перетворення артефактів у моделі домену. Ми називаємо цей крок «формалізацією». Формалізації допомагають отримати абстрактне уявлення про програму та обчислювальну платформу. Ці формалізації створюються за допомогою предметно-спеціальних мовних інструментів для перетворення цих артефактів у відповідні моделі. Це включає в себе трансформацію моделей домену, що стосуються рівня процесу, апаратного забезпечення та рівня ОС. У наступних розділах розглядається технологічна підтримка та методологія трансформації.

2.2.1. Середовище розробки

Розробка та формалізація моделей домену здійснюється в інтегрованому середовищі розробки (IDE), яке називається Eclipse. Eclipse широко використовується для програмування та моделювання програмного

забезпечення за допомогою плагінів. Наступні плагіни з Eclipse IDE є основними компонентами цього проекту.

Eclipse Modeling Framework (EMF) — це набір плагінів, які дозволяють створювати структуровані доменні моделі у формі метамоделей. EMF дозволяє створювати мета-модель різними способами, такими як уніфікована мова моделювання та розширювана мова розмітки. Цей проект вимагає, щоб мета-модель була визначена у формі файлу опису ecore. Файл опису ecore містить визначення класу, атрибута, посилання та типу даних. Пізніше модель gen, пов'язану з файлом ecore, можна використовувати для генерації коду для моделі EMF.

EMF Refactor в основному складається з шести компонентів із двох вимірів: щодо основних функціональних можливостей (обчислення показників моделі, виявлення запахів і виконання рефакторингу) для кожного є модуль програми . Подібним чином існують три модулі специфікації для генерації плагінів метрики, запаху та рефакторингу, що містять код Java, який може використовуватися відповідним модулем програми. Для простоти ми далі називаємо ці плагіни користувацькими плагінами QA . Ми починаємо з опису розміру специфікації. На рис. 2.3. показано архітектуру модуля специфікації з використанням компонентної моделі UML.

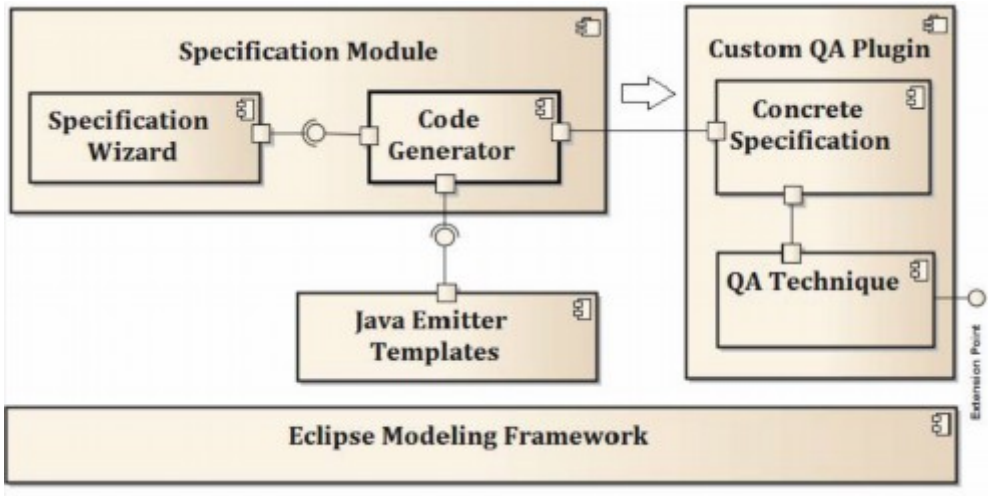


Рис. 2.3. Архітектура модуля специфікації з використанням компонентної моделі

Модуль специфікації забезпечує генерацію настроюваних плагінів QA, що містять специфічний для метрики, запаху або рефакторингу код Java. Використовуючи технологію плагіна Eclipse, можна надати бібліотеки, що складаються з методів забезпечення якості моделі. Отже, вже реалізовані методики можна використовувати повторно.

Xtext — це основа для розробки доменно-спеціальних мов. В даному проекті певну інформацію в артефактах потрібно перетворити на моделі домену. Xtext визначає граматику для файлу конфігурації системи та підтримує генерацію моделі предметної області завдання SM у формі метамоделі.

XML-схема використовується для визначення та опису XML-документа за допомогою визначених компонентів схеми для обмеження та визначення зв’язку між вказаними частинами, тобто типами даних, елементами та атрибутами зі значеннями. Мова визначення XML-схеми структурує XML-документи та формалізує їх, щоб забезпечити корисний рівень перевірки обмежень.

У цьому проекті XSD допомагає реконструювати модель із інструменту платформи ASML, який має форму XML. Це допомагає формалізувати та структурувати інструмент платформи, який представляє апаратний рівень.

2.2.2. Артефакти до моделі домену

Трансформація артефактів у модель домену реалізується за допомогою спеціального підходу для кожного шару. Оскільки артефакти неоднакові, підхід до перетворення артефактів у модель домену залежить від вихідного артефакту. У цьому розділі пояснюється підхід до формалізації моделей домену процесу та рівня ОС і обладнання.

Файли завдань System Manager (SM) — це файли конфігурації в машині ASML TWINSCAN. Як обговорювалося в попередньому розділі, ці файли відповідають за запуск і завершення різних завдань у літографічному апараті. Файли завдань SM розробляються з використанням спеціального синтаксису

ASML і аналізуються за допомогою файлу Yacc. Щоб перетворити файли завдань SM у моделі предметної області, необхідну інформацію з файлів завдань SM потрібно витягти. Вилучення здійснюється за допомогою спеціального синтаксичного аналізатора, після чого проаналізована інформація перетворюється на конкретні моделі домену. У цьому випадку парсер створюється за допомогою інструменту Xtext.

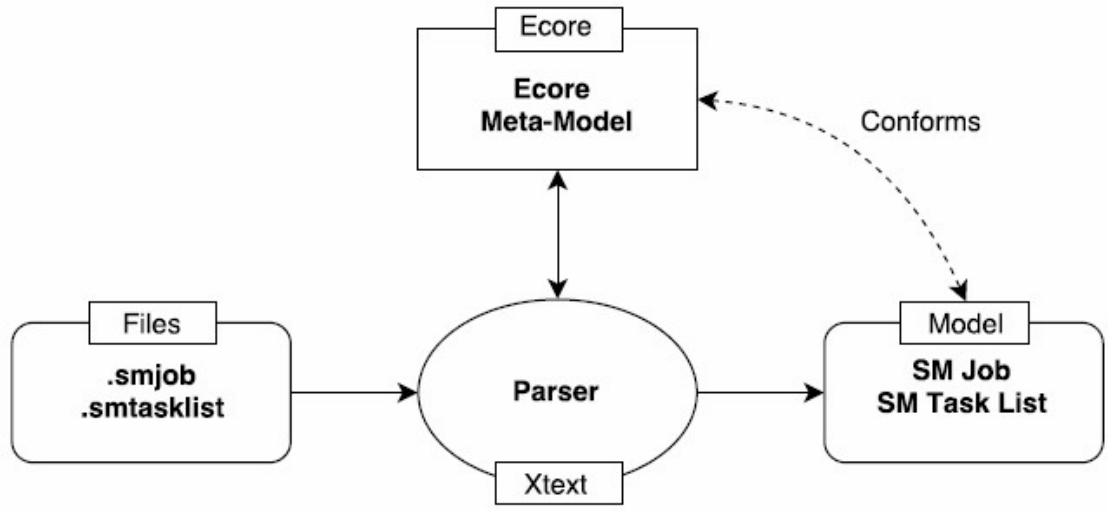


Рис. 2.4. Трансформація моделі - завдання SM на рівень процесу

Як показано на рисунку 2.4, вхідними даними аналізатора є файл завдання SM, а виходом аналізатора є модель предметної області для рівня процесу. Xtext підтримує специфікацію граматики та створює спеціальний аналізатор, який називається аналізатором SM. Крім того, створюється me ta-модель, яка відповідає парсеру SM. Створена мета-модель пов’язує файли SM Job безпосередньо з моделлю домену. Перетворені моделі домену є частиною рівня процесу. Описане перетворення є частиною підходу, який формалізує рівень процесу.

Інструмент апаратно-програмного інтерфейсу (HSI) ASML є інструментом активного моніторингу, який допомагає архітекторам системи переглядати стан апаратного забезпечення та його фізичних з’єднань. Оскільки обсяг цього проекту відповідає цьому інструменту, інструмент HSI

вважається потенційним джерелом інформації для побудови моделі домену. Інструмент HSI надає таку інформацію, як тип машин, доступні операційні системи, доступні плати, доступні ЦП, ієрархія пам’яті тощо. Трансформація HSI може бути використана для формалізації рівня обладнання та ОС.

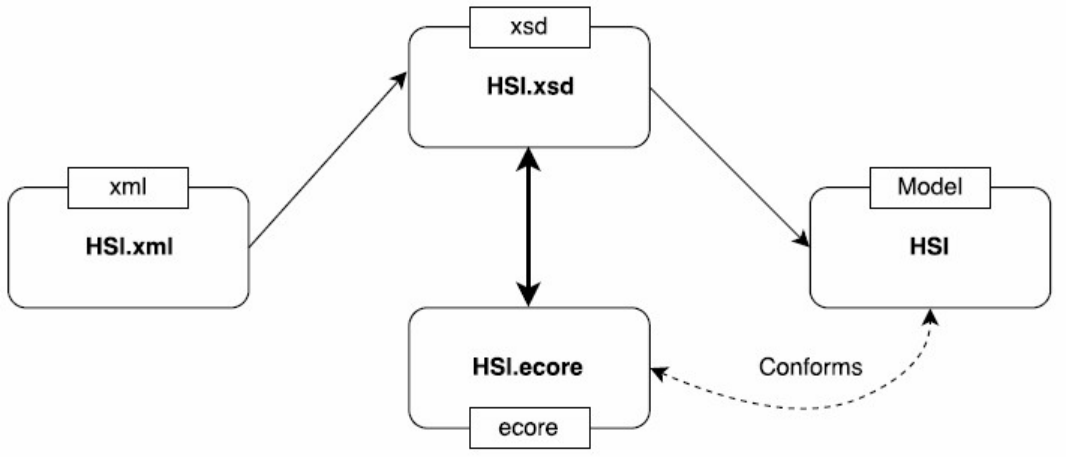


Рис. 2.5. Трансформація моделі - рівень HSI в апаратну ОС

Як показано на рисунку 2.5, HSI.xml перетворюється на модель домену HSI за допомогою визначення схеми XML (XSD). Файл HSI.xml діє як вхідний файл для підходу, а результат має форму визначених моделей домену. Спочатку XSD використовується для визначення інструменту HSI в термінах моделі. Однак XSD визначає модель у термінах підтримуваних XML моделей, а не в ecore. Таким чином, у ecore створюється мета-модель, яка відповідає HSI.xsd. Це може трансформувати та формалізувати інформацію HSI, щоб визначити концепції домену.

2.3. Реалізація рівнів моделі домену

Модель домену надає формалізовані концепції домену для машини, починаючи від прикладного рівня до рівня ОС і апаратного забезпечення. Стек забезпечує спеціальну модель домену для кожного рівня. На рисунку 2.6 показано загальний вигляд моделі домену. Найвищий рівень представляє

прикладний рівень. Середній рівень представляє рівень процесу. Нижній рівень представляє рівень обладнання та ОС разом як рівень обчислювальної платформи.

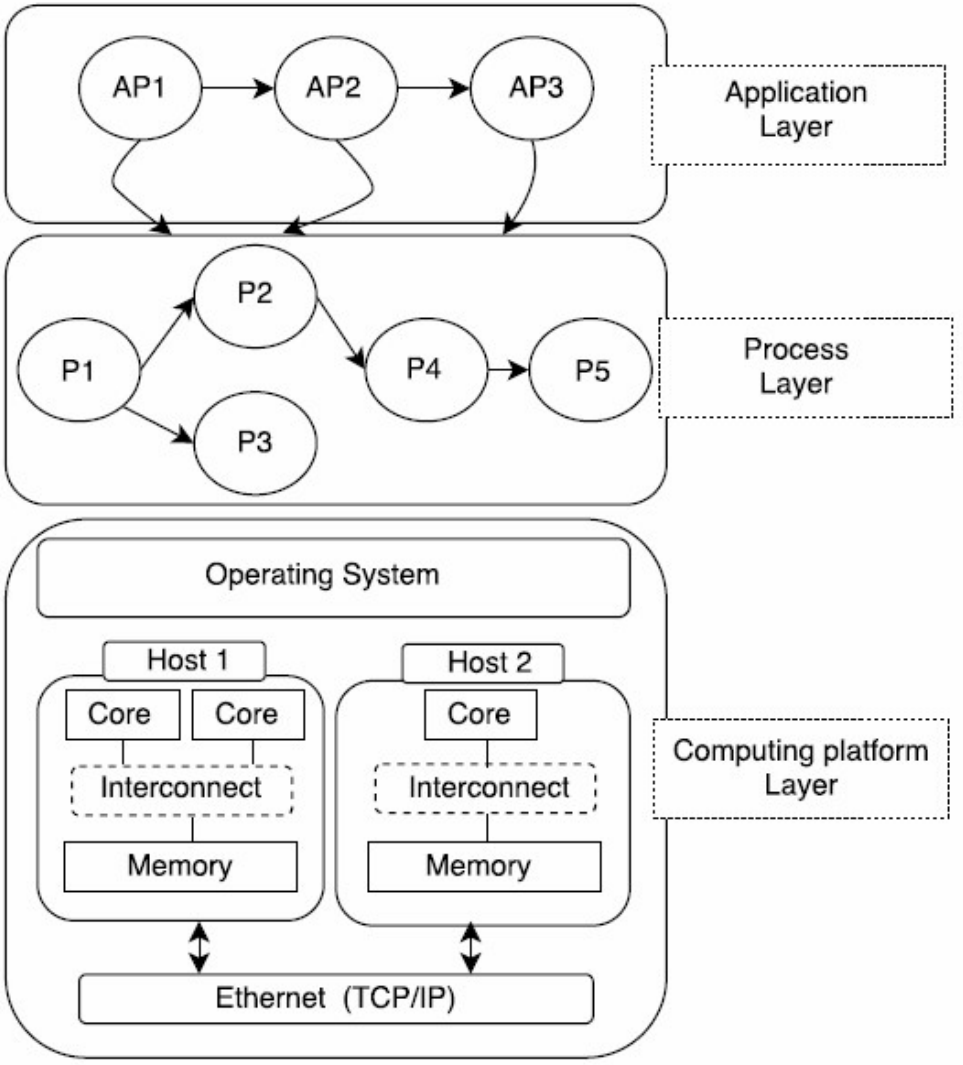


Рис. 2.6. Модель домену

Як показано на рисунку, програма, що складається з різних компонентів, відображається на рівні процесу. Кожна програма має набір процесів і взаємодіє з іншими програмами. Ці процеси далі розгортаються на виділеному хості відповідно до їх функціональності. Як згадувалося раніше, кожен хост має власну архітектуру, і кожна архітектура містить окрему обчислювальну платформу. Хости спілкуються через Ethernet за допомогою протоколу TCP/IP.

2.3.1.Метамодель рівня процесу

Функціональність складної системи, зафіксована в програмах, яка викликає набір процесів для завершення операції. У цьому розділі обговорюється модель рівня процесу з необхідною інформацією з прикладного рівня. Щоб зрозуміти загальну поведінку додатків, важливо абстрагувати процеси з міжпроцесним зв’язком.

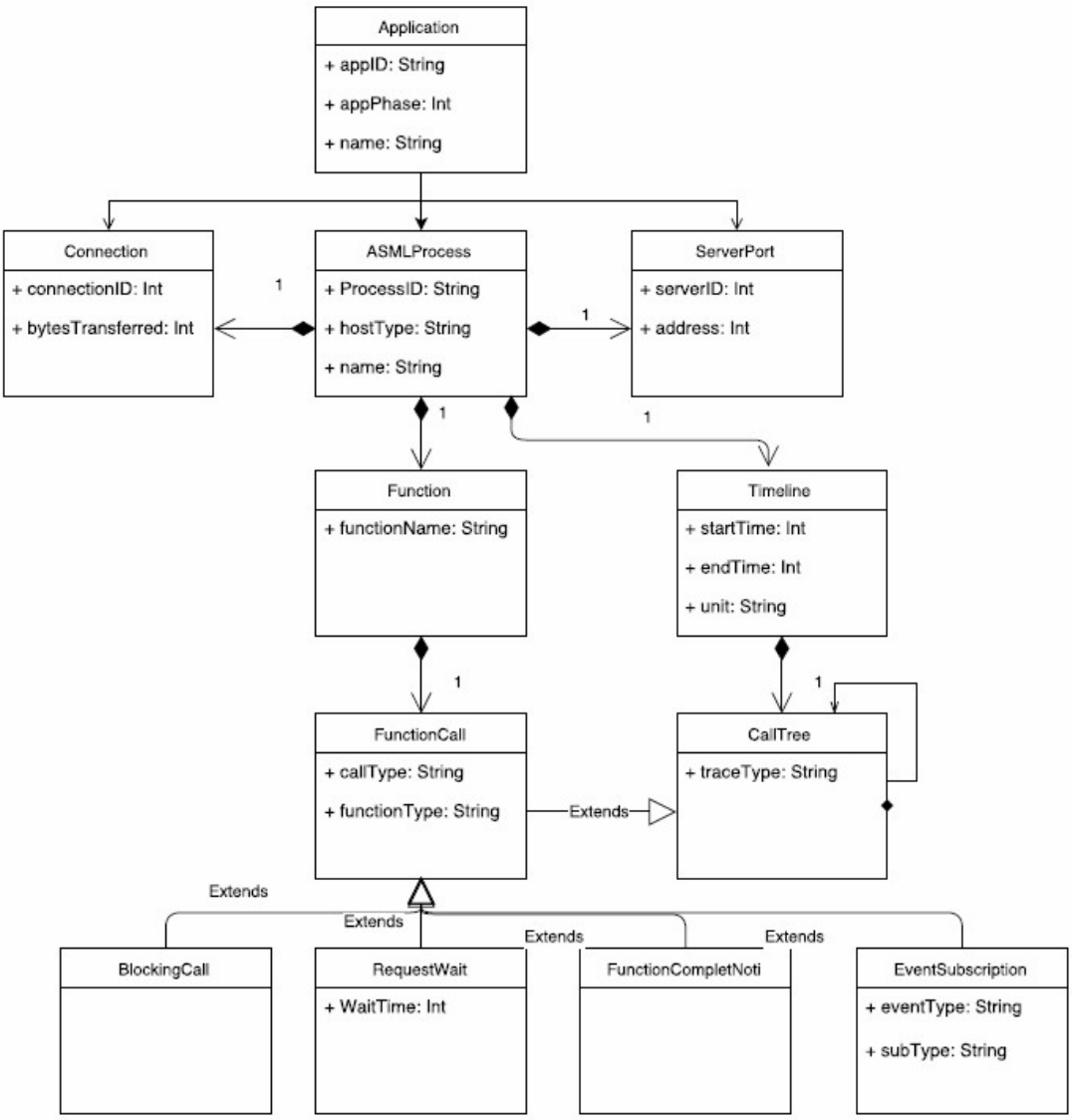


Рис. 2.7. Метамодель рівня процесу

Рівень процесу представляє функціональні можливості програм, які відображаються на процеси. Рівень процесу складається з підключень і портів сервера. Процес спілкування зв'язується з іншими процесами через

порти сервера. Ці процеси викликають інші процеси за допомогою спеціальних викликів функцій. Виконання функцій записується в інструменті аналізу виконання в реальному часі. Дерево викликів розрізняє синхронні та асинхронні виклики функцій.

Взаємозв’язок між процесами відбувається за моделлю клієнт-сервер, як показано на рисунку 2.8. Процес, який запитує будь-яку послугу від іншого процесу, вважається клієнтом, тоді як процес, який відповідає на цей запит, називається сервером. Синхронний виклик передбачає, що процес блокуватиме повернення виклику з сервера перед продовженням, тоді як асинхронні виклики не блокують повернення виклику з сервера. Зв'язок між клієнтом і сервером здійснюється за допомогою протоколу TCP/IP.

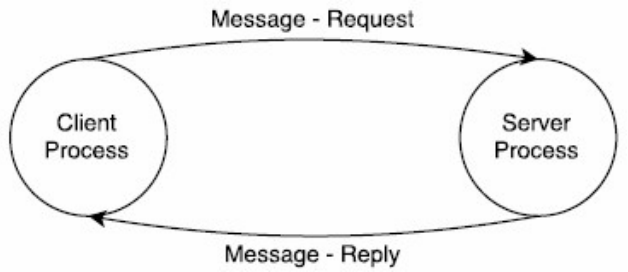


Рис. 2.8. Модель сервер-клієнт

Програми взаємозалежні, щоб забезпечити певні функції. Процеси, у свою чергу, взаємозалежні з іншими процесами та взаємодіють з ними. Проте кожен процес використовує різні механізми виклику функцій для зв’язку з іншими процесами. Ці процеси використовують виклик функції для запиту служби від іншого віддаленого процесу. Крім того, асинхронна функціональність містить два додаткові механізми, а саме «блокування», «функція запиту та очікування», «повідомлення про завершення функції» та «подія/підписка». Підписка на подію підписує подію в іншому процесі для моніторингу даних. У наступному розділі описано вищезгадані механізми, і всі процеси вважаються серверними та клієнтськими. Виклик функції від

клієнта до сервера називається запитом, а виклик функції від сервера до клієнта називається відповіддю.

Синхронні виклики функцій – це виклики функцій, у яких після того, як клієнт надсилає запит серверу, клієнт блокується, доки не буде надано відповідь із сервера. Таким чином, клієнт чекатиме відповіді від сервера, не виконуючи інших завдань. Стандартним інтерфейсом в архітектурі ASML є синхронний виклик функції. Виклик синхронної функції може уникнути інших можливих накладних витрат у спілкуванні. З іншого боку, виклик функції може перешкоджати прогресу інших функцій, що зрештою робить виконання менш ефективним.

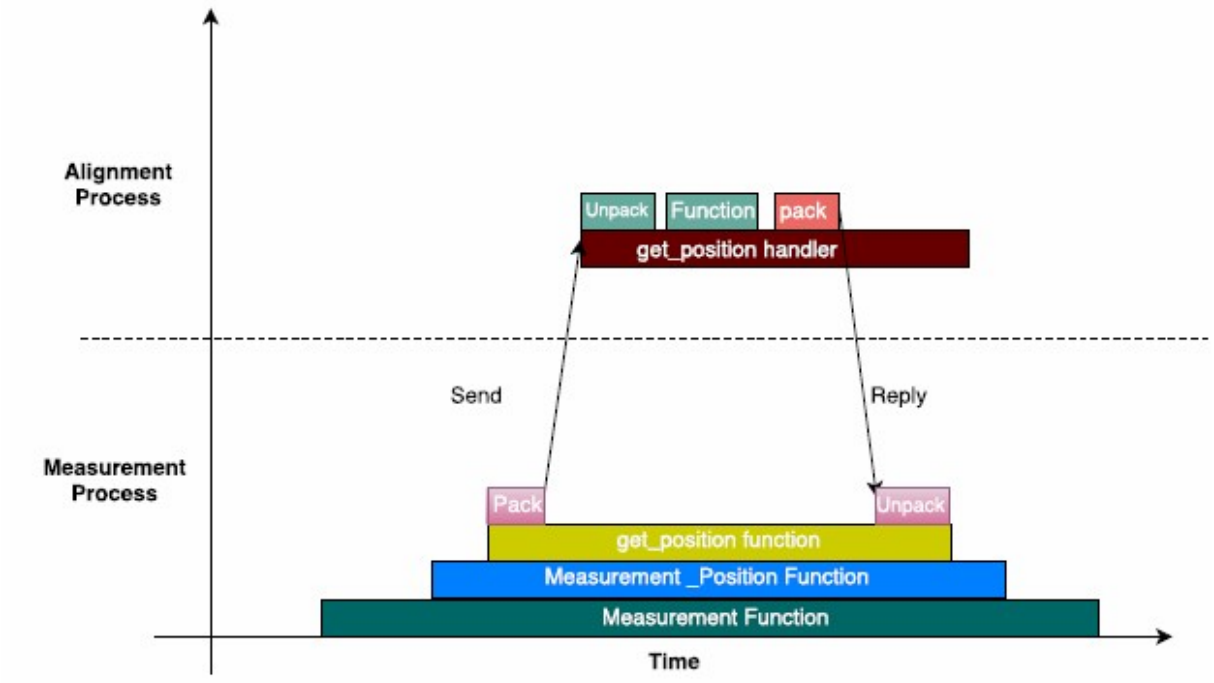


Рис. 2.9. Виклик синхронної функції

На рисунку 2.9 показано трасування функції, де відбувається синхронний виклик функції між процесом вимірювання та процесом вирівнювання. У цьому сценарії процес вимірювання — це клієнт, а процес узгодження — сервер. Клієнт запитує послугу з getposition , і сервер обробляє цей запит. Коли сервер завершує виконання, сервер відповідає відповіддю на getposition. До того часу процесу вимірювання заборонено виконувати будь-

яку іншу функцію, тобто функція getposition блокується, доки вона не отримає відповідь від обробника getposition.

Асинхронний зв’язок відноситься до виклику функції, коли після того, як клієнт надсилає запит серверу, клієнт може продовжити виконання власних завдань, очікуючи відповіді від сервера. Клієнт може чекати або повторно надсилати запит.

У цьому випадку клієнт надсилає запит на сервер і чекає відповіді певний час. Тайм-аут для функції очікування додається до програми перед виконанням у реальному часі. Після надсилання запиту серверу клієнт чекає відповіді від сервера до закінчення часу очікування. Тим часом клієнт може виконувати якусь іншу функцію. Після закінчення зазначеного часу клієнт починає чекати відповіді. Зрештою сервер відповідає або надсилає повідомленням про помилку.

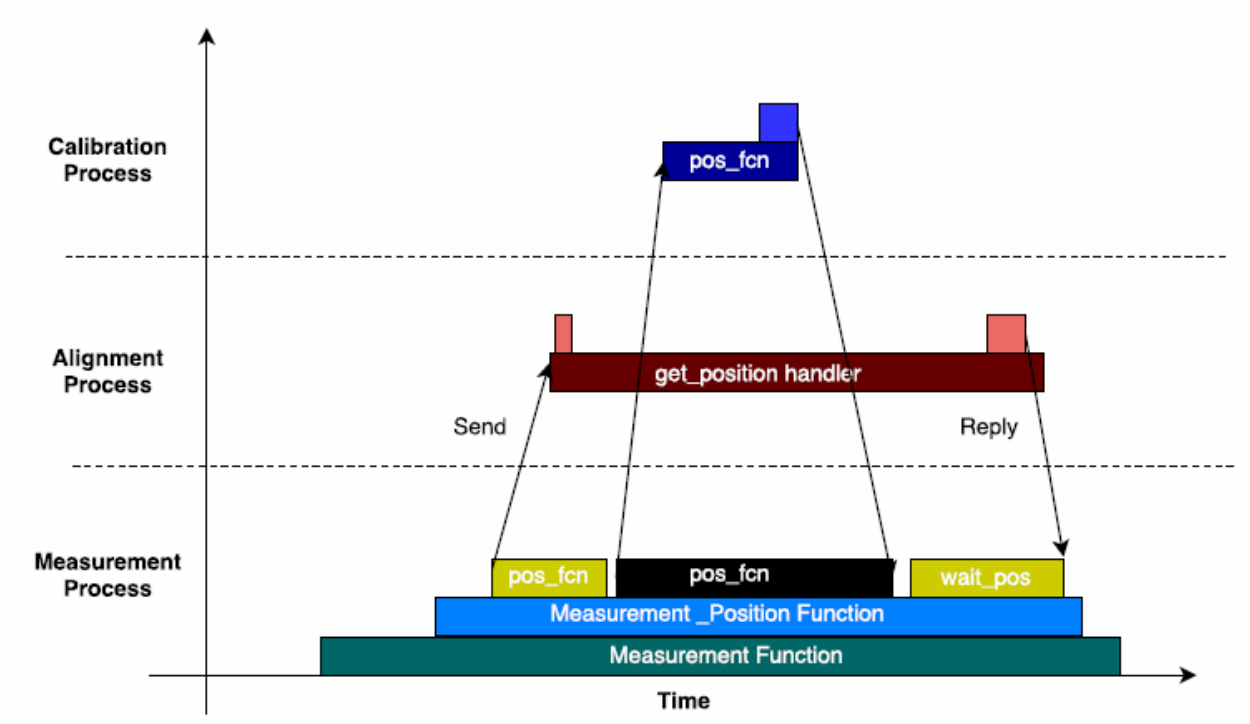


Рис. 2.10. Процес надсилання запиту і очікування відповіді

На рисунку 2.10 показано трасування функції, де між процесом вимірювання, процесом вирівнювання та процесом калібрування

відбувається асинхронний виклик функції, запит і очікування. У цьому сценарії процес вимірювання — це клієнт, а процес вирівнювання — сервер. Клієнт запитує послугу від програми вирівнювання, яка діє як сервер, і вона обробляє цей запит. Однак процес узгодження відповідає відповіддю після отримання запиту на завершення функції та починає його виконання. Згідно з параметром тайм-ауту, процес вимірювання почне очікувати відповіді через вказаний час. Якщо відповідь не була успішною після тайм-ауту, процес вимірювання вважатиме це помилкою. Тим часом, процес вимірювання може виконувати інші функції паралельно. На наведеному малюнку процес вимірювання вимагає виклику калібрів у процесі калібрування. Процес калібрування обробляє запит і відповідає на процес вимірювання. Через вказаний час waitpos починає очікувати відповіді від процесу вирівнювання.

Повідомлення про завершення функції (fcn - Function completion notication) — це механізм, який сигналізує про завершення запиту за допомогою виклику функції зворотного виклику. Щоразу, коли клієнт викликає функцію на сервері, клієнт може виконувати якийсь інший процес, доки не отримає сповіщення про завершення функції від сервера. У контексті ASML функція має префікс fcn і змінюється перед виконанням під час виконання. Однак сповіщення про завершення функції створює ризик блокувань, спричинених клієнтом і сервером, які пишуть одне одному велике повідомлення. Крім того, обробка у функції зворотного виклику залежатиме від стану клієнта та впливатиме на нього. Реалізація буде складнішою, ніж синхронний зв’язок.

На рисунку 2.11 показано трасування функції з асинхронним викликом функції, у якому сповіщення про завершення функції відбувається між процесом вимірювання та процесами вирівнювання та калібрування. У цьому сценарії процес вимірювання — це клієнт, а процес узгодження — сервер. Клієнт запитує послугу процесу узгодження на сервері, і він обробляє цей запит. Однак процес узгодження відповідає відповіддю після отримання повного запиту та починає його виконання.

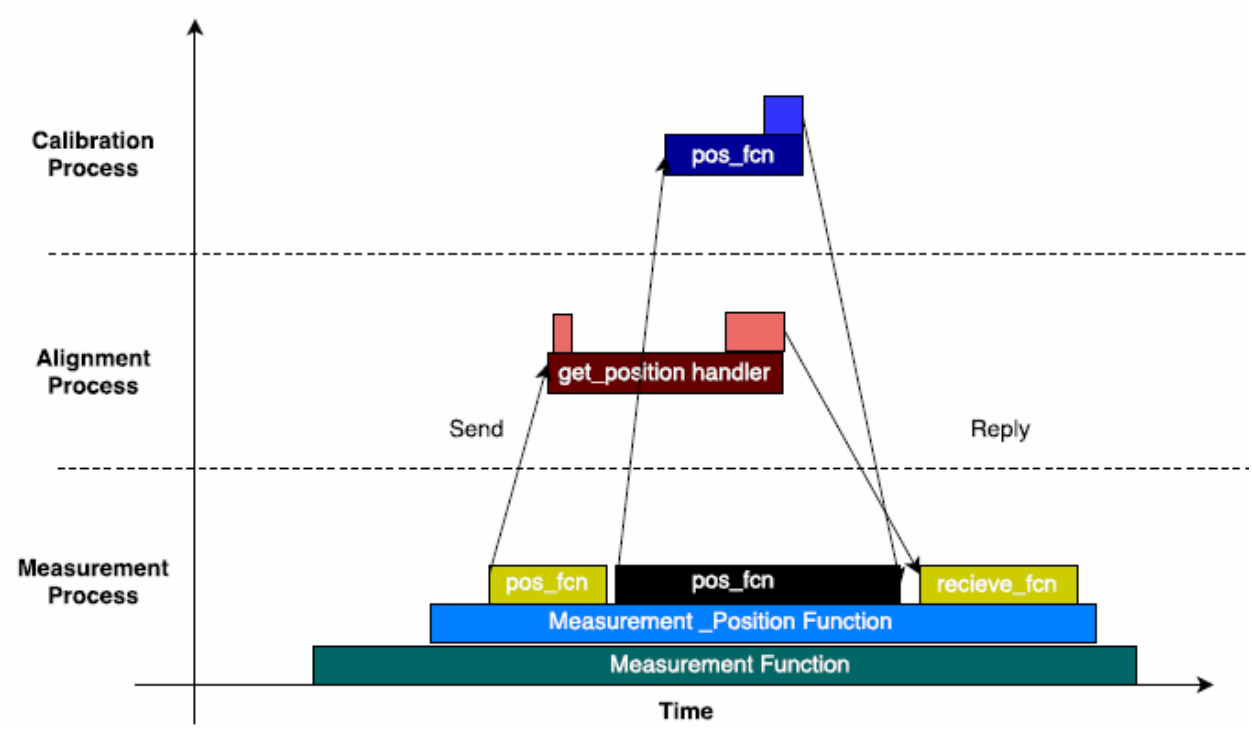


Рис. 2.11. Повідомлення про завершення функції

У fcn getpos не чекатиме відповіді, а натомість починає виконання після отримання сповіщення fcn від процесу вирівнювання. Тим часом, процес вимірювання може виконувати інші функції паралельно. На наведеному рисунку процес вимірювання вимагає процесу калібрування. Процес калібрування обробляє запит і відповідає на процес вимірювання. Коли обробник getpos з процесу вирівнювання відповідає сповіщенням, процес вимірювання відновить функцію getpos і завершить її виконання.

Окрім синхронного та асинхронного зв'язку, клієнту дозволено контролювати дані сервера за підпискою. Подія є одним із таких механізмів виклику функції. Крім того, процесам дозволено викликати подію в іншому процесі за допомогою викликів подій. Це запитає службу на сервері для виконання певної функції. Підписка надає доступ до моніторингу даних сервера шляхом підписки на події сервера. Щоразу, коли сервер викликає подію, функція клієнта буде повідомлена. Запити на підписку додаються до програмної пам'яті з деталями клієнта та сервера як параметр. Коли процес припиняє роботу, підписки або скасовуються, або втрачаються.

На рисунку 2.12 показано трасування функції, де відбувається підписка та скасування підписки. У цьому сценарії процес вимірювання — це клієнт, а процес узгодження — сервер. Процес вимірювання підписує функцію з процесу вирівнювання. Процес вирівнювання продовжує відстежувати subspos. Коли процес вимірювання завершує моніторинг процесу, він скасовує підписку, надсилаючи запит на скасування підписки.

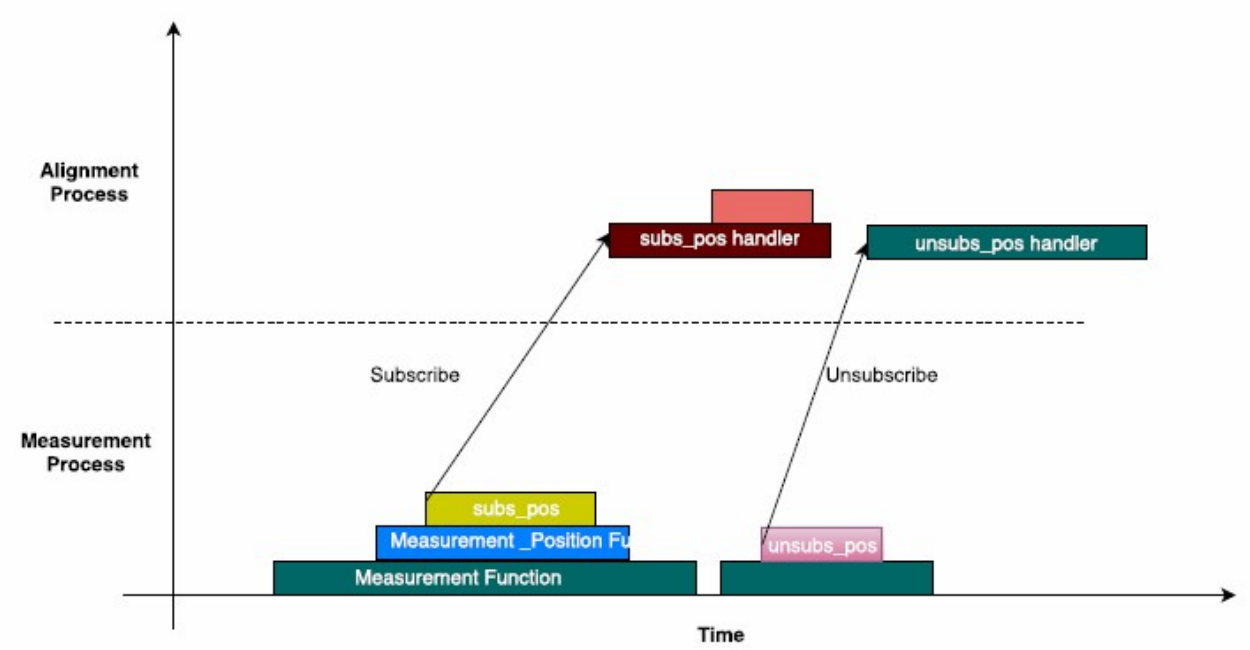


Рис. 2.12. Процеси підписки та її скасування

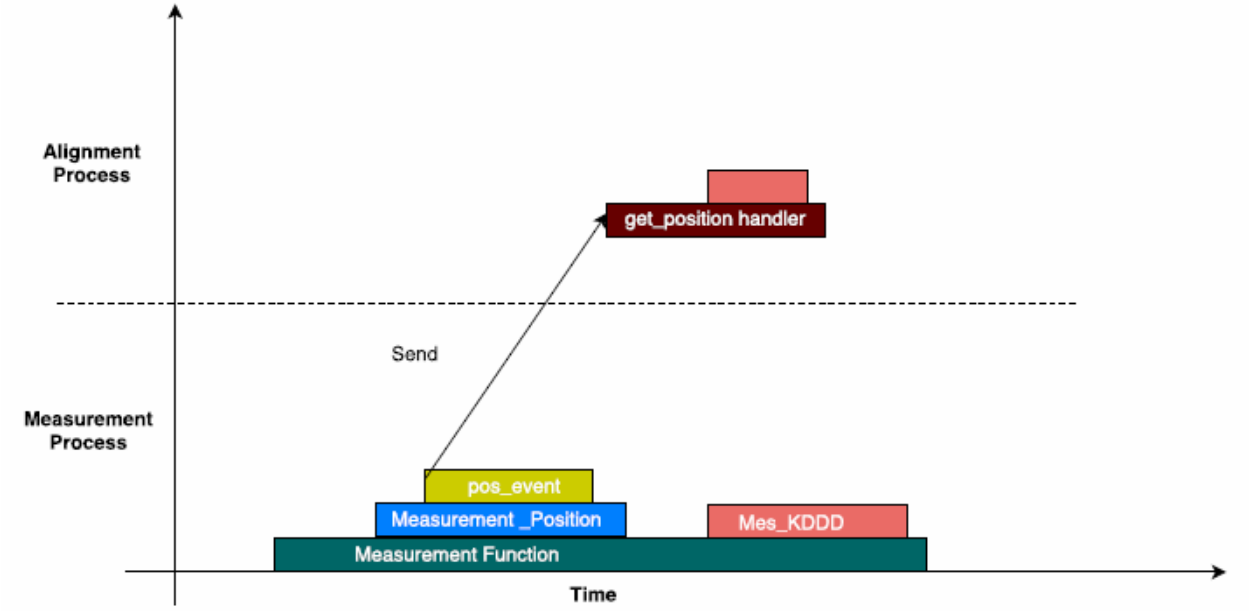


Рис. 2.13. Виклик події

На рисунку 2.13 показано трасування функції, де відбувається виклик події між процесом вимірювання та процесом вирівнювання. У цьому сценарії процес вимірювання є клієнтом, а процес вирівнювання — сервером. Процес вимірювання викликає posevent для процесу вирівнювання, а процес вирівнювання виконує подію getposition для обробки запиту. Подія завершується після завершення обробника події getposition.

2.4. Реалізація моделі рівня обчислювальної платформи

Рівень операційної системи містить формалізацію сутностей в ОС, таких як ядро, файлова система та системне сховище. Абстракція ОС додатково визначає, як плануються та виконуються процеси на доступній платформі. Апаратний рівень - це формалізація фізичних об'єктів, таких як блоки обробки, блоки зберігання та фізичні з'єднання. Таким чином, рівень операційної системи та рівень апаратного забезпечення інтегровані, щоб відображати поведінку обчислювальної платформи.

Операційна система (ОС) відповідає за управління ресурсами обчислювальної платформи шляхом планування завдань на доступному наборі ресурсів. Абстракція операційної системи може розглядати поведінку на рівні ОС. Рівень ОС складається з типів операційної системи, типу ядра, процесів і потоків ОС, а також доступних ресурсів. Ядро містить інформацію про планувальник, стек і файлову систему. Процес пов’язаний з потоками та міжпроцесним зв’язком через сокети.

На рисунку 2.14 показана метамодель рівня операційної системи. Як згадувалося раніше, модель операційної системи складається з процесів і ядра. Модель ОС надає інформацію про тип операційної системи, версію, яку вона підтримує, і інформацію про планувальник. Процес надає інформацію про тип процесу, загальну кількість залучених процесів і доступні ЦП. Далі процес ділиться на кілька потоків для паралельного виконання. Ці процеси

взаємодіють з іншими процесами через сокети. Крім того, ядро має власний ресурс, і кожному ядру надається файлова система, стек і планувальник.

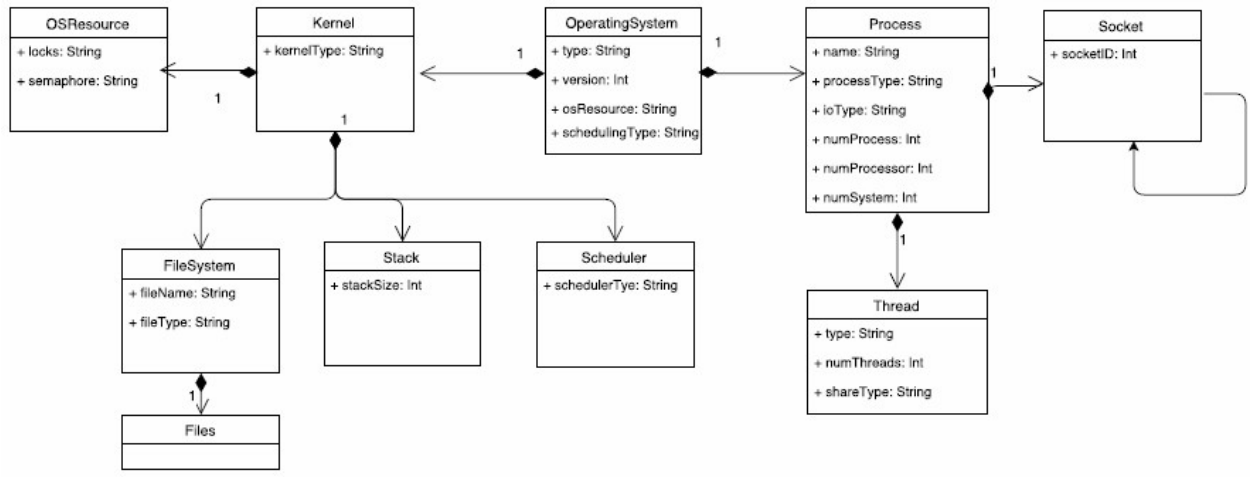


Рис. 2.14. Рівень операційної системи

У цьому завданні ми більше зосереджуємося на планувальнику та обробці процесів, щоб ефективно визначати та аналізувати поведінку системи за часом.

У ASML Linux і Solaris є двома широко використовуваними операційними системами. Обчислювальна платформа на машині TWINSCAN містить кілька хостів. Кожен хост постачається зі спеціальною операційною системою. Ці хости отримують команди від центрального сервера, який називається головним. В останній версії Linux процеси плануються за допомогою Completely Fair Scheduler (CFS). CFS наближає ідеальну багатозадачність до доступних завдань; ці кілька завдань отримують необхідну частку ресурсів відповідно до їх ваги. Частка кожного процесу визначається його привабливістю. Значення точності в CFS призначається від -20 до 19. Це значення акуратності використовується для обчислення ваги кожного процесу, який визначає його частку серед інших на доступній платформі.

Апаратний рівень представляє апаратне забезпечення та фізично підключені ресурси, які присутні в системах літографії. Апаратний рівень

має справу з фізичними об’єктами, які відповідають за виконання і показані на рисунку 2.15 як мета-модель апаратного рівня. Кожна стійка містить модуль стійки, а кожен модуль стійки містить порт. Ці стелажні модулі взаємодіють з іншими стійками через з’єднання та порти.

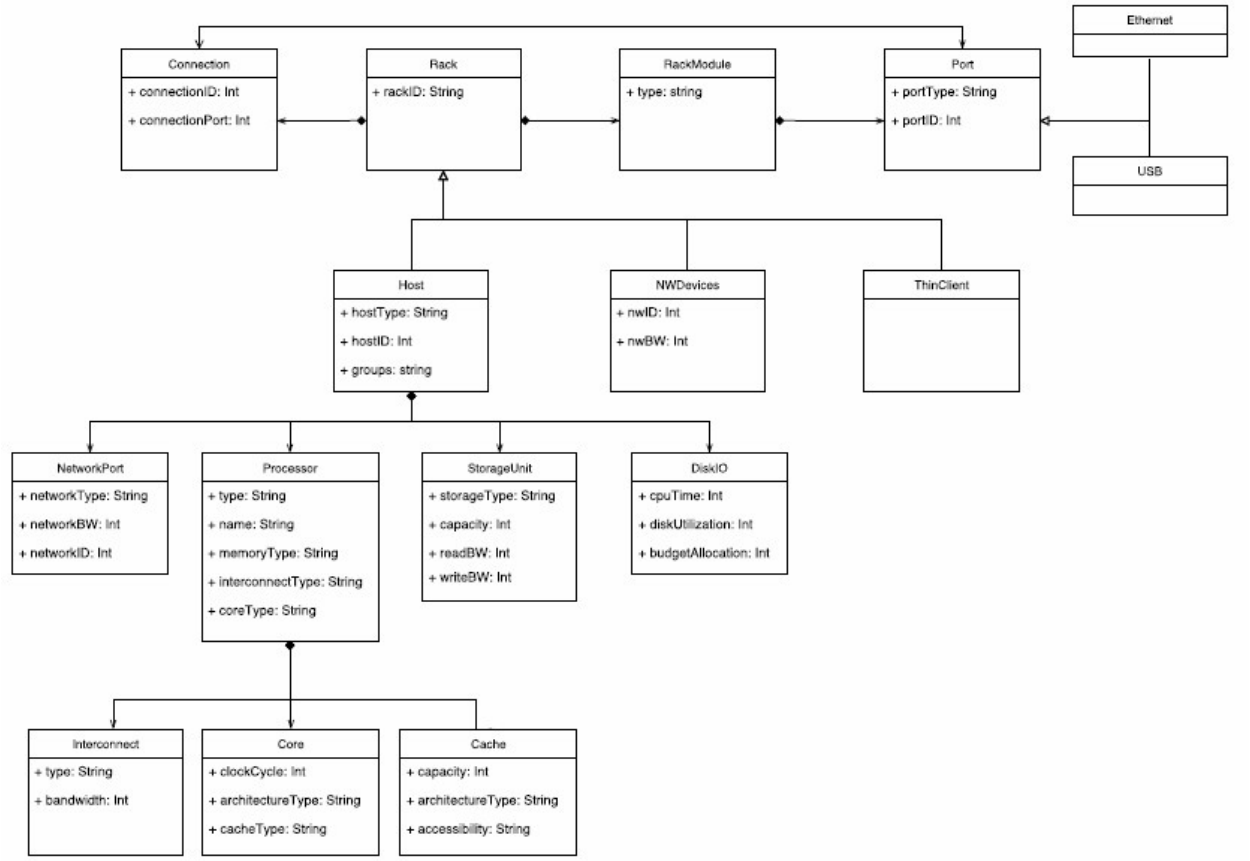


Рис. 2.15. Реалізація апаратного рівня

Модуль в стійці реалізується у вигляді хостів, мережевих плат або тонкого клієнта. Оскільки ми зосереджені на платформі виконання, хости та мережеві плати є доречними, а тонкі клієнти не є актуальними. Хост містить процесор, мережеві порти та блоки зберігання. Процесор має виділене ядро, тип з’єднання та кеш-пам’ять. Ядро — це фізична сутність, яка відповідає за виконання процесів. Інтерконнекти дозволяють зв'язуватися між ядрами з певною топологією. Блок зберігання в хості відповідає за зберігання відповідних даних для його функціонування. Це також фізична сутність, яка зберігає або отримує дані під час виконання процесу.

Додатково стійки реалізовані у вигляді мережевих плат. Ці плати є спеціальними блоками для підтримки модулів стійки для зв’язку з іншими активними модулями стійки. Як правило, ці зв’язки відбуваються через порт Ethernet за допомогою протоколу TCP/IP. Ці сутності згруповуються разом, щоб діяти як ресурс платформи.

Висновки до розділу

У цьому розділі досліджується трансформація моделей предметної області з вилучених артефактів. Початковий підхід було здійснено шляхом збору необхідної інформації. Зібрана інформація далі перетворюється на модель домену для візуалізації цільового перегляду. Модель домену складається з прикладного рівня, рівня процесу та рівня обчислювальної платформи. Прикладний рівень і рівень процесу разом представляють процеси, специфічні для ASML, і їх взаємозв'язок. Крім того, обговорюються механізми взаємозв’язку, щоб зрозуміти поведінку процесів. Рівень обчислювальної платформи представляє фізичні та логічні сутності, які використовуються для обробки та виконання програми та її процесів.

Також, в розділі описується загальний підхід, надаючи необхідні елементи для аналітичної моделі. Основні елементи моделей домену розглядаються та застосовуються до аналітичної моделі, щоб зрозуміти поведінку синхронізації. Аналіз додатково допомагає оптимізувати продуктивність синхронізації шляхом визначення альтернативних конфігурацій.

РОЗДІЛ 3. ЗАСТОСУВАННЯ ПАРАМЕТРИЧНОЇ АНАЛІТИЧНОЇ МОДЕЛІ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСІВ ВИКОНАННЯ ПРОГРАМНИХ ДОДАТКІВ

3.1. Підхід до розробки параметричної моделі

Параметрична аналітична модель використовується для прогнозування продуктивності синхронізації та оцінки проектних рішень. Реконструйовані моделі з активної машини аналізуються, щоб зрозуміти різні фактори продуктивності, такі як кількість обчислювальних блоків, тип процесора, мережа зв'язку між обчислювальними блоками та кількість залучених процесів. Ці моделі перетворюються на параметричну аналітичну модель для оцінки продуктивності обчислювальної платформи. За результатами аналізу продуктивність системи може бути покращена за допомогою різних комбінацій конфігурацій параметрів.

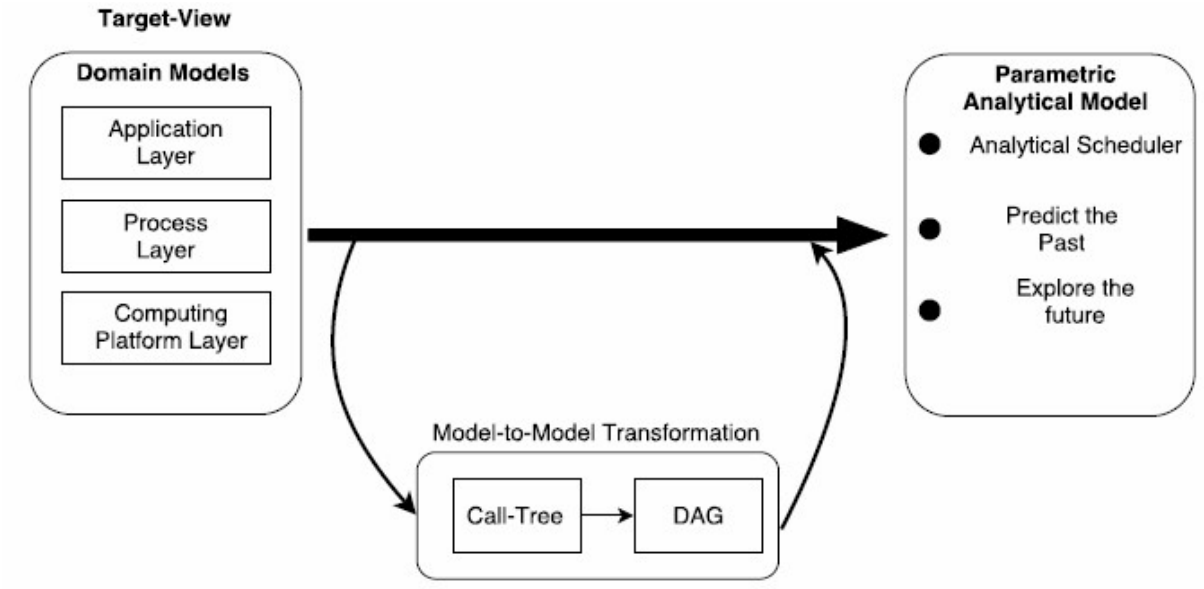


Рис. 3.1. Представлення підходу реконструкції моделей домену

В цьому розділі зосереджено увагу на початковому кроці до визначення характеристик синхронізації системи. На початковому етапі буде використано важливу інформацію з різних рівнів моделі цільового перегляду.

Важливою інформацією, яка впливає на ефективність синхронізації, є кількість задіяних процесів, використовуваний планувальник і доступні обчислювальні блоки. Таким чином, інформація про процес виводиться з рівня процесу, а інформація про планувальник і обчислювальний блок – з рівня обчислювальної платформи. Запропонована параметрична аналітична модель описується в цьому розділі, а суттєва інформація, необхідна для оцінки продуктивності представлена в розділі 3.2.

Пропонований підхід використовує інформацію з активної машини. Блоки параметричної аналітичної моделі показані на рисунку 3.2.

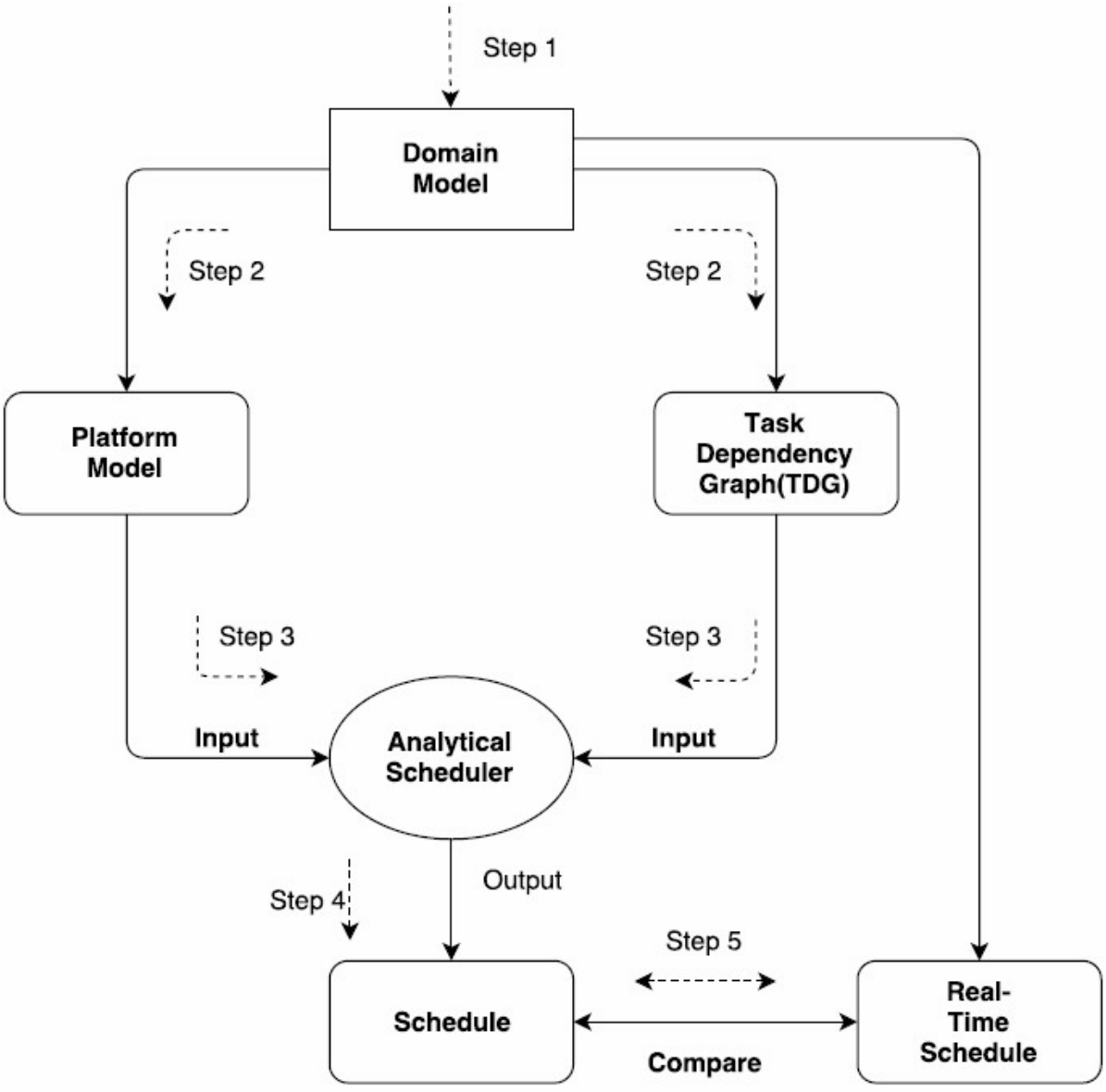


Рис. 3.2. Представлення підходу розробки параметричної аналітичної моделі

Параметрична аналітична модель відповідає за розклад, який спостерігається з працюючої машини. Пізніше цю модель можна використовувати для налаштування обчислювальних параметрів, таких як кількість ядер у процесорі, пріоритети завдань і тактова частота процесора для покращених конфігурацій. Покращена конфігурація може дати кращий результат, ніж поточна конфігурація.

Модель домену містить інформацію про виконання для аналізу продуктивності синхронізації. Основна інформація витягується з кожного рівня для підтримки параметричної аналітичної моделі. Крім того, модель домену надає інформацію про завдання, які виконуються на процесах і далі розгортаються на однорідному мультипроцесорі. Аналітична модель обмежена мультипроцесорами з ідентичними ядрами та без гіперпотоків, а затримка під час отримання даних із пам’яті вважається нульовою. Модель імітує цілком чесний планувальник із діапазоном пріоритетів від -20 до 19 і кожному завданню призначається пріоритет. Кожен блок на рисунку 3.2 представляє крок для побудови параметричної аналітичної моделі. Кроки такі:

1. Першим кроком є реконструкція моделі предметної області з працюючої машини тобто моделі домену реконструюються з різних артефактів в працюючій машині. Ці моделі містять інформацію про використовувані програми, операційну систему та обчислювальну платформу.

2. По-друге, дуже важливо витягти відповідну інформацію, таку як інформація про платформу та залежності завдань, із моделі домену. Інформація витягується з рівня програми, процесу та обчислювальної платформи разом. Точніше, він надає інформацію про виконання функцій та їхніх залежностей як міжпроцесний зв’язок. Залежності виникають, коли функція вимагає виконання іншої функції шляхом передачі необхідної інформації. Функція також включає серіалізацію, яка збирає та організовує інформацію, яку потрібно надіслати, і десеріалізацію, яка розділяє

організовану інформацію та процес. Тому було враховано розділення часу, витраченого на функції та спілкування. Поділ розглядається як завдання як функціональні завдання та завдання спілкування відповідно.

Функціональне завдання — це час процесора, який витрачається функцією на виконання її з виділеним ресурсом. Завдання зв'язку — це час, який необхідний зв'язку для надсилання або отримання даних із виділеним ресурсом. Розділення в дереві викликів надає інформацію про час функцій і зв'язок між процесами. Крім того, інформація про центральний процесор функціонального завдання та завдання зв'язку витягується з дерева викликів. Таким чином, час виконання та час процесора завдань буде проаналізовано, щоб зрозуміти загальну поведінку обчислювальної платформи. Цей крок також дозволяє побудувати спрямований ациклічний граф (DAG) з отриманих деталей завдання та їх залежностей. DAG додатково доповнюється часом виконання завдань у вигляді графіка залежностей завдань (TDG). Функціональні завдання та комунікаційні завдання є вузлом, а залежності в міжпроцесному зв'язку є ребрами.

Модель платформи надає інформацію про ресурси платформи, такі як кількість використовуваних однорідних мультипроцесорів і тактова частота однорідного мультипроцесора. Параметри, які можна налаштувати в планувальнику, це кількість ядер, тактова частота мультипроцесора та пріоритет завдань, розгорнутих на мультипроцесорі. Ці параметри можна адаптувати в планувальнику та отримати результуючий розклад.

3. Після визначення інформації про платформу та інформації про залежність завдання інформація планується за допомогою аналітичного планувальника. Планувальник наближає поведінку цілком справедливого планувальника. Початкові ввімкнені завдання класифікуються як активні завдання. Вони розглядають список активних завдань у часі та планують їх з рівною часткою. Частка для кожного завдання визначатиметься цінністю зручності кожного процесу. Ефективний час ЦП буде наближено для його планування.

4. Результатом роботи алгоритму планувальника є розклад. У розкладі завдання відображаються за часом із загальним завантаженням ЦП. Оскільки результати операційних систем Linux у реальному часі базуються на повністю чесному планувальнику (CFS), розклад також буде імітувати повністю чесний планувальник на базі Linux.

5. Останнім кроком буде порівняння отриманого графіка з результатами в реальному часі. Це допоможе нам визначити потужність прогнозування шляхом порівняння з плануванням завдань у реальному часі на активній машині. Буде порівнюватися ефективний час виконання визначеного розкладу та розкладу в реальному часі. Після визначення передбачуваної потужності алгоритму планування, алгоритм можна використовувати для дослідження альтернативного відображення та конфігурацій платформи. Дослідження можна виконати шляхом налаштування параметрів обчислювальної платформи.

3.2. Запропонована модель та алгоритм оптимізації

Запропонована модель використовує підхід, керований моделлю, витягуючи необхідну інформацію з моделей і допомагаючи планувальнику планувати завдання. Кожне завдання в DAG зіставляється з часом процесора, який виконується відповідно до розподілу ресурсів. Перед визначенням моделі та виведенням можливого сценарію, для побудови параметричної аналітичної моделі робляться наступні припущення.

Функціональне завдання – це завдання, яке використовує ресурси процесора. Завдання містить час обчислень, виміряний інструментом трасування. Це включає серіалізацію та десеріалізацію даних, які мають бути передані іншій функції під час віддаленого виклику процедури. Функціональне завдання визначається T_f , де $T_f \in T$.

Комунікаційне завдання – це завдання, яке використовує комунікаційні плати, тобто мережеві плати для зв’язку між процесами. Завдання зв’язку

містить час спілкування, який відстежується під час спілкування між функціями. Комунікаційне завдання визначається T_c , де $T_c \in T$.

Згадані завдання можуть бути змодельовані спрямованим ациклічним графом G = (T, →) , де T = {T_0 ,.., T_n} представляє набір завдань. Ребра, які представляють залежності завдань $\rightarrow \subseteq \tau \times \tau$ так, що $\forall_{T\in}$: T → ∗ T. Кожне завдання τ і має вартість обчислення cpu(τ_i).

Кожне завдання доповнюється часом виконання . Час виконання — це час, потрібний процесору для виконання завдання. Для цього ми вводимо функцію CPU(T):T → R + і представляємо CPU(T) для позначення виконання завдання T, де кожне завдання $T \in T$.

Останньою використовуваною операційною системою є Linux, отже, буде обговорено цілком чесний планувальник. Планувальник CFS використовує значення якості процесів (або завдань), щоб визначити вагу завдання. Значення якості представлено як N , де N коливається від -20 до 19. Спочатку визначається значення ваги процесів і обчислюється проти загальної ваги для визначення індивідуального використання. Індивідуальна вага позначається як W(T), де $W \in R +$ і завдання $T \in T$. З виділеною вагою W(T) ефективний час виконання визначатиметься планувальником. Ефективний час виконання — це час, потрібний завданню для завершення з виділеним ресурсом у планувальнику, і він позначається E(T), де кожне завдання $T \in T$.

$$Wi = 1024/((1.25)^{Ni})$$

$$Wt = \sum_{o}^{i} Wi$$

$$\mathcal{W}(T) = Wi/Wt$$

З виділеною вагою W ефективний час виконання визначатиметься планувальником. Ефективний час виконання — це час, потрібний завданню

для завершення з виділеним ресурсом у планувальнику, і він позначається E(T), де кожне завдання T ∈T.

$$E(T) = CPU(T) \times W(T)$$

Запропонований алгоритм визначає розклад завдань з використанням інформації про програму та ресурс платформи. Наступний алгоритм намагається наблизити планувальник Linux, який називається повністю справедливим планувальником (CFS). Повністю справедливий планувальник відомий рівною справедливістю до всіх завдань, які він обробляє. CFS намагається забезпечити ідеальну частку для всіх запущених завдань відповідно до значення акуратності.

Набори завдань: нехай τs — це набір завдань із інформацією про залежність і пріоритет (якість), а τe — початкові дозволені завдання, тобто завдання, які заплановано для виконання на процесорі, а τc — завершений набір завдань.

Алгоритм описує аналітичний планувальник, який використовується для отримання розкладу. Алгоритм використовує запропоновану модель для отримання набору завдань τ s з інформацією про залежність. Набір завдань зіставляється, щоб визначити активний набір завдань.

Як уже згадувалося, завдання містить інформацію про час виконання та значення точності.

У рядках 3 і 4 time.start і time.end представляють час початку і завершення різних завдань. Рядок 7 обчислює розподіл ЦП для ресурсу в увімкненому наборі завдань, а рядок 8 надає час виконання увімкнених наборів завдань. Потім у рядку 9 визначаються завдання з мінімальним часом виконання. Мінімальне виконання називається часом закінчення початкового завдання. Рядки з 11 по 13 представляють оновлення розкладу під час процесу виконання.

Algorithm 1: Algorithm for Analytic Scheduler

```
1: schedule = ∅
2: τ_e ← Sources(τ_s)
3: time.start = 0
4: time.end = 0
5: while ( τ_e ≠ ∅ ) do
6:     for each τ ∈ τ_e do
7:         Compute CPU allocation C_u(τ_e) for enabled tasks using definition 1
8:         Compute execution time E(τ_e) for enabled tasks using definition 2
9:         Min[E(τ_e)] = M_t
10:        M_t → time.end
11:        time.start ← time.start + time.end
12:        update.schedule(end.time,τ_e)
                                          ▷ %When task completed%
13:        τ_{M_t} ← τ_e / τ_{M_t}
14:        τ_c ← τ_c ∪ τ_{M_t}
15:        τ_c ↦ Schedule
16:
                                  ▷ %When task added to Enabled Task Set%
17:        if Any τ ∈ τ_e added then
18:            τ_e ← τ_e ∪ τ
19:        end if
20:    end for
21: end while
```

Рис. 3.3. Запропонований алгоритм для аналітичного планувальника

Рядки з 14 по 19 представляють обробку завдань після їх завершення та ввімкнення. Щоразу, коли завдання буде виконано, воно буде видалено з увімкненого набору завдань і оновлюватиметься за розкладом. Це запобігає непотрібному виконанню виконаного завдання. Крім того, виконані завдання також оновлюються на виконаних наборах завдань. Щоразу, коли нове завдання ввімкнено, воно буде додано до списку активних завдань, а розподіл ЦП буде визначено знову. Це підтримує майже наближену CFS і забезпечує ідеальну багатозадачність для ввімкнених завдань.

Складність алгоритму — це велика проблема. Складність алгоритмів впливає на час виконання та їх ефективність. Порядок складності представляє кількість циклів, які використовуються в алгоритмі. Наприклад, якщо алгоритм містить один цикл із вкладеними ітераціями, то порядок

складності визначається як O(n). Якщо в алгоритм з n ітераціями вкладено ще один цикл, то обчислювальна складність розглядається як O(n + 1) = O(n).

В представленому алгоритмі порядок складності визначається до $O(n^3)$. Складність визначається наступними кроками. Рядок 7 обробляє активне завдання τe для обчислення як розподіл ЦП. Незалежно від кількості вкладених обчислень, обчислювальна складність рядка 7 становить O(τe). Подібним чином рядок 8 виконує обчислення для обчислення часу виконання з увімкненою задачею τ e . Це також забезпечує обчислювальну складність O(τe). Така сама процедура виконується і в рядку 9. Загальна обчислювальна складність обчислюється як потрійне O(τe). Це призводить до O(τe) 3 як загальної обчислювальної складності алгоритму. Однак результуюча складність не є оптимальною, але оптимізація алгоритму виходить за межі даної роботи.

Крім того, послідовність обчислень виконується за допомогою алгоритму розгортки. Алгоритм лінії розгортки використовує поверхню розгортки для вирішення проблеми з простором і часом. Реалізація додатково зменшує загальну складність, маючи справу з простором у часі. Пропонований алгоритм розподіляє ЦП за завданням і виконанням за часом. Таким чином, алгоритм лінії розгортки допоміг впоратися з нею одночасно.

3.3. Представлення трансформації модель-модель. Сценарії оптимізації виконання програмних додатків

"Model-to-Model Transformation" – це термін, який у контексті комп'ютерних наук і штучного інтелекту означає перетворення однієї моделі даних в іншу. Це процес, коли інформація, представлена в одній форматі моделі, перетворюється на еквівалентну інформацію, представлену в іншій форматі моделі.

Модель дерева викликів відіграє ключову роль у визначенні поведінки процесу. Модель дерева викликів показана на наступних рисунках. Вона

складається з функції та її викликів. Ці виклики функцій відображаються на шкалі часу. Ці функції мають різні типи, наприклад блокуючий виклик, запит і очікування, сповіщення про завершення функції (fcn) і підписку на події.

Функції розділені як функціональне завдання та комунікаційне завдання. Функції представлені в трасах реального часу з їх початковим і кінцевим часом. Цікаві мітки часу позначаються та нарізаються, щоб квантувати їх як завдання. Нарізка для функціонального завдання відбувається з моменту виконання певного завдання, включаючи накладні витрати на зв’язок. У наведених нижче сценаріях показано, як відбулося абстрагування часу від кожного завдання для всіх можливих механізмів. Це оформлено у вигляді діаграми Ганта з різними наборами завдань.

Модель DAG представляє функціональне завдання та комунікаційне завдання із залежностями. Вузли - це функція та комунікаційне завдання, яке має ефективний час виконання. Ребра - це залежності, отримані через міжпроцесний зв'язок. Наступні сценарії побудовано з використанням абстракції дерева викликів у попередньому розділі. Ребра посилаються на функціональне завдання та комунікаційне завдання. DAG діє як графік для аналітичного планувальника.

Нарешті, ми опишемо можливі сценарії для викликів різних функцій під час перетворення. Розділ часу для кожного сценарію відрізняється залежно від типу виклику віддаленої процедури. Крім того, описано перетворення трас реального часу в DAG для кожного сценарію.

Сценарій 1

Виклик синхронної функції показаний на рисунку 3.3 і представлений як блокуючий виклик. Як вже було описано, клієнт запитує послугу від сервера та чекає, поки не отримає відповіді. Протягом періоду очікування процесу не дозволяється виконувати будь-які інші функції. це пов’язано з тим, що задіяний процес є однопоточним, і він виконує одну функцію за раз. Час, витрачений функціями, представлений у вигляді часових позначок T

={T1 , T2 ,.., Tn}. Таким чином, функціональне завдання та комунікаційне завдання на малюнку нижче представлені як FT і CT окремо. Процесорний час функцій відстежується на фоні процесу, а значення якості вважається однаковим для наведених нижче сценаріїв. У наведеному нижче сценарії функціональне завдання (FT) і комунікаційне завдання (CT) виводяться з різниці часових позначок (T).

Наприклад, FT1 — це функція, яку виконує процес вимірювання для завершення getposition. FT1 включає пакування та серіалізацію запиту. Час, витрачений FT1, є різницею T2 і T1. Крім того, CT1 — це час зв'язку, який getposition витрачає на передачу getPositionHandler. Подібним чином, час, витрачений на FT2, FT3 і CT2 як різницю T4 & T3 , T6 & T5 і T5 & T4 відповідно.

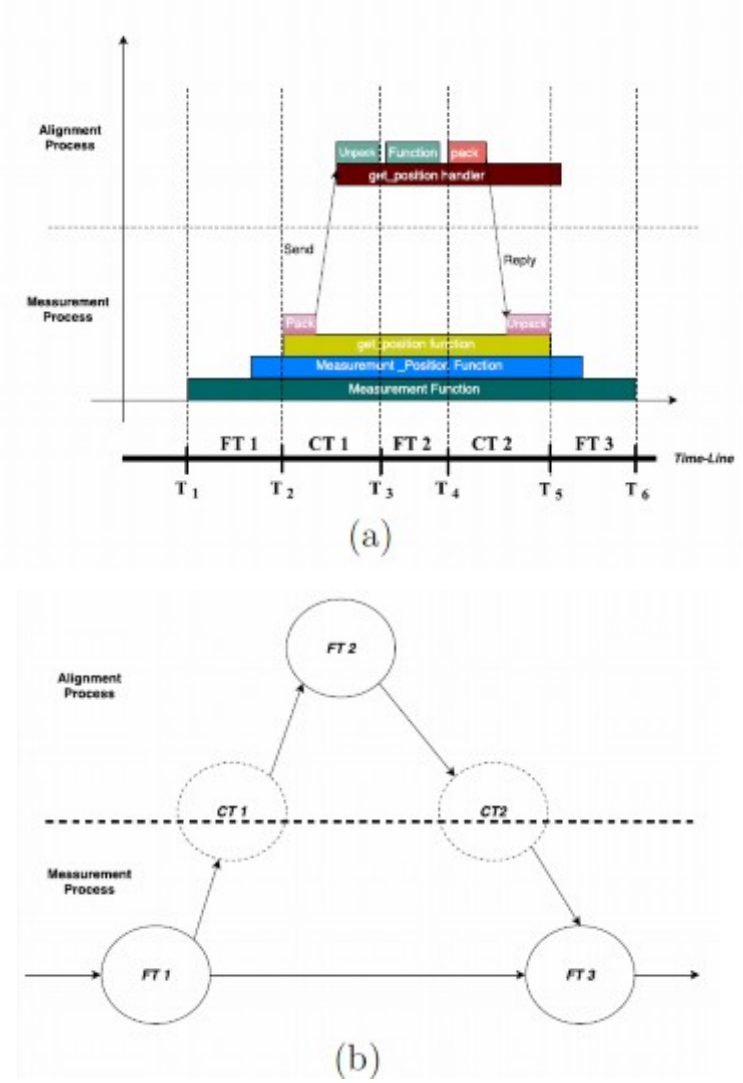
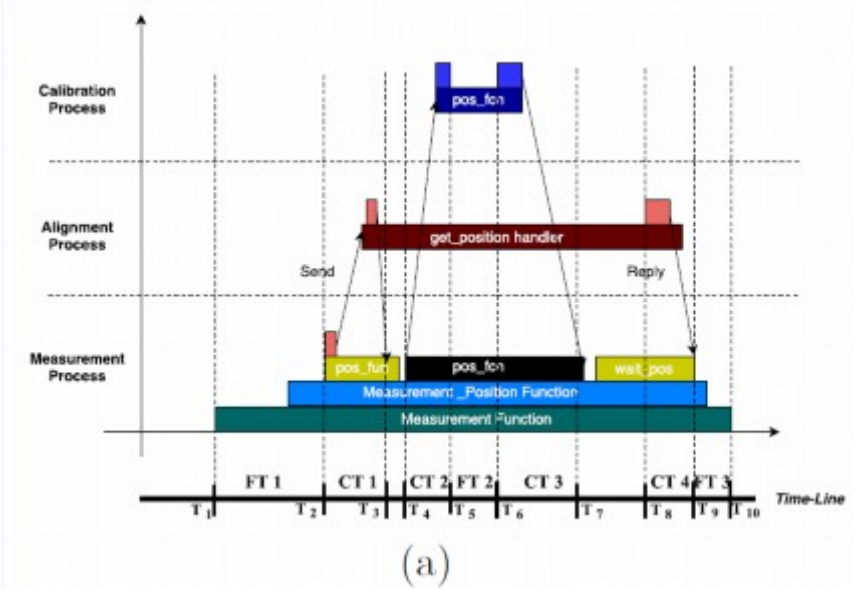


Рис. 3.3. (a) Позначка часу (b) DAG – блокування виклику

Під час перетворення DAG функціональними завданнями є FT1, FT2 і FT3, а комунікаційні завдання CT1 і CT2 вважаються вузлами. Виклики функцій розглядаються як залежності. DAG будується на основі інформації з функціонального завдання та їх міжпроцесного зв'язку. Цей DAG додатково покращується за рахунок часу, який витрачається кожною функцією як графік залежностей завдань (TDG). Оскільки виклики синхронних функцій є блокуючими викликами, FT1 спілкується через CT1 із FT2 , а FT1 чекає, поки не отримає відповідь від FT2 . Після отримання відповіді виконується FT3.

Сценарій 2

На рисунку 3.4 показано дерево викликів виклику асинхронної функції. Показаний виклик асинхронної функції, який описує сценарій запиту та очікування. Клієнт запитує сервер про функцію та продовжує виконувати іншу функцію в той же час. Після вказаного часу функція запускає тайм-аут і чекає, поки не отримає функцію. Подібно до Сценарію 1, функція та комунікаційні завдання представлені тим самим поняттям. Час, витрачений на виконання функціонального завдання FT1 , FT2 і FT3, є різницею T2 & T1, T4 & T3 і T10 & T9 відповідно. Час, витрачений на комунікаційне завдання CT1 , CT2 , CT3 і CT4, є різницею між T3 і T2 , T5 і T4 , T7 і T6 і T9 і T8 відповідно.



(a)

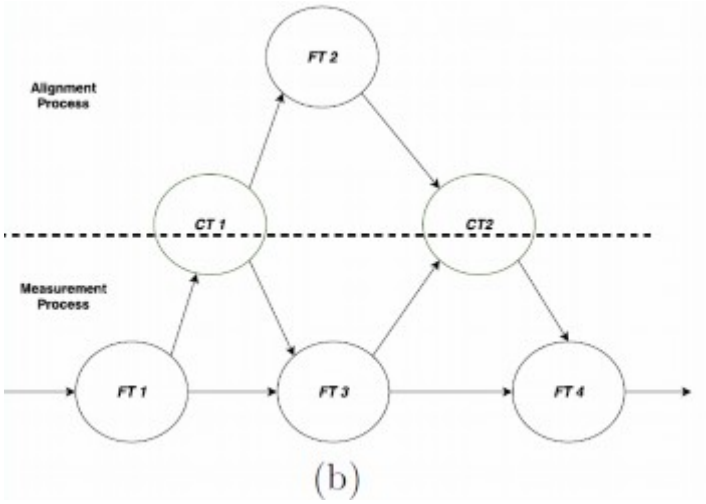


Рис. 3.4. Мітка часу – запит і очікування

Сценарій 2 дотримується виклику функції запиту та очікування. Трансформований FT1 є функцією, що виконується процесом вимірювання для повної функціональності вимірювання. Під час процесу він викликає процес вирівнювання для виконання функції. Під час виклику функції він пакує та надсилає певні дані для процесу вирівнювання, які потрібно обробити. Потім обробник виклику функції розпаковує отримані дані та виконує функцію. На відміну від синхронного виклику функції, він виконує інші функції паралельно з тим самим. У цьому сценарії FT1 викликає FT2 і FT1 зірки, виконуючи FT3 у той же час. Через певний час FT1 чекатиме на відповідь FT2 . Після отримання відповіді від FT3 FT4 обробляється.

Сценарій 3

Дерево викликів і DAG сповіщення про завершення функції показано на рисунку 3.5. Сповіщення про завершення функції — це асинхронний виклик функції, який виконує кілька процесів, доки не отримає відповідне сповіщення від потрібного процесу. Подібно до сценарію 1 і 2, функції та комунікаційні завдання представлені тим самим поняттям. Час, витрачений на виконання функціонального завдання FT1, FT і FT3, є різницею T2 & T1, T6 & T5 і T10 & T9 відповідно. Час, витрачений на комунікаційне завдання

CT1, CT2, CT3 і CT4, є різницею між T3 і T2, T5 і T4, T7 і T6 і T9 і T8 відповідно.

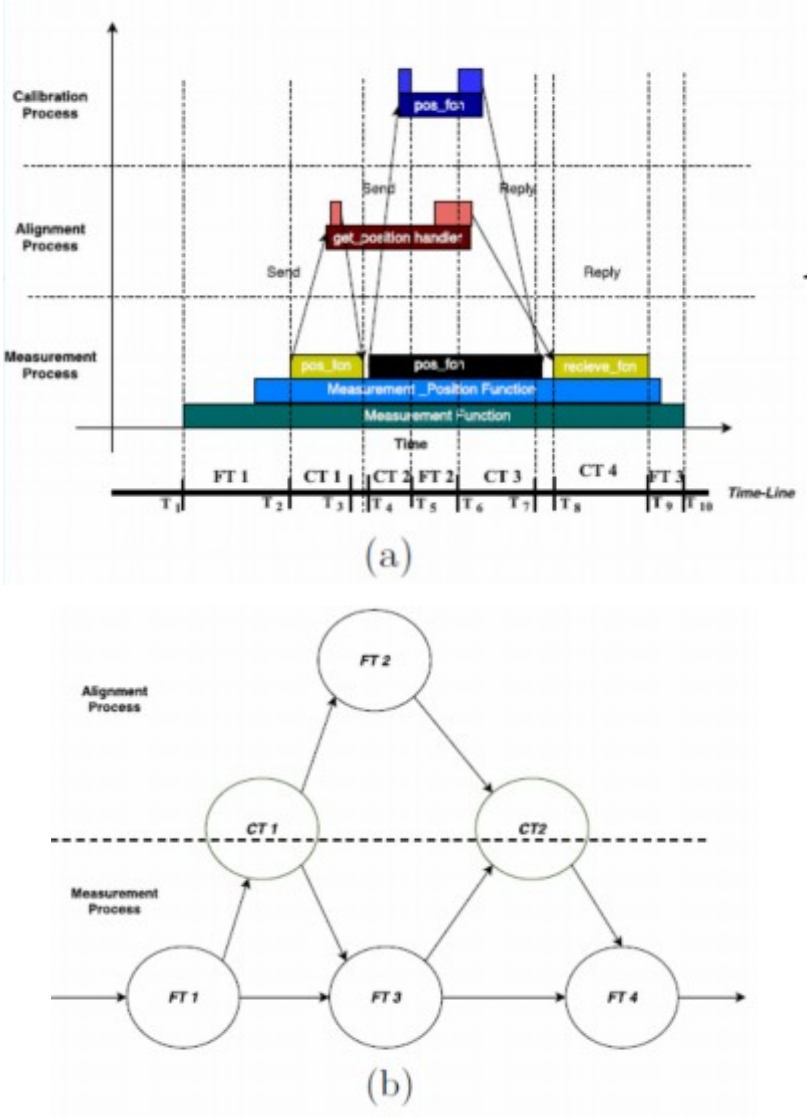


Рис. 3.5. Мітка часу – сповіщення про завершення функції

Поведінка fcn схожа на запит і очікування з однією істотною відмінністю. Запит і очікування ініціюють певний параметр часу, і якщо час перевищено, відповідь буде вважатися помилкою. З іншого боку, fcn не має параметра очікування, і він може виконувати будь-яку функцію, доки не отримає сповіщення про відповідь від запитуваного процесу. Оскільки поведінка fcn схожа на запит і очікування, перетворена DAG така сама, як запит і очікування. Трансформований граф показаний на рисунку 3.5 (b).

Сценарій 4

Окрім синхронного та асинхронного викликів, є ще два виклики, які називаються підпискою на події. Виклик події запитує службу для виконання певної функції. Підписка надає доступ до моніторингу даних сервера шляхом підписки на події сервера. Коли подія закінчується, підписка або скасовується, або втрачається. З точки зору позначок часу дерева викликів, час, який витрачається на виконання функціонального завдання FT1, FT2 і FT3, є різницею T2 & T1, T4 & T3 і T6 & T5 відповідно. Час, витрачений на комунікаційне завдання CT1, є різницею T3 & T2. Мітки часу застосовуються як до події, так і до підписки.

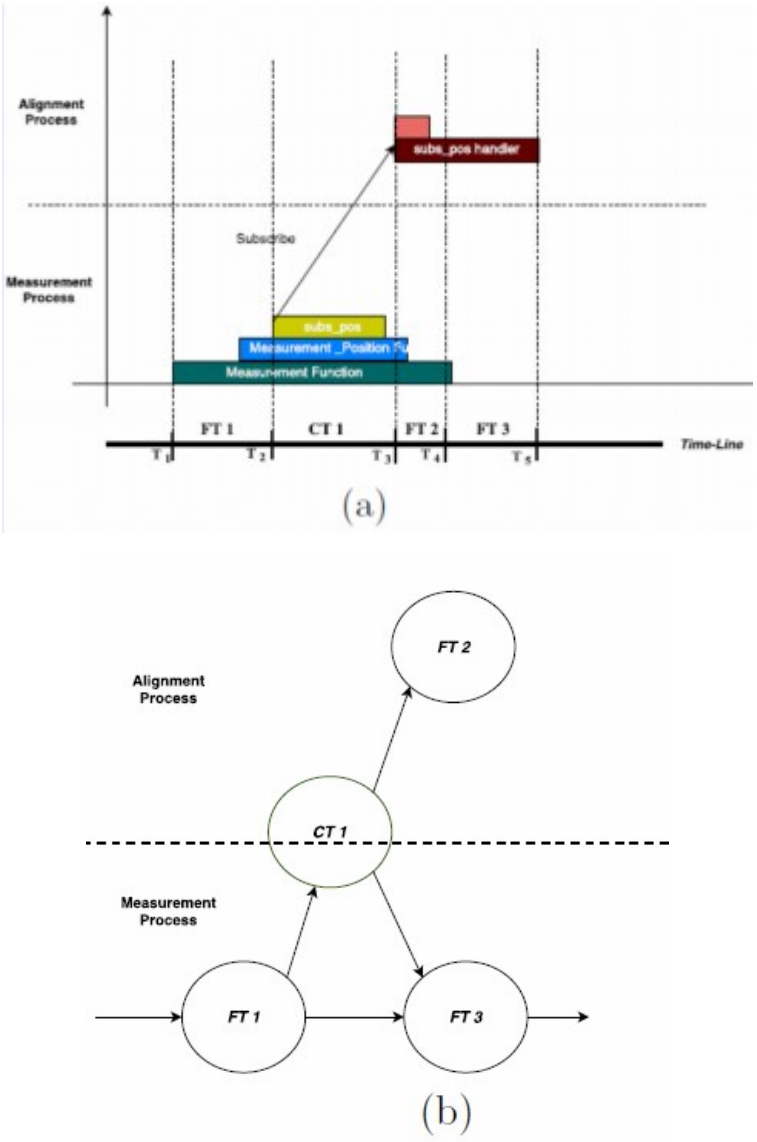


Рис. 3.6. Позначка часу – подія

Перетворення DAG подібне до інших сценаріїв. Під час виклику події FT1 запускає виклик події FT2 через CT1, а після ініціалізації події виконується FT3. Перетворення DAG підписки таке ж, як виклик події. Під час підписки можна викликати іншу функцію FT3, щоб скасувати підписку на подію.

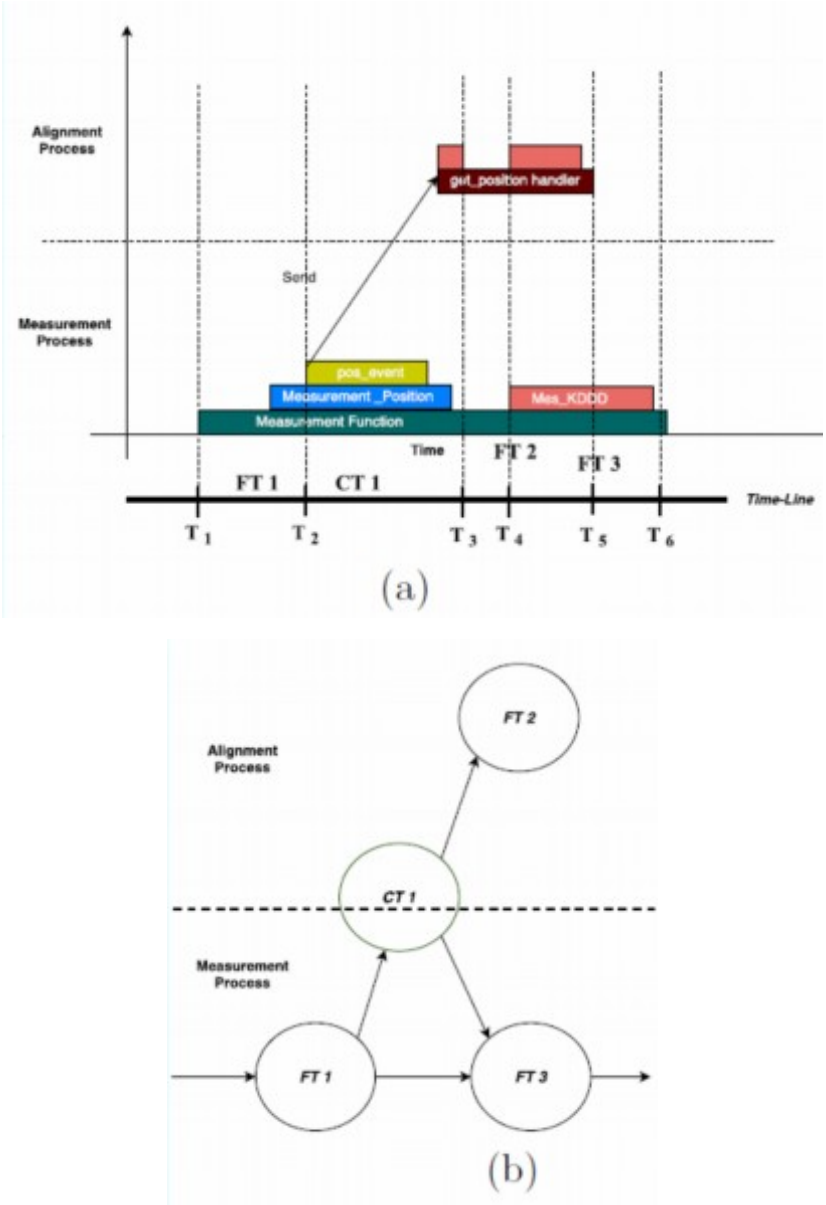


Рис. 3.7. Мітка часу – підписка

3.4. Представлення розкладу як результат аналітичного планування

Розклад є результатом аналітичного планувальника. Отримана інформація з моделі предметної області про завдання та їх залежності

реалізується в термінах DAG. Подібним чином інформація про платформу також витягується з моделі домену. Ця інформація надається аналітичному планувальнику (алгоритм звернення) для побудови розкладу завдань у часі відповідно до розподілу ресурсів. Розклад надає інформацію про виконання завдань у різні періоди часу та використання їх обчислювальних ресурсів. Використання ресурсів для кожного завдання динамічно розподіляється відповідно до їхнього значення якості та набору ввімкнених завдань.

Передбачуваність аналітичного планувальника визначається його передбачуваною здатністю. Передбачувана здатність аналітичного планувальника обчислюється шляхом порівняння отриманого розкладу від аналітичного планувальника з розкладом реального часу. Як згадувалося в розділі, розклад у реальному часі витягується з трасування в реальному часі. Розклад у реальному часі включає час виконання завдання, час зв'язку між завданнями, введення/виведення та інші практичні питання. Вони порівнюються з розкладом аналітичного планувальника, який містить лише інформацію про виконання та зв'язок.

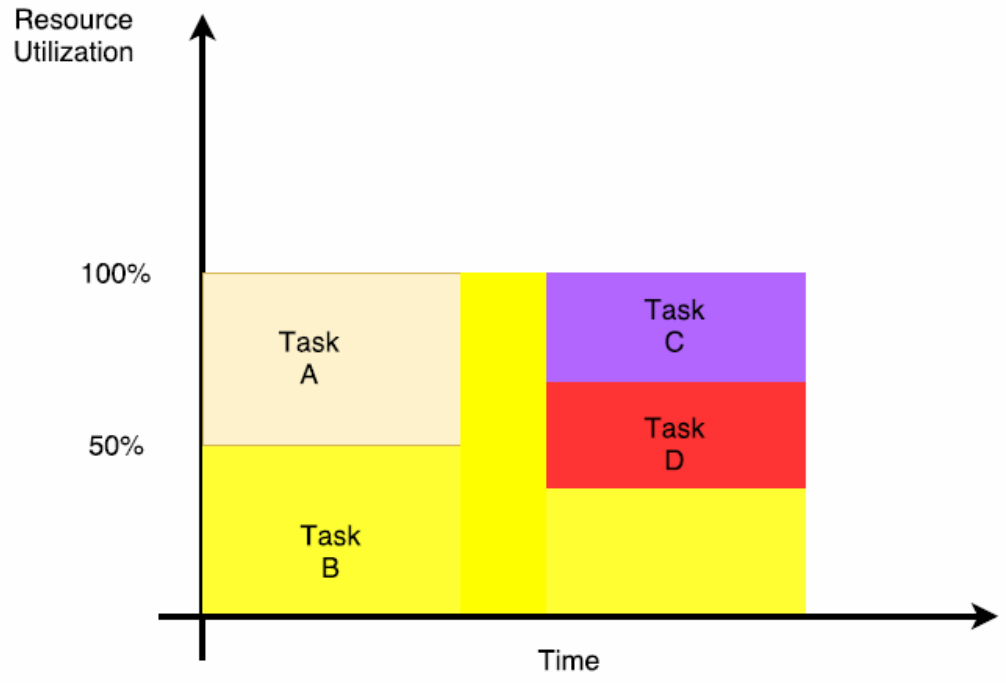


Рис. 3.8. Очікуваний графік

Екземпляр розкладу показано на рисунку 3.8. На даному рисунку показано графік виконання завдань a, b, c і d. Припускається, що значення акуратності для всіх завдань дорівнює 0 (за замовчуванням). У цьому випадку завдання a і b виконуються паралельно, рівномірно розподіляючи ресурс. Оскільки завдання a завершується раніше завдання b, завдання b використовує весь ресурс, поки не прийде нове завдання. Якщо завдання c і d увімкнуто разом, завдання a, c і d виконуються з рівними ресурсами з 33% від загального ресурсу.

Після визначення передбачуваної потужності аналітичного планувальника параметри, які можна налаштувати, можна використовувати для дослідження майбутнього, тобто можна виконати оптимальне відображення на однорідному мультипроцесорі. Крім того, настроювані параметри додатково використовуються для підвищення швидкості.

Точність аналітичного планувальника визначається його передбачуваною здатністю. Як пояснюється в розділі 3.1, потужність прогнозування вказує на те, наскільки добре він передбачив минуле, імітуючи планувальник Linux. Якщо передбачувана потужність аналітичного планувальника висока, тоді настроювані параметри в аналітичному планувальнику можна змінити для досягнення кращого розкладу.

У цьому планувальнику розглядаються проблеми підходу Y-діаграми, такі як відображення та платформа. Поділ проблем допомагає вивчити альтернативну конфігурацію та визначає покращену. Покращена конфігурація також допомагає отримати кращий розклад із мінімальним часом виконання та більшим розподілом ресурсів ЦП.

3.5. Перевірка отриманих результатів

Передбачуваність аналітичного планувальника визначається його передбачуваною здатністю. Передбачувана потужність аналітичного

планувальника обчислюється шляхом порівняння розкладу в реальному часі з розкладом, створеним аналітичним планувальником.

Перевірка розкладу в реальному часі та розкладу з аналітичного планувальника здійснюється з використанням ефективного часу виконання функції. Ефективний час виконання використовується для визначення потужності прогнозування шляхом ділення часу виконання розкладу в реальному часі та часу виконання аналітичного планувальника.

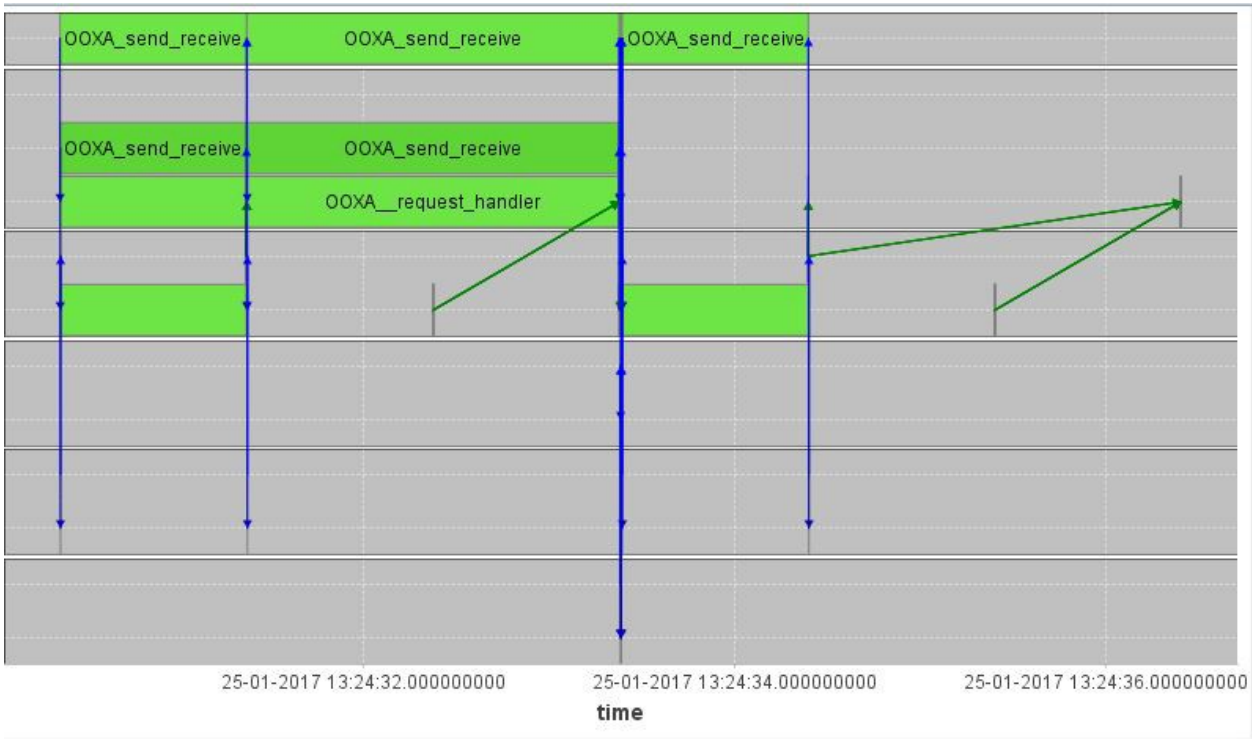


Рис. 3.9. Трасування в реальному часі

На рисунку 3.1 показано трасування в реальному часі. Перевірка розкладу в реальному часі та розкладу з аналітичного планувальника здійснюється з використанням ефективного часу виконання функції. Ефективний час виконання використовується для визначення потужності прогнозування шляхом ділення часу виконання розкладу в реальному часі та часу виконання аналітичного планувальника.

Перевірка буде виконана за допомогою двох тестів. Випадок 1 представляє виклик синхронної функції як блокуючий виклик. Випадок 2

представляє виклик асинхронної функції як виклик підписки на подію. Тестові набори — це віддалені виклики процедур, які розгортаються в однопоточному процесі. У наборах тестів також немає одночасних завдань. Крім того, припускається, що пріоритет завдань на тестових наборах встановлений за замовчуванням, тобто 0, і процес відображається на однорідному мультипроцесорі. Таким чином, увімкнені завдання виконуються одна за одною з повним використанням ресурсів ЦП.

Випадок 1: Блокування виклику

Синхронні виклики функцій – це виклики функцій, у яких після того, як клієнт надсилає запит серверу, клієнт блокується, доки не буде надано відповідь із сервера. Таким чином, клієнт чекатиме відповіді від сервера, не виконуючи інших завдань. Стандартним інтерфейсом в архітектурі ASML є синхронний виклик функції. Виклик синхронної функції може уникнути інших можливих накладних витрат у спілкуванні. З іншого боку, виклик функції може перешкоджати прогресу інших функцій, що зрештою робить виконання менш ефективним.

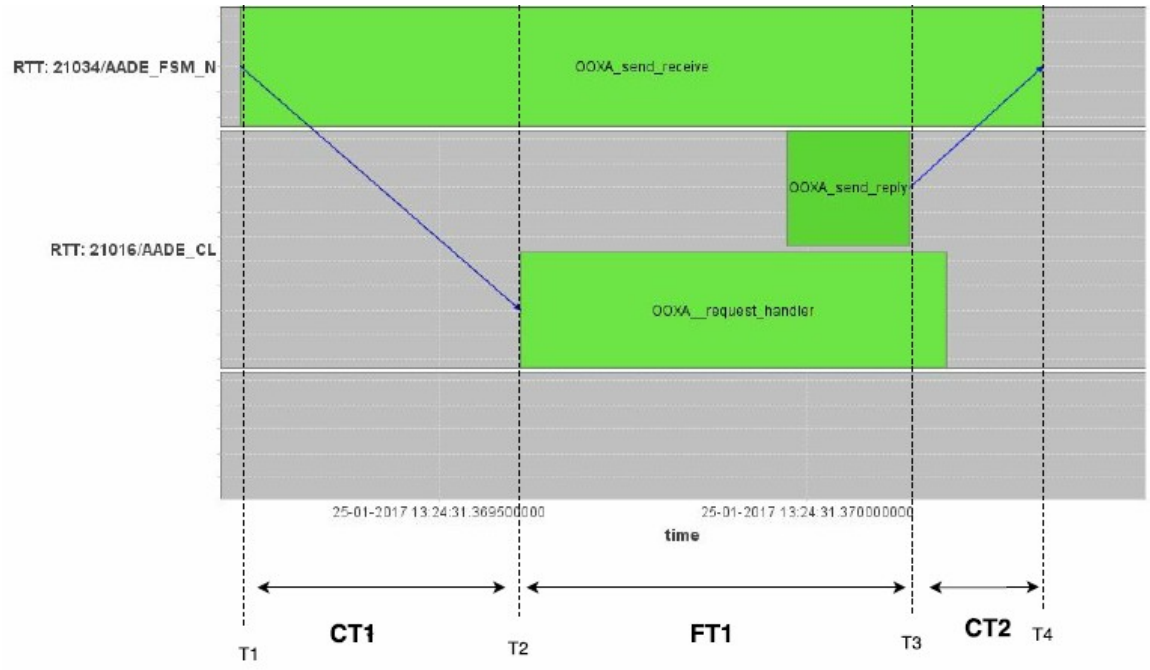


Рис. 3.10. Блокування виклику

На малюнку 3.10 показано блокуючий виклик між процесами в трасах реального часу. На рисунку показано, що процеси 1 і 2 взаємодіють за допомогою віддаленого виклику процедури. У цьому тестовий набір, процес1 і процес 2 діють як сервер і клієнт відповідно.

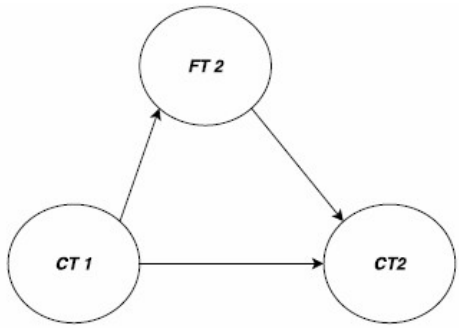


Рис. 3.11. Блокуючий виклик – DAG

Замість розкладу в реальному часі використовуються тестові моделі з ефективним часом виконання. Припускається, що значення акуратності для аналітичного планувальника та розкладу в реальному часі встановлено за замовчуванням, тобто дорівнює нулю. Крім того, результати в реальному часі беруться з однопотокового процесу без одночасних завдань. Тому час виконання завдання такий самий, як і час виконання. Ефективний час виконання наведено в таблиці 3.1. Витрачений час представлений у наносекундах.

Таблиця 3.1.

Ефективний час виконання завдання

Завдання	**Час (нс)**
FT1	362000
CT1	379000
CT2	347000

На рисунку 3.11 описано графік блокування виклику. Оскільки ефективний час виконання та час виконання однакові, накладні витрати,

пов’язані з трасуваннями в реальному часі, вважаються змінами. Таким чином, використання процесора приблизно становить 95-96% завдяки врахуванню накладних витрат. Таким чином, прогнозованість того ж розкладу досягає близько 96%.

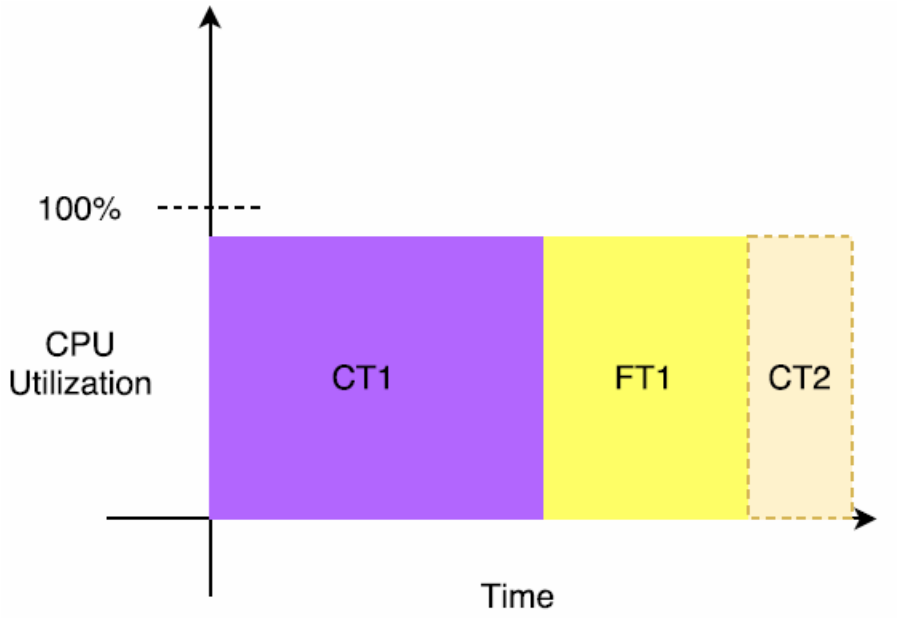


Рис. 3.11. Розклад блокуючого виклику

Випадок 2: Неблокуючий виклик

Окрім синхронного та асинхронного зв’язку, клієнту дозволено контролювати дані сервера за підпискою. Подія є одним із таких механізмів виклику функції. Крім того, процесам дозволено викликати подію в іншому процесі за допомогою викликів подій. Це запитає службу на сервері для виконання певної функції. Підписка надає доступ до моніторингу даних сервера шляхом підписки на події сервера. Щоразу, коли сервер викликає подію, функція клієнта буде повідомлена. Запити на підписку додаються до програми з деталями клієнта та сервера як параметр. Коли процес зупиняється, підписки або скасовуються або втрачаються.

Замість розкладу в реальному часі використовуються тестові моделі з ефективним часом виконання. Припускається, що значення акуратності для аналітичного планувальника та розкладу в реальному часі встановлено за

замовчуванням, тобто дорівнює нулю. Це призводить до того, що ефективний час виконання дорівнює його часу виконання.

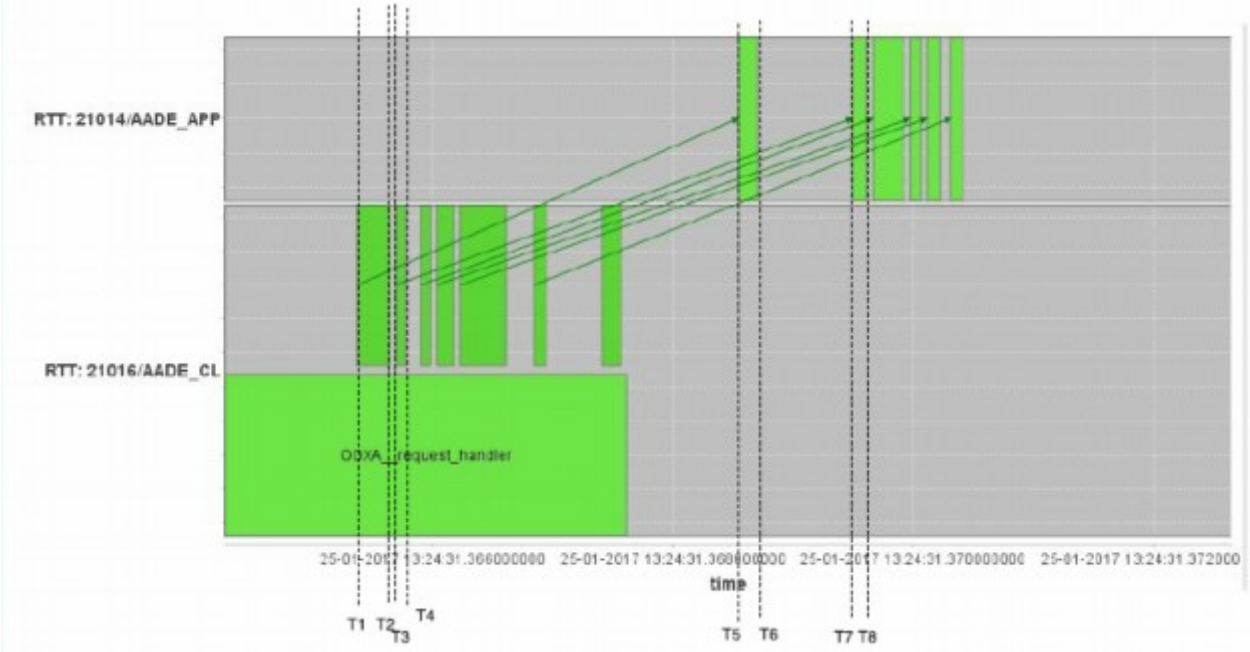


Рис. 3.12. Неблокуючий виклик

Ефективний час виконання показано в таблиці 3.2 і представлено в наносекундах (нс).

Таблиця 3.2.

Ефективний час виконання завдання

Завдання	**Час (нс)**
FT1	270000
FT2	80000
FT3	161000
FT4	117000
CT1	2902000
CT2	3728000

Рисунок 3.13 описує графік підписки на подію. Графік показує, що FT2 і CT1 виконуються одночасно. Однак однопотоковий процесор може

виконувати лише один процес за раз, що залишає ефективний час виконання рівним часу виконання. Тому завантаження процесора оцінюється приблизно на рівні 95-96% через врахування накладних витрат.

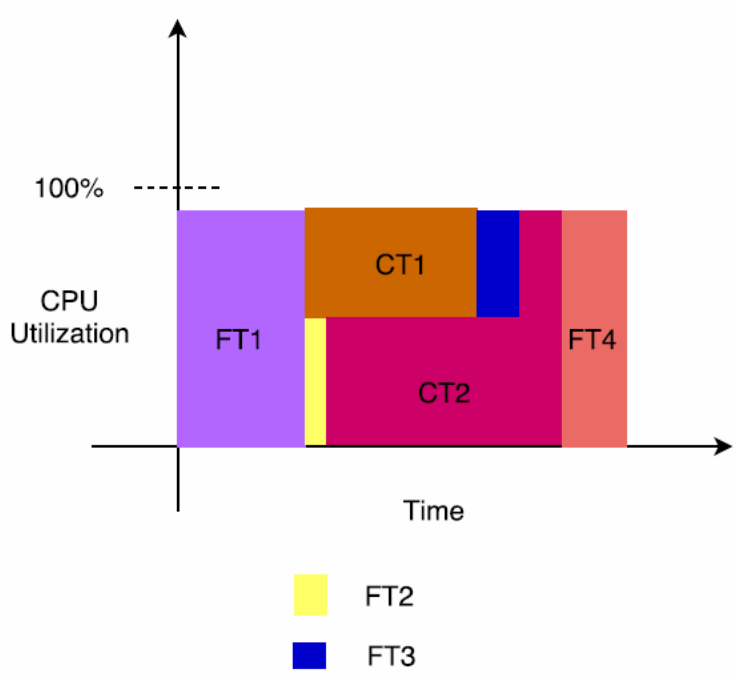


Рис. 3.13. Розклад неблокуючого виклику

Оскільки процес однопотоковий, одночасні завдання виконуються з поділом часу. Таким чином, передбачуваність цього ж графіка досягає близько 96%.

Графік перевірених синхронних і асинхронних викликів перевіряється стосовно графіка реального часу. Ефективний час виконання кожного завдання у отриманому графіку порівнюється з часом виконання завдань у реальних трасах. Порівняння виконується шляхом визначення співвідношення отриманого графіка до графіка реального часу. Визначене співвідношення надає прогностичну потужність аналітичного планувальника. Прогностична потужність аналітичного планувальника є агрегованим співвідношенням, отриманим для різних тестових наборів. У цій магістерській роботі розглянуто два тестові набори для визначення

прогностичної потужності планувальника. Ці два набори даних включають блокуючий виклик (синхронний виклик) та виклик підписки на події (асинхронний виклик).

Прогностична потужність тестового випадку 1 становить приблизно 95%, а тестового випадку 2 — приблизно 96%, і агрегована прогностична потужність буде близько 95,5%. Це пояснюється накладними витратами, такими як обчислення завдань комунікації, які виконуються на комунікаційних платах, та накладними витратами під час пакування/розпакування.

Висновки до розділу

У цьому розділі досліджується часовий аналіз вилучених артефактів за допомогою параметричної аналітичної моделі. Необхідна інформація з моделі цільового перегляду витягується та аналізується за допомогою запропонованого аналітичного планувальника. Інформація з прикладного та процесуального рівня реалізована як дерево викликів. Дерево викликів розділено на функціональне завдання та комунікаційне завдання, і вони реалізовані як спрямований ациклічний граф, що доповнюється відповідним ефективним часом виконання.

ВИСНОВКИ

В магістерській роботі виконано дослідження процесів оптимізації моделей виконання програмних додатків. Описано методику побудови моделей для складних систем, яка передбачає використання модульного підходу з акцентом на взаємодію окремих компонентів. Запропоновано алгоритми, що дозволяють адаптувати методи моделювання до специфіки конкретної системи, з урахуванням її функціональних і структурних вимог.

Виконано огляд наукових джерел, пов’язаних з моделюванням програмних систем. Проаналізовано сучасні підходи, які використовуються в галузі, та окреслено основні напрями розвитку моделей для складних програмних додатків

Для досягнення поставленої мети здійснено систематичний аналіз та реконструкцію доменних моделей, що були відновлені на основі артефактів. Детально досліджено структуру та поведінкові характеристики різних рівнів моделі, з акцентом на їхню функціональну взаємодію в межах визначеного домену.

Запропоновано науковий метод аналізу часових характеристик високотехнологічної системи шляхом збору релевантної інформації з її функціонування. Представлений метод ґрунтується на підході, керованому моделлю (Model-Driven Approach), що дозволяє збирати дані з працюючої машини та перетворювати їх у моделі для подальшого аналізу. Використання модельно-керованої архітектури (MDA) продемонструвало свою ефективність у трансформації даних для візуалізації критичних параметрів системи. Успішне застосування моделей підтвердило їхню здатність прогнозувати поведінку високотехнологічних систем та оптимізувати їхню продуктивність шляхом вибору оптимальної конфігурації.

Представлено формальний підхід до параметричного аналізу, який використовувався для визначення оптимальних параметрів та конфігурацій

системи. Розглянуто теоретичні основи моделі та формалізовано алгоритми для обчислення ключових параметрів.

Використано методології верифікації аналітичного методу та інтерпретації отриманих результатів. Описано процес порівняння прогнозованих даних з емпіричними вимірюваннями для підтвердження точності моделі.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bart Kienhuis, Ed F Deprettere, Pieter Van Der Wolf, Kees Vissers A methodology to design programmable embedded systems. Embedded processor design challenges, 2002.
2. Twan Basten, Martijn Hendriks, Lou Somers, Nikola Trecka Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems. Model-Based Design of Adaptive Embedded Systems, 2013.
3. Y. Lei, M. P. Singh An Evaluation of E-Business Metamodels. Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2005.
4. Roman Taborsky and Valentino Vranic Feature Model Driven Generation of Software Artifacts. Model-Based Design of Adaptive Embedded Systems, 2013.
5. A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen and C. Riva. Symphony View-Driven Software Architecture Reconstruction. IEEE/IFIP Conference on Software Architecture, 2004.
6. J. Desfossez. “Résolution de problème par suivi de métriques dans les systèmes virtualisés”. Master thesis, École Polytechnique de Montréal, Qc, 2011.
7. M. Couture, M. Dagenais, D. Toupin, R. Charpentier, G. Matni, M. Desnoyers, P.-M. Fournier. “Monitoring and tracing of critical software systems - State of the work and project definition”. DRDC – Valcartier Research Centre, Technical Memorendum, TM 2008-144, 2008.
8. K. Yaghmour. “Analyse de performance et caractérisation de comportement à l’aide d’enregistrement d’événements noyau”. Master thesis, École Polytechnique de Montréal, Qc, 2001.
9. The Tracing Monitoring Framework (TMF). Web sites (last accessed July, 2015). http://dmct.dorsal.polymtl.ca/node/27

10. D. Thibault. “LTTng: The Linux Trace Toolkit Next Generation—A Comprehensive User’s Guide.” DRDC – Valcartier Research Centre. Submitted to DRDC Editorial Office. Another source of information can be found at the following Web site. http://lttng.org/docs/#doc-getting-started
11. H. Aljamaan, T. Lethbridge, O. Badreddin, G. Guest, and A. Forward. “Specifying Trace Directives for UML Attributes and State Machine.” In the 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Lisbon, Portugal, Jan. 2014.
12. M. A. Garzón Torre. “Reverse Engineering Object-Oriented Systems into Umple: An Incremental and Rule-Based Approach.” Ph.D. thesis, University of Ottawa, 2015.
13. M. Couture, F. Lajeunesse-Robert, F. Prenoveau, M. Dagenais, B. Ktari. “The Tracing, monitoring and analysis of distributed multi-core systems - Selected feasibility studies”. DRDC – Valcartier Research Centre, Technical Report, TR 2008-300, 2009.
14. A. Duda, G. Harrus, Y. Haddad, and G. Bernard. “Estimating global time in distributed systems.” Proceedings of the 7th International Conference on Distributed Computing Systems, Berlin, vol. 18, 1987.
15. B. Poirier. “Synchronisation de traces distribuées à l'aide d'événements de bas niveau.” Master thesis, École Polytechnique de Montréal, Apr. 2010. Another source of information can be found at the following Web site http://dmct.dorsal.polymtl.ca/node/12
16. B. Poirier, R. Roy, and M. Dagenais. “Accurate offline synchronization of distributed traces using kernel-level events.” Operating Systems Review, 2010.
17. M. Jabbarifar, A. S. Sendi, A. Sadighian, N. E. Jivan, and M. Dagenais. “A reliable and efficient time synchronization protocol for heterogeneous wireless sensor network.” Wireless Sensor Network, Aug. 2011.
18. M. Jabbarifar, M. Dagenais. “On Line Trace Synchronization for Large Scale Distributed Systems." Ph.D. Thesis, École Polytechnique de Montréal,

2013. Another source of information can be found at the following Web site http://dmct.dorsal.polymtl.ca/node/12

19. A. Montplaisir. “Stockage sur disque pour accès rapide d'attributs avec intervalles de temps.” Master thesis, École Polytechnique de Montréal, Dec. 2011. Another source of information can be found at the following Web site http://dmct.dorsal.polymtl.ca/node/25
20. W. Fadel. “Techniques for the Abstraction of System Call Traces to Facilitate the Understanding of the Behavioural Aspects of the Linux Kernel.” Master thesis, Concordia University, 2012.
21. R. M. Rangayyan, R. Gordon and A. P. Dhawan, "Algorithms for limited-view computed tomography: An annotated bibliography and a challenge", Digest Topic. Meet. Indust. Appl. Comput. Tomog. NMR Imaging Opt. Soc. America, 1984-Aug.
22. Boverman, G., Kao, T.-K., Kulkarni, R., Kim, B. S., Isaacson, D., Saulnier, G. J., and Newell, J. C., Robust linearized image reconstruction for multifrequence eit of the breast, IEEE Trans. Med. Im., 27:1439–1448, 2008.
23. Tidswell, A. T., Gibson, A., Bayford, R. H., and Holder, D. S., Validation of a 3d reconstruction algorithm for eit of human brain function in a realistic head-shaped tank, Physiol. Meas., 22:117–185, 2001.
24. I.Frerichs, Electrical impedance tomography (eit) in applications related to lung and ventilation: a review of experimental and clinical activities, Physiol. Meas., 21:R1–R21, 2000.
25. Frerichs, I., Hinz, J., Herrmann, P., Weisser, G., Hahn, G., Dudykevych, T., Quintel, M., and Hellige, G., Detection of local lung air content by electrical impedance tomography compared with electron beam ct, J. Appl. Physiol, 93:660–666, 2002.
26. Victorino, J., Borges, J. B., Okamoto, V. N., Matos, G. F. J., Tucci, M. R., Caramez, M. P. R., Tanaka, H., Sipmann, F. S., Santos, D. C. B., Barbas, C.

S. V., Carvalho, C. R. R., and Amato, M. B. P., Imbalances in regional lung ventilation, Am. J. Respir. Crit. Care. Med., 169:791–800, 2004.

27. Cheney, M., Isaacson, D., and Newell, J. C., Electrical impedance tomography, SIAM Rev, 41:85–101, 1999.
28. Kaipio, J. P., Kolehmainen, V., Somersalo, E., and Vauhkonen, M., Statistical inversion and Monte Carlo sampling methods in electrical impedance tomography, Inv. Probl., 16:1487–1522, 2000
29. Kolehmainen, V., Lassas, M., and Ola, P., Electrical impedance tomography problem with inaccurately known boundary and contact impedances, IEEE Trans. Med. Im., 27:1404–1414, 2008.
30. Kolehmainen, V., Lassas, M., and Ola, P., The inverse conductivity problem with an imperfectly known boundary in three dimensions, SIAM J. Appl. Math., 67:1440–1452, 2007.
31. Lionheart, W. R. B., Boundary shape and electrical impedance tomography, Inv. Probl., 14:139–147, 1998.
32. Nissinen, A., Kolehmainen, V., and Kaipio, J. P., Compensation of modelling errors due to unknown domain boundary in electrical impedance tomography, IEEE Trans. Med. Im., 30:231–242, 2011.
33. Kouroshfar, E.; Mirakhorli, M.; Bagheri, H.; Xiao, L.; Malek, S.; Cai, Y. A Study on the Role of Software Architecture in the Evolution and Quality of Software. In Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories, Florence, Italy, 16–17 May 2015; pp. 246–257.
34. Bass, L.; Clements, P.; Kazman, R. Software Architecture in Practice; Addison-Wesley: Boston, MA, USA, 1998.
35. Kruchten, P. The4+1 view model of architecture. IEEE Softw. 1995, 12, 42–50.
36. Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Merson, P.; Nord, R.; Stafford, J. Documenting Software Architectures: Views and Beyond; Pearson Education: London, UK, 2010.

37. Garcia, J.; Popescu, D.; Edwards, G.; Medvidovic, N. Identifying architectural bad smells. In Proceedings of the 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, Germany, 24–27 March 2009; pp. 255–258.
38. Hochstein, L.; Lindvall, M. Combating architectural degeneration: A survey. Inf. Softw. Technol. 2005, 47, 643–656.
39. Mirakhorli, M. Software architecture reconstruction: Why? What? How? In Proceedings of the IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, Canada, 2–6 March 2015; p. 595.
40. Pollet, D.; Ducasse, S.; Poyet, L.; Alloui, I.; Campan, S.; Verjus, H. Towards a process-oriented software architecture reconstruction taxonomy. In Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR'07), Amsterdam, The Netherlands, 21–23 March 2007; pp. 137–148.
41. Harman, M.; Jones, B.F. Search-based software engineering. Inf. Softw. Technol. 2001, 43, 833–839.
42. Geem, Z.W.; Kim, J.H.; Loganathan, G. A new heuristic optimization algorithm: Harmony search. Simulation 2001, 76, 60–68.
43. Toklu, Y.C.; Bekdaş, G.; Geem, Z.W. Harmony Search Optimization of Nozzle Movement for Additive Manufacturing of Concrete Structures and Concrete Elements. Appl. Sci. 2020, 10, 4413.
44. B. Byram and M. Jakovljevic, "Ultrasonic multipath and beamforming clutter reduction: A chirp model approach", IEEE Trans. Ultrason. Ferroelectr. Freq. Control, vol. 61, no. 3, pp. 428-440, Mar. 2014.
45. B. Byram, K. Dei, J. Tierney and D. Dumont, "A model and regularization scheme for ultrasonic beamforming clutter reduction", IEEE Trans. Ultrason. Ferroelectr. Freq. Control, vol. 62, no. 11, pp. 1913-1927, Nov. 2015.

46. K. Dei and B. Byram, "The impact of model-based clutter suppression on cluttered aberrated wavefronts", IEEE Trans. Ultrason. Ferroelectr. Freq. Control, vol. 64, no. 10, pp. 1450-1464, Oct. 2017.
47. B. Dsp, GPU vs FPGA performance comparison, 2016, [online] Available: https://www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_Performance_Comparison_v1.0.pdf.
48. M. A. Lediju, G. E. Trahey, B. C. Byram and J. J. Dahl, "Short-lag spatial coherence of backscattered echoes: Imaging characteristics", IEEE Trans. Ultrason. Ferroelectr. Freq. Control, vol. 58, no. 7, pp. 1377-1388, Jul. 2011.