Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Моделі процесу імітаційного моделювання складних розподілених комп'ютерних систем

(тема)

Виконав:

студент II курсу, групи СПзм-21-1

Лемішко Д.В.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Волк М.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Лемішку Дмитру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі процесу імітаційного моделювання складних розподілених комп'ютерних систем

затверджена наказом по університету від " 24 " березня 2023 р. № 60 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 17 травня 2022 р.

3. Вхідні дані до роботи

4. Перелік питань, що потрібно опрацювати у роботі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Презентація 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	27.03.23 – 06.04.23	
2	Розробка моделей	06.04.23 – 12.04.23	
3	Реалізація алгоритмів	13.04.23 – 20.04.23	
4	Розробка структури програмних засобів	21.04.23 – 25.04.23	
5	Розробка програмних модулів	26.04.23 – 30.04.23	
6	Оформлення матеріалів атестаційної роботи	01.05.23 – 10.05.23	
7	Подання атестаційної роботи керівникові та її	11.05.23 – 14.05.23	
	попередній захист		
8	Подання атестаційної роботи на рецензування	14.05.23 – 17.05.23	

Дата видачі завдання 27 березня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Волк М.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 72 с., 14 рис., 1 дод., 61 джерело.

ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ, ПОВЕДІНКОВА МОДЕЛЬ, МАЙСТЕР-АЛГОРИТМ, ГЕТЕРОГЕННІ КОМП'ЮТЕРНІ СИСТЕМИ

Кіберфізичні системи – це складні інженерні системи, де обчислювальні частини взаємодіють між собою, а фізичні частини складають середовище. Щоб врахувати зростаючу складність таких систем, їх зазвичай розкладають на різні частини, які моделюють різні експерти, можливо, з різних організацій. Ці експерти використовують мову, адаптовану до їхньої проблеми, як синтаксично, так і семантично. На цьому етапі, з одного боку, це призводить до виконуваних моделей кіберчастин і виконуваних моделей фізичних частин, а з другого боку, виконувані моделі повинні зберігати інтелектуальні властивості та зазвичай використовуються як одиниці моделювання чорної скриньки. Однак, щоб зрозуміти поведінку всієї системи, потрібно виконати спільне моделювання.

Метою роботи є забезпечення інтерфейсу на рівні моделі, який охоплює як кібернетичні, так і фізичні частини кіберфізичної системи, і визначити мову, призначену для координації таких частин. На їх основі та оригінального інтерфейсу спільного моделювання забезпечити можливість автоматичного створення точної та продуктивної розподіленої інфраструктури для спільного моделювання.

ABSTRACT

Master’s thesis: 72 pages, 14 figures, 1 appendice, 61 sources.

SIMULATION, BEHAVIORAL MODEL, MASTER ALGORITHM, HETEROGENEOUS COMPUTER SYSTEMS.

Cyber-physical systems are complex engineering systems where computational parts interact with each other, and physical parts make up the environment. To account for the increasing complexity of such systems, they are usually broken down into different parts that are modeled by different experts, possibly from different organizations. These experts use language adapted to their problem, both syntactically and semantically. At this stage, on the one hand, it leads to executable models of cyber parts and executable models of physical parts, and on the other hand, executable models must retain intellectual properties and are usually used as black-box simulation units. However, to understand the behavior of the entire system, joint simulations must be performed.

The goal of the work is to provide a model-level interface that spans both the cybernetic and physical parts of a cyber-physical system, and to define a language designed to coordinate such parts. Based on them and the original collaborative modeling interface, provide the ability to automatically create an accurate and productive distributed infrastructure for collaborative modeling.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ADL – Architecture Description Languages

API – Application Programming Interface

CL – Control Language

CPS – Cyber Physical System

CT – Central Time

DAE – Differential-Algebraic Equations

DE – Discrete Event

DSL – Domain-Specific Language

FMI – Functional Mock-up Interface

IDE – Integrated Development Environment

IP – Internet Protocol

HLA – High Level Architecture

MCI – Media Control Interface

MEMS – Micro-Electro-Mechanical Systems

MDE – Model-Driven Engineering

ODE – Ordinary Differential Equation

PDE – Partial Differential equation

SU – Simulation Unit

ВСТУП

Кіберфізичні системи – це складні інженерні системи, де обчислювальні частини взаємодіють між собою, а фізичні частини складають середовище. Щоб врахувати зростаючу складність таких систем, їх зазвичай розкладають на різні частини, які моделюють різні експерти, можливо, з різних організацій. Ці експерти використовують мову, адаптовану до їхньої проблеми, як синтаксично, так і семантично. На цьому етапі, з одного боку, це призводить до виконуваних моделей кіберчастин і виконуваних моделей фізичних частин, а з другого боку, виконувані моделі повинні зберігати інтелектуальні властивості та зазвичай використовуються як одиниці моделювання чорної скриньки. Однак, щоб зрозуміти поведінку всієї системи, потрібно виконати спільне моделювання.

Проблеми з існуючими підходами різноманітні. По-перше, одиниці моделювання, які представлені у вигляді чорних ящиків, не дозволяють врахувати семантичні особливості кожної моделі. По-друге, спільне моделювання нині в основному базується на інтерфейсі прикладного програмування, керованому часом, який, як було показано, вводить «штучні затримки» при застосуванні до кіберфізичних систем. Такі затримки означають погану точність, що може зробити результати спільного моделювання недійсними. Крім того, зменшення таких затримок спільного моделювання означає погану загальну продуктивність спільного моделювання. По-третє, визначення точної та продуктивної структури спільного моделювання може бути алгоритмічно складним, тому потрібна підтримка для підвищення рівня абстракції при визначенні координації.

Метою роботи є забезпечення інтерфейсу на рівні моделі, який охоплює як кібернетичні, так і фізичні частини кіберфізичної системи, і визначити мову, призначену для координації таких частин. На їх основі та

оригінального інтерфейсу спільного моделювання забезпечити можливість автоматичного створення точної та продуктивної розподіленої інфраструктури для спільного моделювання.

Робота містить реалізацію двох предметно-орієнтованих мов. Одна для визначення інтерфейсу координації моделі та друга для визначення специфікації координації моделі. Крім того, ця в роботі визначено прототип семантичного API, представленого як узагальнення існуючих стандартних API. Нарешті, був реалізований компілятор, щоб координації, визначені запропонованими мовами, могли використовуватися для автоматичного створення розподіленої структури спільного моделювання. Різні пропозиції застосовуються до прикладу, де чітко проілюстровано переваги підходу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз літератури

Зростаюча складність систем робить їх проектування надзвичайно складним завданням, яке потребує передових інженерних методів для проектування, розробки, створення та обслуговування цих систем. Швидкий технологічний прогрес у 21-му столітті та попит на більш інтегровані та взаємопов’язані системи спонукають інженерів до розробки таких систем, як «розумні міста», мережеві медичні пристрої, передові автоматизовані авіаційні системи, критичне керування електроенергією, інтегровані системи захисту. Ці системи зазвичай поєднують кібер-можливості (обчислення, управління та комунікації) з фізичними процесами (електричні, механічні або хімічні). Наприклад, автоматичний пілот - це система, яка контролює траєкторію літака, автомобіля або судна без постійного контролю з боку людини-оператора. Транспортний засіб фізично рухається в просторі за траєкторією, яка визначається комп’ютеризованим алгоритмом керування, який регулює її за допомогою приводів на основі показань датчиків фізичного простору. Такі системи є кіберфізичними системами (CPS).

Оскільки CPS об’єднує кібернетичну та фізичну частини, нам потрібно знати обидві, щоб зрозуміти нову поведінку цих систем. Їх розробка потребує передових і глибоких знань у кількох областях, таких як інженерія програмного забезпечення, електротехніка, машинобудування та інженерія керування. Під час розробки кожна область потребує більш ніж однієї точки зору та включає навички з різних наукових і технічних галузей [1]. Тим не менш, недостатньо мати глибокі знання окремої області: нам потрібно зрозуміти як різні домени працюють разом, і що у поведінці змінюється, коли вони взаємодіють.

Нам потрібно представити ці системи, щоб зрозуміти їх поведінку, передбачити їхній вплив на навколишнє середовище, яке вони контролюють, і безпечно визначити їхню поведінку. Враховуючи критичний характер цих систем, зростаючу вимогливість до складності, суворіші правила безпеки та охорони навколишнього середовища, а також швидші цикли розробки проекту, розробку можна розподілити між різними товариствами, експертами, зацікавленими сторонами та інженерами, використовуючи однозначні представлення. Офіційні специфікації та представлення дозволяють виконувати завдання валідації та верифікації для кожної моделі та, що більш важливо, для системи в цілому.

Розробки, засновані на моделюванні (MDE), сприяють використанню предметно-орієнтованих мов моделювання для розробки складних CPS, використовуючи мови, інструменти та методології, адаптовані як синтаксично, так і семантично до сфери знань користувача [2]. Використання цих мов і інструментів полегшує роботу з моделювання, імітації, верифікації та валідації частин системи з певної точки зору. Тому використання різних мов і інструментів для розробки різних частин системи призводить до неоднорідного середовища розробки, тобто система створена з використанням різних моделей, які відповідають різним мовам. Прикладом можна вважати розробку CPS неоднорідним середовищем розробки через внутрішню неоднорідність мов і інструментів до вимог CPS для представлення різних формалізмів і доменів, які складають ці системи (тобто фізичні та кібер домени).

Зокрема, гетерогенна природа CPS призводить до використання різних формалізмів, що представляють різні частини цих систем. Наприклад, фізичні процеси визначаються за допомогою безперервного часу (CT) для кращого представлення безперервної природи фізичних механізмів у природі, де час представлено як безперервний потік, де між двома точками

часу існує нескінченна кількість інших точок. Навпаки, кіберпроцеси повинні виражати логіку та алгоритми, що використовуються в CPS, і їх краще описувати за допомогою, наприклад, моделями з дискретними подіями (DE), де їх виконання задано послідовністю викликаних подій.

Перевірка та верифікація цих систем є складними завданнями через неоднорідність формалізмів, мов та інструментів, які використовуються в розробці. Одним із можливих рішень цієї проблеми є використання спільного моделювання. У спільній симуляції або спільному моделюванні, різні слабко пов’язані та автономні модулі моделювання спільно виконуються. Блок моделювання (SU) є сутністю виконання, яка створює вихідні дані та споживає вхідні дані під час свого функціонування. Наприклад, це може бути окрема модель із розв’язувачем чи симулятором, програмне забезпечення з інтерпретатором або проксі-сервер до існуючої частини, як-от апаратне чи програмне забезпечення в циклі. Модуль моделювання може відповідати чорному ящику, який інкапсулює алгоритми, рівняння та інші роботи в модель, де відомі лише його вхідні та наступні виходи, гарантуючи, що внутрішня поведінка невидима для зовнішніх.

Підхід чорної скриньки відіграє важливу роль у тих підприємствах, які потребують співпраці з іншими компаніями. На розширеному підприємстві широке використання електронних комунікацій для обміну інформацією з постачальниками та продавцями дозволяє скоротити життєвий цикл обробки матеріалів та інформації, розробки продукту та інфраструктури, час, необхідний для виходу на ринок через зростаючу конкуренцію, і створити більш ефективний алгоритм організації та структуру системи [3]. Тоді розширене підприємство відповідає за весь життєвий цикл своєї продукції: від закупівлі матеріалів і управління ланцюгом постачання до виробництва, виробництва, розповсюдження продукції та обслуговування клієнтів. Крім того, він відповідає за процеси утилізації та переробки продуктів із вичерпаним терміном служби. Однак розробка складної системи, заснованої

на підході, заснованому на моделях, потребує спільного використання та обміну моделями для виконання інтеграції між ними для виконання завдань валідації та перевірки. Підхід «чорної скриньки» забезпечує захист інтелектуальної власності, приховуючи конфіденційні знання в нечитабельному форматі (двійковий файл).

Для підтримки спільного моделювання чорної скриньки стандарт FMI [4], на сьогоднішній день реалізований більш ніж сотнею промислових інструментів, пропонується об’єднати, у спосіб чорної скриньки, модулі моделювання за допомогою однорідного інтерфейсу, керованого часом. Крім того, стандарт HLA [5] запропонував ту саму ідею, але використовує інтерфейс, керований подіями. Базуючись на одному з цих інтерфейсів, координатор (також званий головним алгоритмом) відповідає за підтримку узгодженості часу та обмін даними між різними моделями, що виконуються. Розробка такого координатора зазвичай спирається на графік, що представляє обмін даними між різними моделями [6–9] та характер взаємодії між різними моделями [10].

Кілька робіт показали, що ані підхід, керований часом, ані чисто подіями, не може правильно обробляти всі виконання моделі [11–13]. Інші дослідження показали, що правильність спільного моделювання залежить не лише від коректність кожної виконуваної моделі, але також залежить від координатора [11, 14–16]. Тоді саме системний інженер відповідає за написання координатора, щоб спільне моделювання забезпечувало фактичну поведінку системи.

У цій роботі вважаємо алгоритм координації правильним, якщо він не створює жодних затримок і не втрачає інформацію під час зв’язку з модулем моделювання.

Отже, затримки та втрата інформації, які з’являються під час використання API, що запускається за часом, на покроково-постійних даних, вважаються неправильними. На цьому етапі слід звернути увагу на три

важливі речі. По-перше, вибірка кусково-постійного значення може мати сенс і не обов'язково створює серйозну проблему; однак це має бути зроблено навмисне, а не результатом невідповідного API. По-друге, існує в багатьох API (наприклад, стандарт FMI [4]) можливість уникнути такої затримки, як правило, шляхом відкату симуляції до попереднього стану та спроби знайти фактичну зміну значення. Це можна зробити, лише якщо симуляцію можна фактично відкотити; також це дорого з точки зору часу моделювання. Нарешті, по-третє, варто зауважити, що проблема ширша, ніж простий ілюстративний випадок. Як показано в [17], алгоритм узгодження може впливати на коректність моделювання системи.

Суть проблеми була визначена в кількох документах: будь-якому модулю моделювання неприйнятно спілкуватися лише через API, що запускається за часом або подією. У літературі пропонуються деякі підходи для розширення деяких існуючих API для вирішення конкретної проблеми. Так було, наприклад, у [18], де вони запропонували додати новий параметр до FMI, викликаного часом doStep(dt) функція. Новий параметр *nextEventTime*, заповнювач для зберігання часу, коли відбулися непередбачувані події. [11] пішов далі, запропонувавши розширити FMI API новими функціями, які симулюють, доки вхідні та вихідні порти не будуть відповідно готові до читання або просто запису. Нарешті, нові функції FMI 3.0 для гібридного спільного моделювання намагаються об'єднати такі пропозиції (див. розділ 5 версії розробки FMI 3.0∗).

Однак у всіх цих пов'язаних роботах проблема не розглядається в її загальному вигляді, і вони розглядають конкретні випадки чогось, що має бути зрозумілим. Щоб взаємодіяти з модулем моделювання ви повинні знати про його поведінкову семантику та адаптувати спосіб реалізації doStep відповідно. Як абстракція поведінкової семантики одиниці моделювання, попередні роботи пропонували зосередитися на природі входів і виходів одиниць моделювання [8,19] як, наприклад, безперервний, кусково-

неперервний або кусково-постійний.

Потім ми визначаємо три основні аспекти, які впливають на спільне моделювання:

- написання координатора стає дедалі складнішим, оскільки воно має враховувати характеристики даних, які він передає;
- характеристики моделей, поведінкова семантика використовуваної мови та їх реалізація розв'язувачем;
- топологія між різними взаємопов'язаними виконуваними моделями.

Врахуванням усіх цих аспектів часто нехтують, але було показано, що це обумовлює правильність спільного моделювання.

У цій кваліфікаційній роботі ми беремо до уваги три аспекти, визначені раніше, пропонуючи структуру для полегшення написання правильних координаторів на основі спеціального набору інформації про модуль моделювання, наприклад, часткове уявлення про синтаксис і семантику, що використовуються всередині. Тоді каркас складається з трьох основних елементів (рисунок 1.1): інтерфейс координації моделі, мова координації моделі та розподілений алгоритм спільного моделювання як середовище виконання.

Інтерфейс координації моделі натхненний роботами над мовами опису архітектури [20,21], Координаційні мови [22] і різнорідні рамки [23–26], і демонструє мінімальну кількість інформації, необхідної для визначення значущої взаємодії між різними модулями моделювання, які розроблені з використанням різних мов або інструментів.

Характеристики моделі та її розв'язувача є ключовими елементами для визначення взаємодії з іншими SU. Можливим рішенням є демонстрація їх через інтерфейс, пристосований до семантики базового блоку моделювання. На відміну від білого ящика, для координаційного інтерфейсу моделі ця пропозиція розкриває лише частковий погляд на синтаксис і семантику SU.

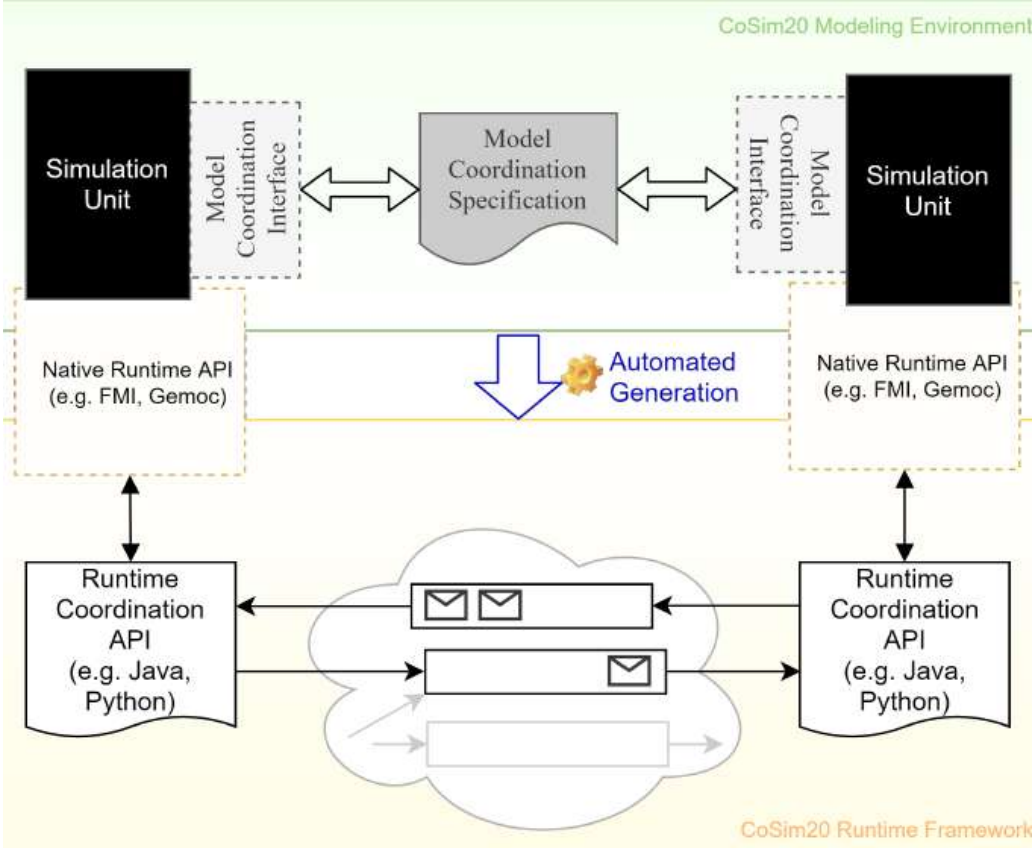

Рисунок 1.1 – Середовища моделювання та виконання

Інтерфейс розроблено спеціально для спільного використання лише елементів, необхідних для координації виконання та зв'язку між модулями моделювання. Мета полягає в тому, щоб допомогти захистити інтелектуальну власність, що міститься в SU. Ми запропонували забезпечити вищий рівень абстракції щодо семантики, щоб полегшити міркування. Ми розділили інтерфейс на три основні елементи:

- набір портів, їхні властивості, такі як ім'я, напрямок і тип, а також їхні характер даних, що представляє елемент, який демонструє часткове уявлення про семантику моделей, визначаючи її через свою поведінку;

- тимчасове посилання використовується моделлю, яка представляє часові посилання та параметри, які використовує модель.

- набір властивостей моделі, що реалізують поведінку (тобто можливість відкату або збереження стану, які дозволяють відновити попередній стан, якщо необхідно).

У контексті гетерогенних систем кожну модель можна змоделювати за допомогою різних часових посилань. Наприклад, у паливному двигуні кут розподільного вала використовується, щоб визначити, коли відкривати/закривати клапани або активувати паливні насоси. Семантика виконання цієї моделі базується не на часі, а на куті. Дії залежать від руху розподільного вала, а не від фізичного часу.

Отриманий інтерфейс надає достатньо інформації, щоб переконатися, що системний інтегратор володіє знаннями, необхідними для правильної координації різних виконуваних моделей, як досліджено в [27].

Специфікація моделі координації (MCS) чітко визначає взаємодію та правила між різними інтерфейсами координації моделі. Він складається з наборуРоз'єми. Кожен з’єднувач відповідає за визначенняколиіякодин або кілька даних передаються від моделі до іншої. Щоб визначити взаємодію, ми структурували конектор за допомогою трьох різних елементів:

- взаємодія визначає, як фактично відбувається обмін даними між портами та яке перетворення має тут відбутися, тобто вирівнювання одиниць між метрами та футами;

- умова запуску визначає момент, у який має бути взаємодія; момент можна виразити як періодичний стан (наприклад, кожні 5 мс) або як аперіодичний стан (наприклад, подія, використання внутрішньої змінної [11]);

- обмеження часу визначає відношення між часовими посиланнями моделі.

Ми перевіряємо нашу пропозицію, розробляючи інтегроване середовище розробки (IDE), яке підтримує специфікацію інтерфейсів координації моделі (MCI) і специфікації координації моделі (MCS). Було запропоновано дві предметно-специфічні мови для написання цих специфікацій, використовуючи мову, здатну виражати концепції координації та їх інтеграцію. Обидві мови можна використовувати в текстовому

середовищі, наданому Xtext, і графічному візуалізаторі, наданому за допомогою Sirius. Деякі засоби, такі як функції імпорту FMI та автозавершення, полегшують розробнику написання правильних специфікацій. Однак остаточну координаційну модель має розробити проектувальник. Після вказівки в моделі координації, тобто FMU) і експортовані моделі Gemoc. Ми пропонуємо середовище виконання, здатне координувати кілька розподілених об’єктів без необхідності централізованого координатора. Під час виконання цей фреймворк інкапсулює SU, відповідно до його MCI, спеціальною оболонкою, адаптованою до його семантики. Обгортка містить параметризовану семантику API для керування виконанням моделі та комунікаційний модуль для взаємодії з рештою системи.

1.2 Дослідження проблем спільного моделювання

У цьому розділі ми проілюструємо основні проблеми, які розглядаються в атестаційній роботі. Наша головна мета — розробити інструмент спільного моделювання, який можна безперешкодно використовувати на різних етапах розробки для підтримки процесу проектування різними зацікавленими сторонами. Далі представлені деякі з основних проблем.

Розширений корпоративний підхід дає можливість спільно використовувати виконувані моделі між кількома підприємствами. Підхід «чорної скриньки» для захисту ІВ є одним із найважливіших аспектів цього процесу, і його збереження є обов’язковим.

Поточні стандарти та інструменти пропонують незначну підтримку для забезпечення правильної спільної симуляції на основі підходу чорної скриньки. Він забезпечує незначний або повний контроль над внутрішньою

конфігурацією та виконанням моделі та її симулятора.

Різні мови та інструменти, що використовуються для розробки системи, призводять до неоднорідності системи з точки зору представлення та обробки часу в моделі та в глобальній системі. Часові моделі можуть відповідати різному формалізму, який представляє час за допомогою різних представлень (наприклад, Ньютонівський, дискретний або аналоговий [28]). Взаємозв'язки між цими уявленнями визначаються їхнім зв'язком із фізичним або настінним часом. Однак через неоднорідність систем інші представлення часу повинні мати можливість інтегруватися в систему. Наприклад, модельний час – це представлення часу, де час плине нерівномірно та незалежно від будь-яких посилань на фізичний час. У цій роботі ми розглядаємо цю проблему як часову синхронізацію між різнорідними моделями.

Системні архітектори та системні дизайнери можуть використовувати спільне моделювання як метод виявлення загальної поведінки системи під час спільного моделювання. Його доступ повинен бути прозорим для них і легко встановлюватися. Однак фактичні підходи, необхідні для налаштування та налаштування алгоритму, використовуваного для координації виконання спільного моделювання. Крім того, специфікація алгоритму написані мовою загального призначення (Java, C++, Python), обмежують вибір фреймворків спільного моделювання відповідно до мови, відомої системним архітекторам/дизайнеру. Це дозволяє використовувати спільне моделювання на всіх етапах розробки, від ранньої фази розробки, де оцінюються кілька рішень, до верифікації та підтвердження остаточного рішення.

Представлені виклики будуть проілюстровані та широко обговорені та розглянуті далі. Однак ця робота передбачає певні обмеження на системи, з якими ми маємо справу, та їх склад, щоб зосередити наше дослідження на інших проблемах.

1.3 Встановлення обмежень спільного моделювання

У цьому розділі ми представляємо припущення та обмеження для цієї роботи. У розширеному корпоративному контексті ми припускаємо, що постачальники хочуть захистити свої моделі за допомогою найсучасніших підходів. Спільне використання моделі з симулятором дозволяє уникнути розкриття внутрішніх обчислень і поведінки, які можуть бути зафіксовані зовнішнім симулятором. Зауважте, що якщо постачальники не проти поділитися моделлю білого ящика, тоді наш підхід може отримати користь від додаткової інформації про її внутрішню семантику та про керування симулятором.

Інтеграція моделі може відбуватися за допомогою підходу, обраного для обміну моделями: підхід білої скриньки ділиться внутрішніми механізмами та вихідним кодом, надаючи чіткий доступ до інтелектуальної власності; підхід чорної скриньки поділяє об’єкт лише з відкритими входами та виходами. Ми вважаємо, що спільне використання моделі чорної скриньки може призвести до кращого захисту IP, враховуючи технічну перешкоду для зворотного проектування внутрішніх алгоритмів. Однак цей підхід призводить до проблем з інтеграцією моделі, присутність яких відсутня або обмежена в підході білого ящика.

Хоча одну мову можна використовувати для розробки різних моделей, вона не завжди є більш прийнятним вибором. Неоднорідність реальності призводить до наявності різних точок зору та поведінки для представлення. Наприклад, мови, призначені для представлення фізичних процесів, такі як Modelica, не можуть використовуватися для представлення логічних процесів, таких як поведінка алгоритмів, які використовуються в розробці програмного забезпечення.

З промислової точки зору, підхід чорної скриньки забезпечує захист інтелектуальної власності внутрішньої роботи моделі та знань, які

використовуються для їх розробки.

Як припущення, ми вирішили зосередити нашу роботу на координаційному аспекті спільного моделювання, ми припускаємо, що моделі, з якими ми працюємо, правильні та дійсні.

Останнім часом зростаючий інтерес до методів спільного моделювання постійно змінює стан техніки. Важливість цієї технології спонукала кілька компаній і дослідницьких організацій докласти значних зусиль у її дослідження та розробку: отже, сучасний рівень і зрілість деяких представлених технологій можуть відрізнятися протягом останнього періоду.

2 РОЗРОБКА МОДЕЛІ ПРОЦЕСУ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ СКЛАДНИХ РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ

2.1 Управління процесом моделювання

Складна системна інженерія вимагає інтеграції кількох різних формалізмів, інструментів і стандартів. Внутрішня складність реального світу призводить до використання абстракцій і мов, присвячених різним сферам, які явно демонструють складну поведінку. Інструменти та мови використовуються для вираження цієї поведінки та міркування про можливі рішення. Гетерогенність цих мов і інструментів призводить до складностей, які ми знаходимо в реальному світі, а також у світі інженерії. У спільноті архітектури програмного забезпечення мови опису архітектури (ADL) було запропоновано вирішити цю проблему: вони надали доменно-спеціальні мови (DSL) для організації та інтеграції різних моделей. Як правило, мова опису архітектури (ADL) керує SU як компонентом, інкапсульованим в однорідний інтерфейс, який представляє певну відповідну частину моделі як набір вхідних/вихідних даних, які дозволяють обмінюватися даними з моделлю. Потім, на основі цього, вони визначають зв’язки між моделями. Варто зауважити, що ADL мови керування (CL) в основному зосереджено на інтеграції компонентів програмного забезпечення компонентів кіберфізичної системи (CPS); однак вони запропонували концепції, які були використані в цій атестаційній роботі.

У спільноті симуляторів спільне моделювання зосереджується на керованої взаємодії між різними модулями моделювання, які представляють різні частини однієї системи, щоб краще зрозуміти нову поведінку системи. У цьому контексті модуль моделювання – це, як правило, чорний ящик, сутність виконання, яка може, наприклад, інкапсулювати модель та її

розв’язувач, двійковий виконуваний процес або проксі-сервер апаратного пристрою. Тоді оркестровка має першочергове значення, оскільки вона визначає моменти, коли модуль моделювання обмінюється даними з іншими блоками моделювання. Існує кілька моделей координації, які мають різну семантику. Наприклад, найпопулярніші моделі координації засновані на семантиці запуску часу та події. Крім того, модель координації може бути розподілена між різними ядрами, ЦП, пристроями, або мережевої інфраструктурою. Усі ці властивості та неоднорідність семантики впливають на точність і загальну продуктивність спільного моделювання, як ми обговоримо далі.

У наступному розділі ми представляємо основну семантику координації, яка використовується для координації спільного моделювання, таку як семантика безперервного часу та дискретної події. Після цього ми проаналізуємо предметно-орієнтовані мови для спільного моделювання, запропоновані для опису спільної архітектури та взаємодій, які мають відбуватися між її компонентами. Далі ми аналізуємо координаційні інтерфейси, які використовують представлені мови. Наприкінці ми представляємо підходи до спільного моделювання, підкреслюючи їх можливості для розподілу виконання спільного моделювання. Централізований і розподілений підходи представлені на основі підтримуваної топології.

2.2 Модель процесу управління спільним моделюванням

У цьому розділі ми проілюструємо моделі, які використовуються для координації виконання спільного моделювання. Терміни моделювання і спільне моделювання можна заплутатися під час читання: для зрозумілості вживаємо термін моделювання для позначення незалежного процесу, який

виконує модель та її розв’язувач, програмне забезпечення та його інтерпретатор або виконуваний файл. Вважається, що ці об’єкти виконуються в рамках процесу або потоку, і вони не призначені для роботи в кооперативному та спільному контексті. Отже, ми використовуємо термін спільне моделювання для позначення спільного моделювання між двома або більше моделюваннями. Зокрема, ми виконуємо спільну симуляцію за допомогою блоку симуляції чорної скриньки: внутрішні механізми та алгоритми приховані за інтерфейсом, який надає API для зовнішнього керування, не розкриваючи його внутрішньої IP-адреси.

У цьому розділі ми аналізуємо базові алгоритми, які використовуються для фактичного запуску спільного моделювання, і їхні наслідки для виконання. У літературі алгоритм, який координує виконання набору моделей, називається декількома способами: головний алгоритм і програма управління [12], інтерфейс функціонального макету (FMI)[4], координатор в CLs[74], або режисер у Птолемея II [14]. У цій роботі ми посилаємося на реалізацію (тобто вихідний код) семантики виконуваного файлу, що використовується для розробки керування як алгоритм узгодження. Варто зауважити, що алгоритм координації може бути розподіленим або централізованим: FMI спільнота запропонувала в основному централізовані майстер-алгоритми, тим часом координаційні мови зосереджені на розподіленій координації [75,76].

2.3 Спільне моделювання на основі безперервного часу

У кіберфізичних системах фізичні компоненти зазвичай розробляються та моделюються за допомогою диференціальних рівнянь або мови на їх основі. Наприклад, мікро електромеханічні системи (MEMS), механічні частини та аналогові схеми можна описати за допомогою диференціальних рівнянь; потім ці рівняння можна визначити за допомогою

мови Modelica [77], що підтримує визначення систем диференціальних рівнянь. У цих компонентах вхідні та вихідні сигнали є переважно сигналами безперервного часу. Диференціальні рівняння, які використовуються для моделювання динаміки безперервного часу, можна розділити на кілька типів: розрізнення ґрунтується на тому факті, що рівняння бувають звичайними чи частковими, однорідними чи неоднорідними, лінійними чи нелінійними. Наприклад, неповний список включає звичайні диференціальні рівняння (ODE), диференціальні алгебраїчні рівняння (DAE) і часткові диференціальні рівняння (PDE).

При симуляціях, зокрема, ми зосереджуємося на звичайних диференціальних рівняннях (ОДР), які в цій роботі є диференціальними рівняннями за змінною часу. Загальним підходом до чисельного розв'язування диференціальних рівнянь є використання чисельного інтегрування. Основна задача полягає в обчисленні наближеного розв'язку певного інтеграла із заданим ступенем точності. У цьому розділі ми даємо короткий огляд чисельного інтегрування та розв'язувачів на основі [78] визначення.

Ми представляємо просту модель із двома змінними безперервного часу, ω и x, як описовий приклад. Припускаємо, що τ є інтегрованою функцією виду

$$x(t) = x_0 + \int_0^t \omega(\tau)d\tau \qquad (2.1)$$

де x_0 є константою, $x(t)$ представляє площу під кривою, утвореною $\omega(\tau)$ в інтервалі від τ=0 до τ=t, перекладений на початкове значення x_0. Тому ми можемо обчислити x, надаючи як вхідні дані для числового інтегратора з початковим станом x_0.

Основна задача числового інтегрування полягає в обчисленні

наближеного розв’язку певного інтеграла із заданим ступенем точності. Ступінь точності залежить від застосування: одним із критеріїв визначення необхідної точності є те, що обчислене значення для x має бути достатньо точним у достатній кількості точок, щоб ці значення можна було використовувати для розрахунку майбутніх значень x вчасно $t \in T$.

Реалізація алгоритму чисельного інтегрування виконується на зрозв'язувачі. Ми представляємо розв’язувач із фіксованим кроком на основі числового підходу, який називається метод Ейлера. Це найпростіший із явних чисельних методів розв’язування диференціальних рівнянь. Постійний розмір кроку h фіксує наближення x_{n+1} до $x(t_{n+1} = (n+1)h)$ явно обчислюється як

$$x_{n+1} = x_n + h\tau(x_n) \tag{2.2}$$

Починаючи з початкового значення $x_0 = x(0)$, метод обчислює послідовність наближень $x_1, x_2, \ldots, x_n$ до рішення, використовуючи одну оцінку f за крок [79]. Цей метод є менш точним, і помилки накопичуються швидше, але розв’язувач не вимагає знати вхідні дані на час n+1 [78]. Якщо потрібна висока точність, менший розмір кроку h можна використовувати, але, як недолік, це збільшує кількість обчислень.

Метод Ейлера можна узагальнити для регулювання розміру кроку відповідно до того, наскільки швидко змінюється сигнал. Такі розв’язувачі використовують певний алгоритм для оцінки більш відповідного розміру кроку для оцінки моделі в кожен момент часу. По-перше, необхідно вибрати допуск на основі необхідного ступеня точності. Ідея полягає в тому, що алгоритм фіксує розмір кроку h, а потім обчислює перше наближення x_{n+1}розрахункова похибка ε. Якщо помилка перевищує допуск, тоді алгоритм повинен повторно обчислити наближення n+1 використовуючи менший розмір кроку h. Розмір змінного кроку розв'язувача методом Ейлера

потім визначить приріст часу h_n до $t_n=t_{n-1}+ h_n$, потім обчислює

$$x(t_n) = x(t_{n-1}) + h\tau(t_{n-1}) \quad (2.3)$$

Розмір змінного кроку розв'язувача методом Ейлера в такому випадку є окремим випадком широко використовуваних методів Рунге-Кутта (РК) [78,80] у спільному моделюванні.

2.4 Спільне моделювання з безперервним часом

Мета спільного моделювання полягає в наближенні поведінки пов’язаних SU з певним ступенем точності. У цьому контексті, коли кілька моделей мають свій внутрішній розмір кроку (інтервал часу) та не залежать від інших, необхідно визначити розмір макрокроку. Потім визначається розмір кроку зв’язку (тобто значення частоти дискретизації), при якому динамічні моделі готові для обміну даними. Зазвичай він більше або дорівнює всім розмірам кроку, визначеним у кожній моделі. Щоб організувати загальне спільне моделювання, необхідно створити керуючу програму - Майстер-алгоритм (МА). Вона відповідає за визначення, в якому порядку моделям надаються вхідні та вихідні дані та обчислюється наступний інтервал кроку часу. Два широко використовувані методи реалізації МА: Гаусса-Зейделя і Якобі [81].

Аналгебраїчна петля може виникнути, коли вхідний порт із прямим проходженням керується виходом того самого модуля моделювання, або безпосередньо (рисунок 2.1), або шляхом зворотного зв’язку через інші блоки моделювання, які мають пряме проходження (рисунок 2.2).

Якщо виявлено алгебраїчний цикл, є два можливих рішення [15]: виконати алгоритм фіксованої точки [82,83] або додати третій компонент, який затримує один із двох вхідних даних [15].

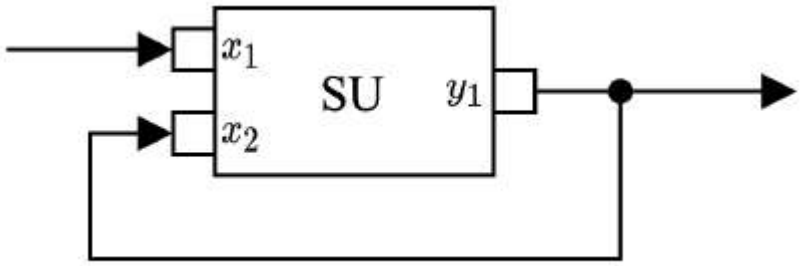

Рисунок 2.1 – Цикл на тієї самої програмної моделі

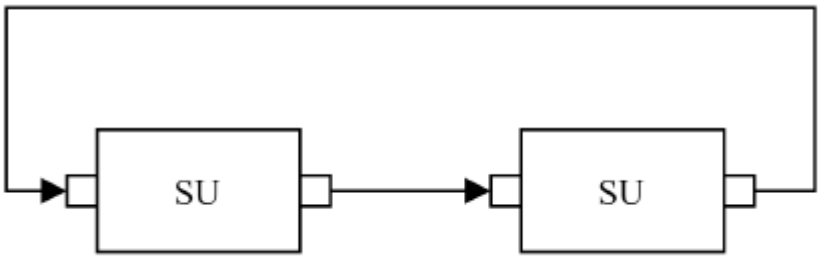

Рисунок 2.2 – Цикл між двома програмними моделями

Однак у багатьох випадках вирішення алгебраїчного циклу не завжди можливо через відсутність підтримки відкату, а додавання третього компонента із залежностями введення/виведення може бути неможливим рішенням. Розв'язок алгебраїчних циклів є загальновідомою задачею [9,15,83] і це виходить за рамки цієї атестаційної роботи. Однак, розв’язувачі алгебраїчних циклів, що підлягають вирішенню, засновані на підході Ньютона-Рафсона, мають як вимогу гладкі функції та несумісні з дискретними змінами [84]. З цієї причини алгебраїчні цикли повинні бути відхилені структурою, і розробнику системи буде надіслано попередження. Крім того, наступні алгоритми не враховуватимуть алгебраїчні цикли.

В методі Гаусса-Зейделя, МА просить послідовно кожну модель обчислити наступний інтервал кроку в часі та отримує результати. Потім, МА надає останні результати для наступної моделі (відповідно до їх топології). Послідовний характер алгоритму вимагає встановлення порядку серед моделей. Загальний порядок можна знайти за допомогою підходів,

запропонованих у [85]. Можлива реалізація Гаусса-Зейделя метод задається [85], який відтворюється в алгоритмі, що представлено у Лістингу 2.1. Наведено псевдокод для реалізації методу Гаусса-Зейделя координації системи спільного моделювання.

Листинг 2.1 – Синхронізація спільного моделювання методом Гаусса-Зейделя

```
foreach  su in SU do {
     uc[su] = y[su] =0;
     ip[su] = 0;
};
t = 0;
foreach  su in SU do {
     uc[su] = C[su]({y[su]});
     y[su] = getOutput(su,uc[su]);
     up[su]=uc[su];
};
while(t<=T){
    foreach  su in SU do {
     uc[su] = C[su]({y[su]});
     doStep(su,H,uc[su],up[su])
     y[su] = getOutput(su,uc[su]);
    }
    foreach  su in SU do {
     up[su] = uc[su]
    }
    t=t+H;
}
```

Список моделей є вектором модулів моделювання. Алгоритм приймає як вхідні дані час зупинки моделювання T, розмір кроку зв’язку H та впорядкований вектор програмних моделей SU, обчислює вхідні дані для модуля моделювання. Метод C[su]() отримує вихідні значення для модуля моделювання su. Метод doStep просіть виконати обчислювальний крок розміром H для кожного модуля моделювання su. Метод Якобі в МА викликає кожну модель одночасно обчислити свій інтервал. Потім, наприкінці їхнього кроку, МАвстановлює входи програмної моделі (відповідно до їх топології). Список моделей SU не потребує впорядкування через те, що головний алгоритм не нав’язує жодного порядку виконання. Можлива реалізаці методу Якобі задається [85] алгоритмом, який відтворєно

у лістингу 2.2. Зазвичай, алгоритм Якобі менш точний, ніж алгоритм Гаусса-Зейделя через те, що моделі не можуть використовувати методи інтерполяції [85]. Однак можна скористатися перевагами паралелізму для виконання системи розподіленим способом.

Листинг 2.2 – Синхронізація спільного моделювання методом Якобі

```
foreach  su in SU do {
     uc[su] = y[su] =0;
};
t = 0;
while(t<=T){
    foreach  su in SU do {
     uc[su] = C[su]({y[su]});
     y[su] = getOutput(su,uc[su]);
     up[su]=uc[su]
    }
    foreach  su in SU do {
     doStep(su,H,uc[su],up[su])
    }
    t=t+H;
}
```

Коли метод, заснований на безперервному часі, використовується для спільного моделювання CPS, виникає проблема інтеграції, яка визначається місцем розташування подій у часі. Майстер-алгоритми синхронізації часу, представлені в літературі та зазвичай використовувані, не підтримують імітаційні модулі моделювання, які представляють розрив або запускають дискретну подію. Завдяки широкому впровадженню стандарту FMI [4], який поширює використання спільного моделювання для CPS, декілька робіт розглядали проблему визначення місця події, пропонуючи різні методики, в основному засновані на механізмі крокової відмови та відкату.

Модуль моделювання може відхилити запропонований розмір кроку зв’язку: якщо запропонований розмір перевищує мінімально підтримуваний, тоді SU може відхилити його [86]. Потім цей механізм використовується також у випадку, якщо під час внутрішнього моделювання SU виникає розрив. Обробка розриву залежить від можливостей і характеру модулів

моделювання. У випадку, якщо модуль моделювання інкапсулює об’єкт на основі CT, розрив слід усунути шляхом повторної ініціалізації SU [87]. Однак основним недоліком цього підходу є те, що він може викликати каскад повторних ініціалізацій у системі, погіршуючи загальну продуктивність і збільшуючи час моделювання [88,89]. Методика повторної ініціалізації SU складається з відкатом моделювання SU до попереднього стану, в якому розрив ще не відбувся. Він вимагає від SU зберігати свій стан і відновлювати його, коли це необхідно. Однак механізм відкату може підтримуватися не всіма модулями моделювання.

У модулі моделювання на основі КТ місце події пов’язане з детектором переходу через нуль, який запускає подію, коли безперервний сигнал перетинає визначений поріг. Крім того, ці події зазвичай вважаються рідкісними. Однак у модулі симуляції на основі дискретних подій події є важливим елементом семантики. Методи, запропоновані для локалізації подій, не підходять через велику кількість необхідних відкатів: зокрема, усі запропоновані роботи з мінімізації необхідних відкатів марні, оскільки відкати є систематичними.

У контексті FMI було запропоновано кілька головних алгоритмів для обробки гетерогенних систем, таких як CPS. Більшість з них є варіантами відомих методів Якобі або Гаусса-Зейделя і присвячені системі диференціальних рівнянь (безперервного часу) [6,7,9,15,90–92]. Двома можливими рішеннями для приблизного місцезнаходження моменту виявлення події є обчислення часового кроку просування часу з використанням мінімального розміру кроку, прийнятого SU [93] або використовувати двосекційний метод [94,95]. Головним недоліком є те, що всі вони потребують механізму відкату, що знижує загальну продуктивність спільного моделювання.

Широко поширеним стандартом для спільного моделювання є стандарт інтерфейсу функціонального макету (FMI) [4]. Це незалежний від

інструментів стандарт, який пропонує об’єднати модуль моделювання за однорідним API, керованим часом. Головний алгоритм не є частиною стандарту, і його необхідно реалізувати. Це може встановити або отримати поточне значення відкритої змінної (відповідно до її напрямку) за допомогою стандартизованого FMI API. Цей API також використовується для виконання програмної моделі для певного інтервалу часу, зазначеного в методі doStep. У режимі спільного моделювання кожен розв’язувач FMU вирішує, скільки обчислювальних кроків слід виконати за цей інтервал часу, щоб досягти бажаної точності.

Ці стандарти та інструменти спочатку були розроблені для безперервного спільного моделювання. Фактично, у стандарті FMI ми знаходимо багато функцій і властивостей для покращення результатів спільного моделювання та продуктивності на основі можливостей чисельного вирішувача (напрзмінні стану та похідні сигналів). Зі збільшенням використання спільного моделювання для CPS кілька робіт вводять пропозиції щодо підтримки іншого формалізму, відмінного від КТ, наприклад модулів моделювання на основі дискретних подій. Наприклад, [11,19,28] пропонуємо оновити поточний FMI API, щоб нативно підтримувати різні формалізми, такі як Discrete-Event. Однак ці зміни порушують сумісність із інструментами спільного моделювання, які підтримують FMI 1.0 і FMI 2.0, і вони повинні бути прийняті спільнотою FMI, щоб стати частиною стандарту.

2.5 Спільне моделювання на основі дискретних подій

Моделі безперервного часу підходять для розробки та аналізу фізичних моделей, але вони не дуже підходять для проектування, моделювання та аналізу великої колекції моделей або складних систем через їх розмір. Такі системи, як мережі, програмне забезпечення, центральний процесор і дизайн

апаратного забезпечення, представляють деякі системи, які неможливо виразити за допомогою систем диференційних рівнянь. Зокрема, такі проблеми, як синхронізація, взаємне виключення, паралелізм і планування не можна визначити та дослідити за допомогою диференціальних рівнянь. У цьому розділі ми спочатку представляємо симуляцію дискретної події (DE), а потім описуємо підходи спільного моделювання, які використовують семантику дискретної події.

Час блоку моделювання, який розвивається всередині моделі, може не синхронізуватися з реальним часом, або з годинниковим час, тобто часом, який минає в реальному світі, який моделюється. Модельний час може рухатися швидше або повільніше, ніж час настінного годинника. У деяких контекстах, таких як симуляція Hardware-In-the-Loop, керована алгоритмами DE, алгоритм повинен враховувати затримку, введену симуляцією: якщо комп'ютер моделює систему досить швидко, симуляція повинна «чекати» реального світу час для синхронізації виконання. Крім того, якщо час моделювання занадто повільний порівняно знастінний годинник, тоді неможливо виконати симуляцію апаратного забезпечення в циклі. У моделюванні дискретних подій час просувається не через імітацію настінного годиннику (тобто представлений у вигляді континууму), але змінами станів системи в дискретні моменти часу.

Точніше, симуляція дискретних подій (DES) описує поведінку системи, визначаючи, як внутрішній стан розвивається відповідно до дискретної послідовності подій. Подія — це миттєва подія, яка змінює стан системи в певний момент часу, який називається міткою часу. Мітка часу має область застосування, обмежену модулем моделювання. Подія може створювати інші події з тією самою або більшою міткою часу. На відміну від моделювання безперервного часу, модулю моделювання дискретних подій не дозволяється змінювати свій стан між двома послідовними подіями. Потім моделювання продовжується шляхом збільшення часу та відповідного оновлення стану

моделі (або за необхідності).

На рисунку 2.3 проілюстровано основні поняття алгоритму моделювання стандартного симулятора DE.

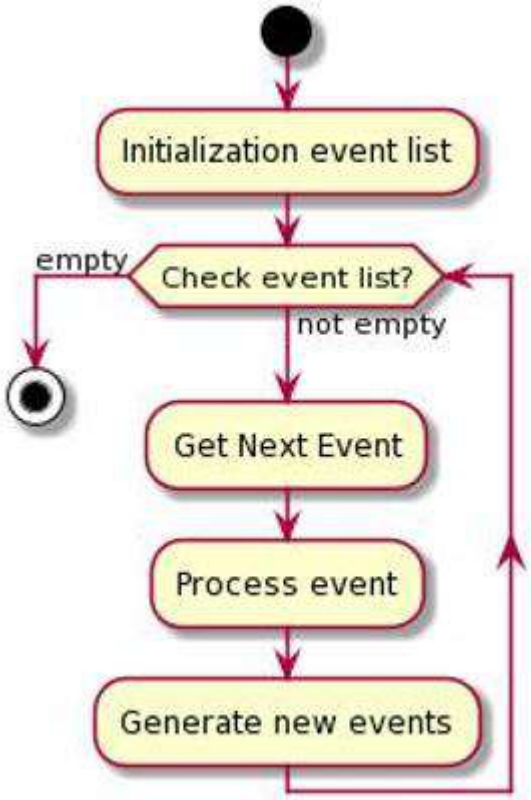

Рисунок 2.3 – Діаграма діяльності системи з дискретними подіями

Основний механізм для збільшення часу моделювання та забезпечення того, щоб події відбувалися в правильному хронологічному порядку, базується на ідеї списку подій, який називається "Список майбутніх подій" (FEL). FEL містить усі сповіщення про всі події, які мають відбутися в майбутньому. Хронологічний порядок передбачає, що всі події повинні задовольняти $t \leq t_1 \leq t_2 \leq .. .\leq t_n$, $n \in \mathbb{N}$ - поточний час моделювання. Планування майбутніх подій передбачає, що дія перемістить у списку подій наступну пов’язану подію. А глобальний годинник потім можна використовувати для синхронізації подій у системі.

У циклі алгоритм симуляції потім бере першу подію зі списку,

встановлює поточний час симуляції на час у події, витягує її зі списку та виконує набір дій, пов’язаних із подією. Андіюпотім може генерувати нові події, які буде вставлено в список подій. Відповідні позначки часу мають дорівнювати або перевищувати поточний час симуляції. Коли список подій порожній або досягнуто кінця часу симуляції, симуляція припиняється.

Якщо подія з більшою міткою часу була виконана перед меншою, яка планувала б подію з більшою міткою часу, виникла б логічна помилка. Ці типи помилок називаютьсяпомилки причинності. В одній симуляції DE управління часом забезпечується локальним годинником, до якого кожна подія посилається для своєї позначки часу. У системі, де виконується кілька паралельних симуляцій DE, локальний годинник не гарантує узгоджену позначку часу для всіх подій.

Спільне моделювання DE — це правильна симуляція моделювання на основі DE у спільній формі. У моделюванні паралельних дискретних подій (PDES), яке також називається розподіленим моделюванням [96], незалежна симуляція дискретних подій виконується на паралельній машині (тобто комп'ютері). Основне занепокоєння полягає в синхронізації часу між компонентами системи: коли кілька частин системи DE виконуються паралельно, вони можуть запускати події епізодично або з різною швидкістю. У PDES можливим рішенням є використання глобального симуляційного годинника, спільного для комп’ютерів. А замковий крок виконання гарантує, що на кожній гілкі модельованого часу перевіряється кожен список подій на комп’ютерах і виконуються події, які мають відбутися. Однак загальне виконання може постраждати від низької продуктивності через те, що дві події рідко мають однакову точну позначку часу [96]. Насправді, якщо жодна подія не відбувається в ту саму позначку часу, симулятор повинен зупинятися при кожному виникненні події, перевіряючи список подій і виконуючи заплановані події. Цей процес має виконуватися послідовно, щоб запобігти помилкам причинності.

Можливим рішенням для прискорення симуляції є дозволити одночасне виконання подій, які відбуваються в інший симульований час. Це дозволяє моделювати паралельні події різних комп'ютерів. Зазвичай симуляція виконується як фізичний процес на одному процесорі: типовим підходом є зіставлення фізичного процесу з логічним процесом (LP), і кожен LP може моделювати незалежно.

У цьому паралельному змаганні виконання, щоб запобігти помилки причинності, ми визначаємо обмеження локальної причинності: "Симуляція дискретної події, що складається з логічних процесів (LP), які взаємодіють виключно шляхом обміну повідомленнями з мітками часу, підкоряється обмеженню локальної причинності тоді і тільки тоді, коли кожен LP обробляє події в порядку неубування часових міток." [97]. Це забезпечує правильність моделювання без внесення помилок, викликаних самим моделюванням (тобто уникаючи помилок причинності). Обмеження причинності є достатнім, але не завжди необхідним, щоб гарантувати відсутність помилок причинності [97]. Одним із головних завдань у PDEVS є одночасне виконання логічних процесів, що забезпечує правильні результати моделювання. Цієї мети можна досягти шляхом надання формального алгоритму для синхронізації виконання кожного логічного процесу з рештою системи.

Синхронізація відноситься до правильного виконання компонента DE в розподіленій системі, гарантуючи, що повторні експерименти виконання з тим самим набором вхідних даних дають однакові результати [97]. Алгоритми управління часом можна розділити на дві основні категорії: консервативну і оптимістичну синхронізації. Ці підходи були розроблені для виконання розподілених або паралельних блоків моделювання в розподіленій мережі або багатопроцесорних платформах.

Консервативний підхід суворо запобігає ймовірності будь-якої помилки причинно-наслідкового зв’язку, визначаючи, чи безпечно обробляти

подію. Перший алгоритм був розроблений [98]: у мережі, яка гарантує, що повідомлення надходять у тому самому порядку, в якому вони були надіслані, кожен LP може надсилати свою мітку часу, яка не зменшується, у повідомленні. Потім повідомлення організовуються в чергу «першим прийшов – першим вийшов» (FIFO) і виконуються. Потім події плануються відповідно до черги FIFO. Локальна симуляція починає споживати подію з найменшою міткою часу: потім локальні події плануються в межах LP у спеціальному списку подій. Коли черга повідомлень (на кожному LP) стає порожньою, тоді LP зупиняється та перебуває в тупику. Щоб цього уникнути, використовуються нуль-події та повідомлення, на їх основі. Нуль-подія не може створювати нові події або оновлювати внутрішній стан системи. Нуль-повідомлення є ключовим елементом концепції: якщо LP знаходиться під час моделювання T і це гарантує, що вихідні повідомлення в майбутньому матимуть мітку часу щонайменше значення T+L, де L є періодом очікування. Однак такий підхід призводить до великої кількості нуль-повідомлень і високих обчислювальних витрат [97].

Оптимістичний підхід використовує таку стратегію: помилки причинно-наслідкового зв’язку допускаються, але коли вони виявляються, виникає механізм відновлення (тобто відкат). Оптимістичний підхід має дві основні особливості: високий рівень паралелізму та синхронізацію, більш прозору для програми, ніж консервативний підхід. Однак через механізм відкату в разі порушення обмеження причинності це може призвести до каскаду відкатів, що вплине на загальну продуктивність. Один із широко використовуваних алгоритмів під назвою Time Warp (TW) [99]: він дозволяє кожному LP незалежно збільшувати свій власний час моделювання, і, коли виявляється порушення обмеження причинності, Time Warp виконує відкат, відновлюючи стан, попередній до порушення, і повторно обчислює події, які порушили часове обмеження [97]. Основна проблема цього підходу полягає в тому, що операції введення/виведення не завжди можна скасувати за

допомогою відкату. Рішенням цієї проблеми є використання глобального віртуального часу [100], що визначає нижню межу мітки часу для кожного майбутнього відкоту, дозволяючи відкидати дані, записані до глобального віртуального часу, і збережені стани, на які це впливає.

Головним недоліком оптимістичного підходу є здожчення обчислень через зберігання станів та відкати [100]. У контексті спільного моделювання CPS блоки моделювання не завжди можуть забезпечити механізм відкату, і це впливає на зручність використання цього підходу в сценаріях, де є такі SU. Навпаки, консервативний підхід не вимагає механізму відкату за рахунокдивитися вперед механізм, який заважає SU йти вперед у часі, якщо він залежить від інших SU.

2.6 Гібридне спільне моделювання

Проектування кіберфізичних систем вимагає враховувати різні взаємодії між середовищем (наприклад, завод) і програмним забезпеченням (наприклад, кіберконтролера). Різна поведінка демонструється неоднорідністю компонентів: кібер (або цифровий) контролер можна представити за допомогою формалізму DE, тим часом плату керування двигуном можна описати за допомогою звичайних диференціальних рівнянь. Однак складність усієї системи, як сукупності установки, контролерів, датчиків, виконавчих механізмів і програмного забезпечення, така, що важко і непрактично її деталізувати [112].

Системи, в яких поєднуються дискретно-подієві та безперервні системи, називаються гібридними системами. Це поняття традиційно використовується спільнотою контролю для позначення дискретної та безперервної динаміки. Зокрема, гібридна система має безперервну еволюцію в часі та кардинальні зміни. Зміни відповідають змінам стану в автоматі у відповідь на зовнішні події [113]. Гібридні моделі системичасто описуються з

використанням спеціальних формалізмів [112] і мовами, такими як Zelus [114], який поєднує поведінку дискретних подій і безперервного часу, описуючи гібридні системи за допомогою однієї мови, для вираження фізичних моделей у вигляді звичайних диференціальних рівнянь (ODE) і кібермоделей як формалізму потоку даних. У контексті спільного моделювання представлення моделей за допомогою однієї мови запобігає неоднорідності мов та інструментів, що використовуються для розробки CPS, але підхід «білої скриньки» змушує нас зрештою розкривати IP моделей. З іншого боку, спільне моделювання за допомогою різних мов програмування (моделювання) викликає неоднорідності мов та інструментів, що використовуються для розробки CPS, і можливий підхід «чорної скриньки», що змушує розкривати IP моделі.

Наша гіпотеза щодо спільного моделювання полягає у використанні підходу чорної скриньки запобігти Розкриття IP, що робить ці мови непридатними для спільного моделювання.

Інтеграція гібридної моделі. Дедалі більше спільна симуляція, заснована на CT і DE, наближається до запропонованих механізмів і методів для інтеграції іншого формалізму. Ми можемо розділити підходи на дві основні категорії:

- спільне моделювання на основі безперервного часу з усіма SU, інкапсульованими як блок моделювання КТ (тобто SU на основі DE використовуватиме інтерфейс на основі CT для зв’язку з головним алгоритмом);

- спільне моделювання на основі DE з усіма SU, інкапсульованими як блоки моделювання DE.

Крім того, на основі основних концепцій спільного моделювання (основний алгоритм, інтерфейс і блок моделювання) можна далі розділити підходи на основі їх пропозиції та відповідного рівня (рисунок 2.4).

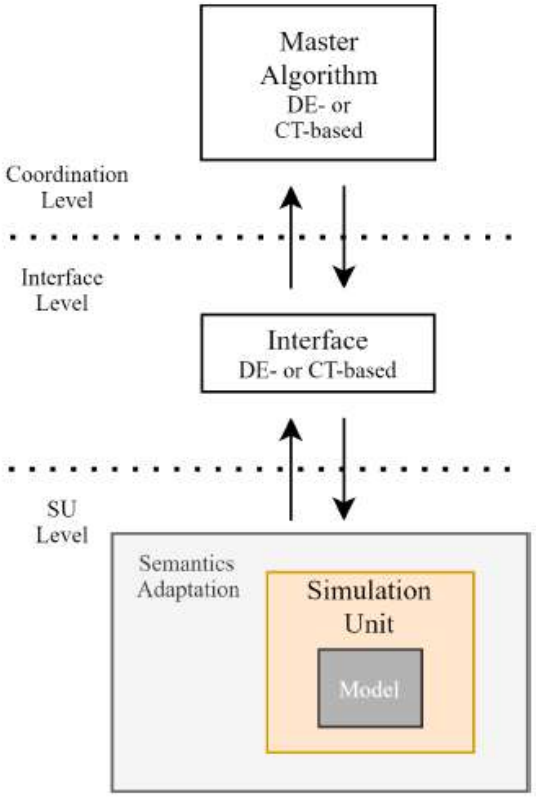

Рисунок 2.4 – Рівні спільного моделювання

У першій категорії кожен SU повинен відповідати інтерфейсу спільного моделювання на основі ДУ і, отже, семантиці головного алгоритму. Кілька підходів пропонують методи та механізми для інтеграції SU на основі DE. Наприклад, [12] запропонував інкапсулювати кожну модель, яка не відповідає формалізму CT, у SU на основі CT, забезпечуючи семантичну адаптацію між внутрішньою семантикою та семантикою CT.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СУМІСНОГО МОДЕЛЮВАННЯ

3.1 Реалізація моделей процесу імітаційного моделювання складних розподілених програмних систем

Будь-який інтерфейс координації моделі має однакову структуру: забезпечує мінімальний набір елементів, необхідних для спільного моделювання модулів. З цієї причини розроблено декларативний DSL, щоб формалізувати специфікацію інтерфейсу координації моделі, використовуючи раніше введені концепції. Цей DSL називається мовою інтерфейсу координації моделі (MCILang).

Абстрактний синтаксис MCILang визначається за допомогою метамоделі Ecore (рисунок 3.1).

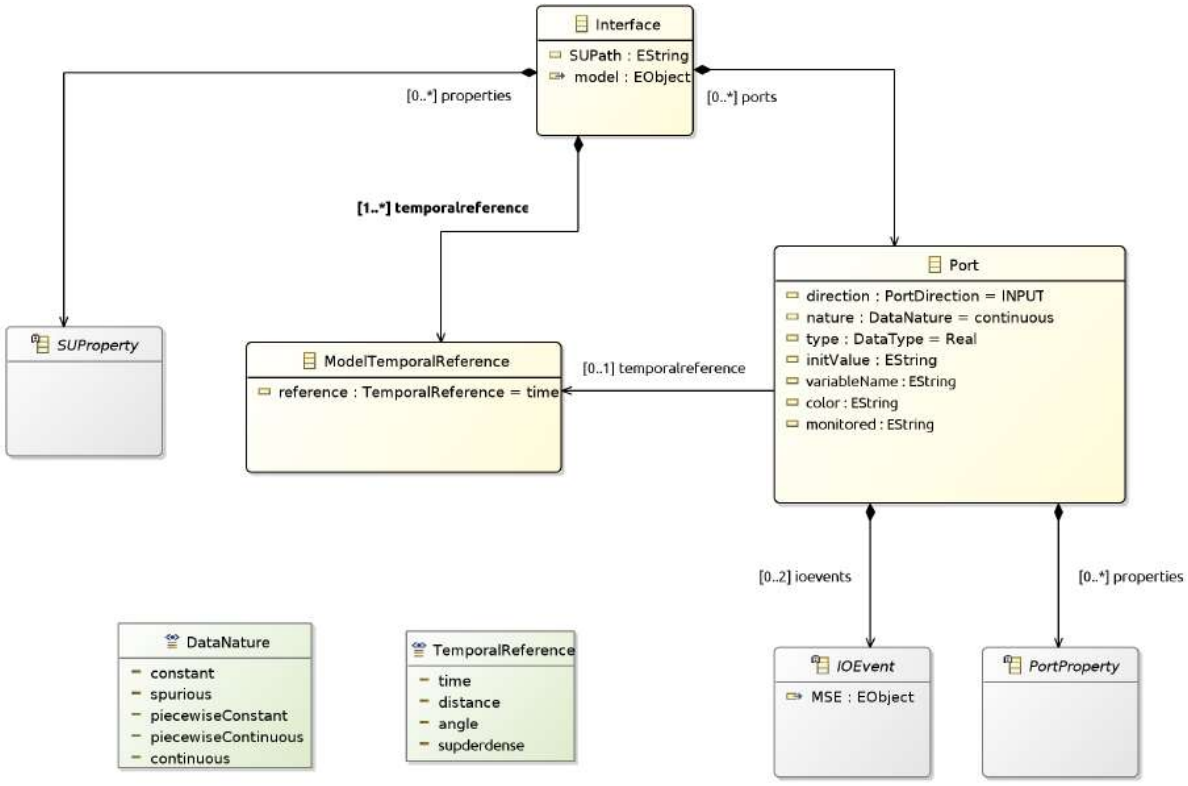

Рисунок 3.1 – Діаграма класів мови MCILang

Кореневим елементом є Інтерфейс. Інтерфейс має посилатися на виконуваний модуль моделювання. На даний момент ми підтримуємо два різних модулі моделювання: FMI, експортований як FMU, і Gemoc Executable Unit, експортований як окремий JAR-файл. The SUPath підтримує обидва типи модулів моделювання та використовується для визначення абсолютного шляху фактичного блоку моделювання. Потім він розкриває частковий погляд на семантику та синтаксис SU, що визначає три основні частини:

а) порти: набір портів містить деяку інформацію, яка буде використана пізніше для визначення специфікація координації. Зокрема, він розкриває тип і природу даних змінної, її часове посилання та її напрямок;

б) часове посилання моделі: часове посилання моделі розкриває тимчасове посилання SU, яке використовується для всіх змінних і використовується пізніше для визначення синхронізації між узгодженими моделями; на даний момент це фіксований перелік, закодований мовою інтерфейсу, але було б цікаво дозволити користувачеві визначення різних часових посилань на рівні моделі; тимчасове посилання можна, наприклад, виразити за допомогою MoCCML, що дозволяє специфікувати семантику різних часових посилань [177];

в) властивості модуля моделювання: це поле використовується для визначення специфіки модель, зокрема, властивості, які можна використати для визначення спеціального алгоритму координації, також можна реалізувати підтримку відкату.

Згідно з класичним визначенням порту в ADL, порт є абстракцією відкритої змінної модуля моделювання. Набір спільних змінних моделі доступний через інтерфейс, що забезпечує доступ із зовнішнього середовища. Щоб взаємодіяти із зовнішнім середовищем або іншими моделями, модель повинна відображати всі змінні, які можуть бути змінені зовнішнім середовищем. Кожен порт містить набір атрибутів і властивостей. Точніше кажучи, порт — це змінна, яку можна прочитати або встановити

ззовні відповідно до її напрямку. Порт відповідає визначенню, наведеному в раніше . Порт є структурованою змінною або структурою, що представляє змінну та її властивості.

3.2 Опис експериментального програмного забезпечення

У цьому розділі ми обговорюємо репрезентативний варіант експериментального програмного забезпечення, використаний для перевірки наших пропозицій. Валідація запропонованих підходів поділяється на чотири частини: визначення інтерфейсу координації моделі для кожного компонента, визначення специфікації координації моделі, створення структури, виконання та аналіз результатів.

Як приклад було використано керування температурою. Ця система складається з 3 блоків моделювання. CPUinBoxWithFan і fanControler були розроблені в інструменті OpenModelica відповідно реалізують процесор у коробці, який охолоджується вентилятором і контролер швидкості вентилятора (простий пропорційний контролер). Тепло між блоком і процесором передається відповідно до швидкості вентилятора. TheOverHeatController був розроблений як кінцевий автомат у студії GEMOC.

Приклад використання системи охолодження ЦП складається з трьох компонентів (рисунок 3.2): логічний контролер, написаний на TFSM, DSL, призначений для моделювання кінцевого кінцевого автомата з часом, контролер вентилятора, написаний на Modelica, змодельований як пристрій, що складається з центрального процесора та вентилятора. Вони написані як один компонент на Modelica. Потім отримана система складається з трьох моделей, написаних двома різними мовами, які відповідають двом різним семантикам. Крім того, ми висуваємо гіпотезу, що маємо справу з компонентами чорної скриньки: дві моделі Modelica потім експортуються як

два FMU, які вбудовують виконувану модель. Логічний контролер експортується як виконуваний двійковий файл, який відповідає Gemoc API для координації.

Отримані три блоки моделювання готові до використання в системі охолодження ЦП. У першій частині ми визначаємо для кожного SU відповідний інтерфейс координації моделі. У другій частині ми визначаємо три різні типи роз'ємів на основі характеру даних портів, відкритих у MCI. Зокрема, перший з'єднувач визначає спільне моделювання тригера часу, вказуючи найбільш відповідну частоту дискретизації для обміну даними між контролером вентилятора та блоком.

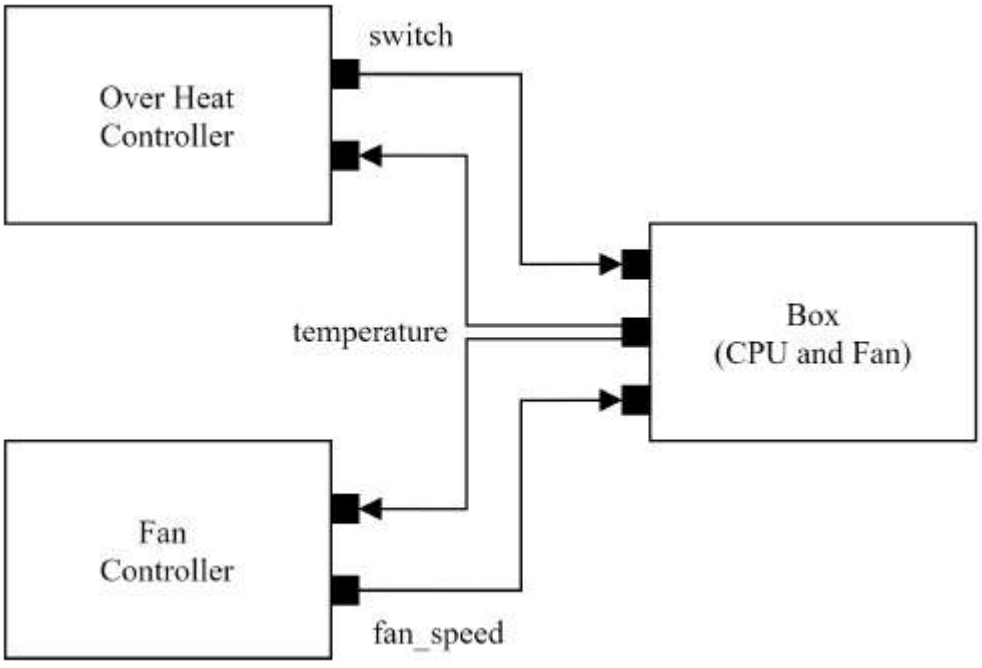

Рисунок 3.2 – Структура системи охолодження процесора

У другому конекторі ми визначаємо модель координації між логічним контролером і блоком, вказуючи спільне моделювання, кероване подіями, для відкритої події. У третьому роз'ємі ми визначаємо координацію між

блоком і логічним контролером як наскрізну петлю, де контролер виставляє вимогу зчитувати найбільш оновлене значення в певний момент часу. У третій частині ми генеруємо відповідну структуру виконання на основі специфікацій, написаних у першій і другій частинах. Нарешті, у четвертій частині ми виконуємо та аналізуємо результати спільного моделювання.

Підсистема, що складається з центрального процесора та вентилятора, вбудована в крпус та моделюється за допомогою OpenModelica (рисунок 5.3). BCPUinBoxWithFan - блок моделювання, ЦП активується, доки isStopped вхід дорівнює false. Під час активації ЦП виробляє тепло, яке більш-менш швидко обмінюється з повітрям його корпусу залежно від fanSpeedCommand входу (∈ [0..10], де при 0 вентилятор зупиняється, а при 10 вентилятор працює на повній швидкості).

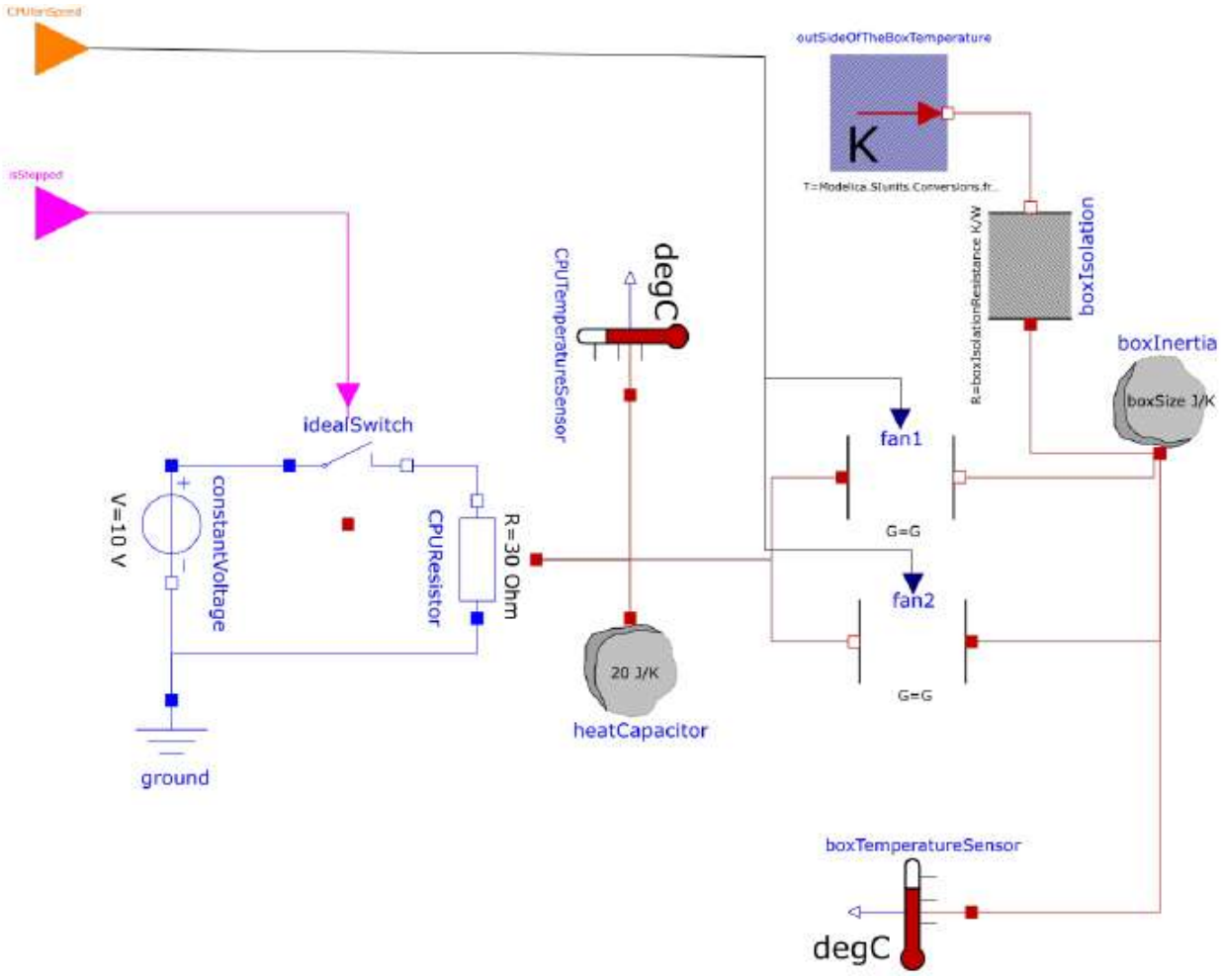

Рисунок 3.3 – Модель підсистеми охолодження в середовищі Modelica

Використовуючи запропонований інтерфейс координації моделі, ми визначаємо інтерфейс таким чином (лістинг 3.1).

В інтерфейсі координації моделі, визначеному в лістингу5.1, ми надаємо всю інформацію, таку як набір портів із їхніми властивостями, часове посилання, що використовується в моделі, і шлях виконуваної моделі.

Лістинг 3.1 – MCI текстове представлення моделі

```
Interface CPUinBox {
     FMUPath "su/ CPUinBoxWithFanHeatModel . fmu "
     ports {
       Port " BoxTemperature " {
          direction INPUT
          nature constant
          type Real
          initValue "25"
       },
       Port " CPUTemperature " {
          direction OUTPUT
          nature continuous
          type Real
          initValue " 0.0 "
       },
       Port " CPUfanSpeed " {
          direction INPUT
          nature continuous
          type Real
          initValue "0"
       },
       Port " isStopped " {
          direction INPUT
          nature piecewiseConstant
          type Boolean
          initValue " false "
       }
     }
     temporalreferences {
       ModelTemporalReference t {
          reference time
       }
     }
}
```

Відповідно до моделі (рисунок 3.3), список портів відображають відкриті змінні. Наприклад, CPUfanSpeed iisStopped визначаються як INPUT

порти типу Real та Boolean відповідно. Природа визначається відповідно з використанням змінної в моделі: theCPUfanSpeed визначає швидкість вентилятора і безпосередньо використовується для встановлення швидкості обох вентиляторів у моделі. В іншому випадку, isStopped керує ідеальним перемикачем, і його значення має змінюватися миттєво.

Контролер вентилятора – модель, написана на Modelica та розроблена на OpenModelica. Ми використовуємо вбудований простий безперервний контролер PID у бібліотеці Modelica. Як показано на рисунку 3.4, він представляє PID -регулятор з обмеженою потужністю, компенсацією проти перемотування та зважуванням заданого значення. Коефіцієнт посилення контролера керується вхідною змінною. Постійна часу блоку інтегратора встановлюється на 0.5. Постійна часу похідного блоку встановлюється на 0.1. Вихід X обмежується між 0 і 10. Вихід контролера перетворюється з Real значення в Integer значення.

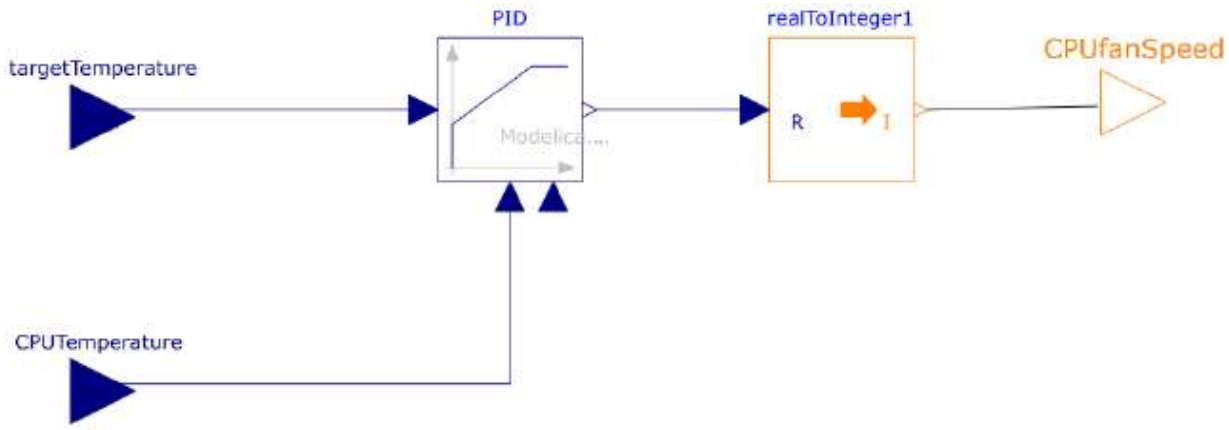

Рисунок 3.4 –Модель вентилятора

Щоб використовувати модель у нашій системі, ми експортуємо модель як FMU. На основі опису моделі FMI ми автоматично генеруємо відповідний інтерфейс координації моделі за допомогою функції імпорту IDE MCILang. Результуючий MCI контролера вентилятора представлений у лістингу 3.2.

Лістинг 3.2 – Опис інтерфейсу контролера вентилятора

```
Interface fanController {
     FMUPath "su/ FanController . fmu "
     ports {
          Port " CPUTemperature " {
               direction INPUT
               nature continuous
               type Real
               initValue "25"
               monitored false
          },
          Port " CPUfanSpeed " {
               direction OUTPUT
               nature continuous
               type Real
               monitored false
          },
          Port " targetTemperature " {
               direction INPUT
               nature constant
               type Real
               initValue "65"
               monitored false
          },
          Port "Kp" {
               direction INPUT
               nature constant
               type Real
               initValue " 5.0 "
               monitored false
          }
     }
     temporalreferences {
          ModelTemporalReference t {
               alidation 103
               reference time
          }
     }
}
```

Контролер перегріву OverHeatController – модель написана за допомогою DSL, розробленого в Gemoc Studio. Тоді контролер визначається як кінцевий автомат. Він періодично (кожні 3 секунди) перевіряє температуру процесора і, якщо він перевищит певного порогу, відбувається подія, і кінцевий автомат переходить у новий стан, де він відстежує температуру ЦП кожні 5 секунд. Якщо він перевищить певний поріг,

відбувається подія, і кінцевий автомат переходить у перший стан (рисунок 3.5). Отриману модель потім експортують як незалежну одиницю моделювання.

Отриманий MCI для OverHeatController представлено в лістингу 3.3.

Лістинг 3.3 – Опис інтерфейсу контролера перегріву

```
Interface overHeatController {
     GemocExecutablePath "su/ overHeatControler .jar :39635 "
     ports {
          Port " SwitchCPUState "
               {
               variableName " CPUprotection :: switchCPUState "
               direction OUTPUT
               nature transient
               type Boolean
               initValue " false "
               ioevents {
               Triggered occurs
               }
          },
          Port " CPUTemperature "
               {
               variableName " CPUprotection :: cpuTemperature "
               direction INPUT
               nature piecewiseConstant
               type Integer
               initValue "25"
               ioevents {
                    ReadyToRead readyToRead
               }
          }
     }
     temporalreferences
          {
               ModelTemporalReference t
                {
                    reference time
                }
          }
}
```

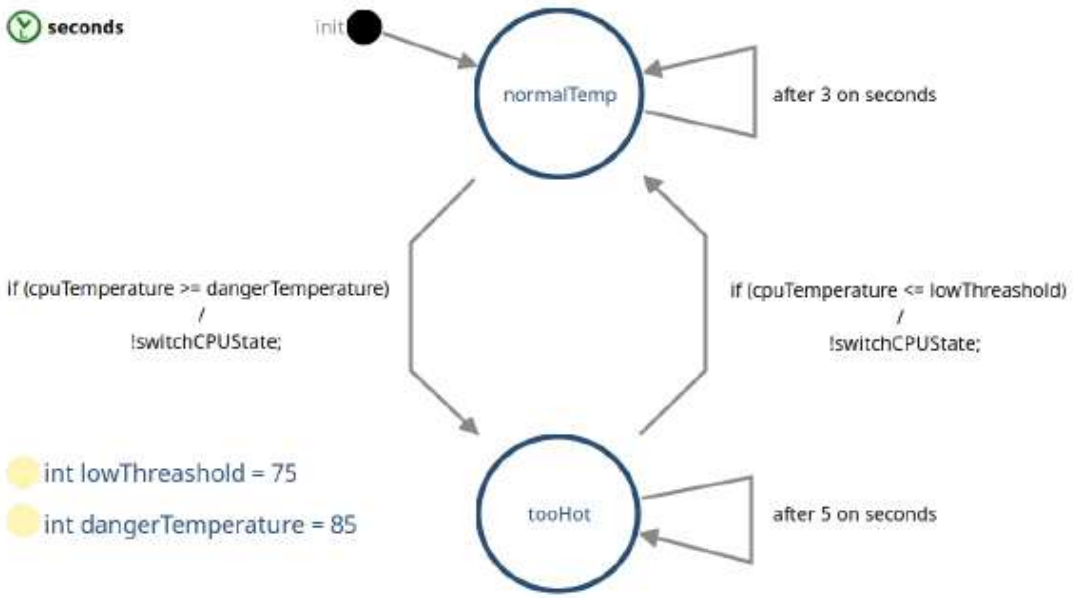

Рисунок 3.5 – Граф стану контролера перегріву

У порівнянні з інтерфейсами, представленими раніше, є два нові елементи: variable Name і ioevents. Атрибут variableName пов’язує внутрішню подію перемикача C-PUState з однойменним портом і виставляє через інтерфейс певну подію типу triggered, яка буде ініційована, коли внутрішня подія буде ініційована внутрішньо. У другому випадку порт буде пов'язаний з внутрішньою змінною, а пов’язана подія матиме тип ReadyToRead. У цьому випадку конкретна подія буде запущена до того, як вираз на guard буде оцінено для двох транзакцій до та з внутрішніх станів "нормальна температура" і "занадто жарко".

3.3 Результати моделювання

Результати спільного моделювання, отримані за допомогою описаноъ вишче системи, представлені на рисунку 3.6.

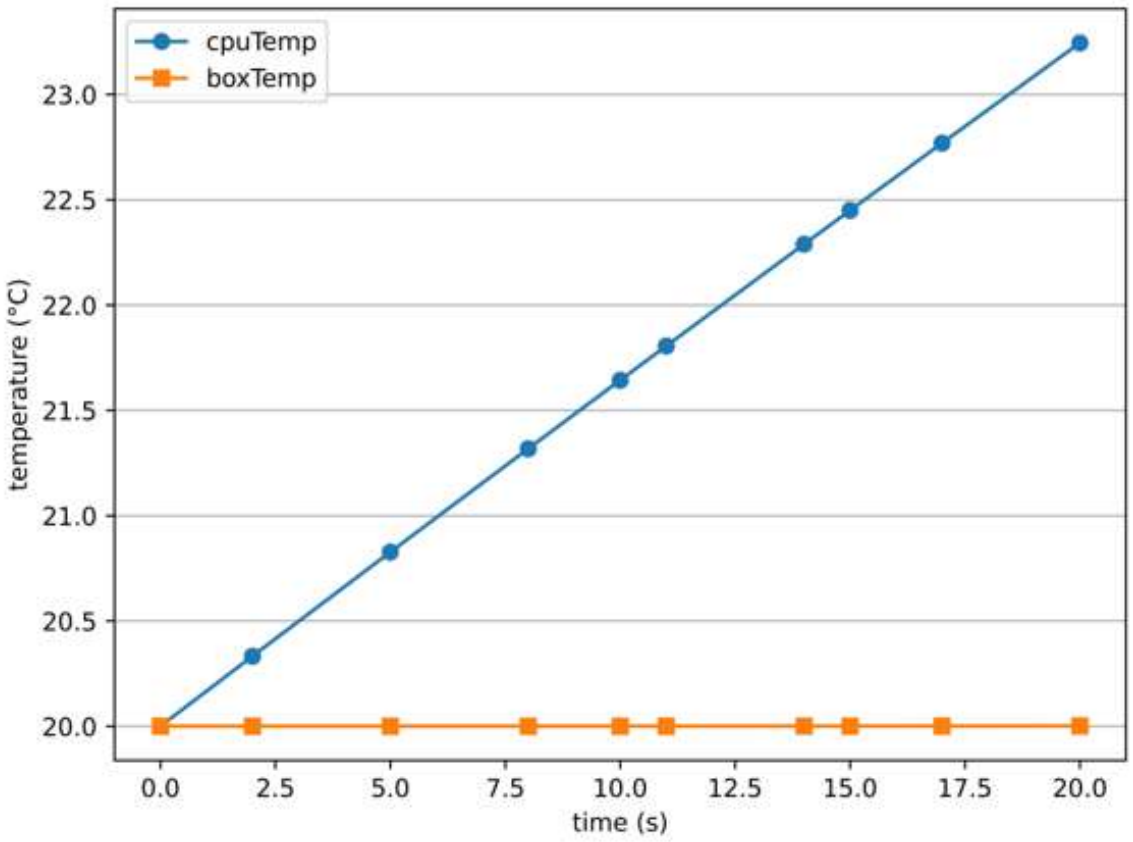

Рисунок 3.6 – Результати спільного моделювання

Треба зауважити, що точки фіксувалися лише так, як зазначено на рисунку 3.7, тобто в точний час необхідно мати правильну спільну симуляцію. Наприклад, на рисунку 3.6, ми бачимо, що перша пауза була реалізована контролером перегріву в момент 2, тобто це недетермінований час, витрачений кінцевим автоматом на входження внормальна температурастан, де оцінюється захисний вихідний перехід і, отже, зчитується температура ЦП. Потім реалізуються паузи кожні 5 секунд і кожні кратні 3 (період читання в першому стані OverHeatController). Таким чином, ми зменшуємо кількість точок зв’язку до їхнього мінімуму, щоб мати правильну спільну симуляцію, і ми уникаємо затримок, які вводить класична стратегія вибірки.

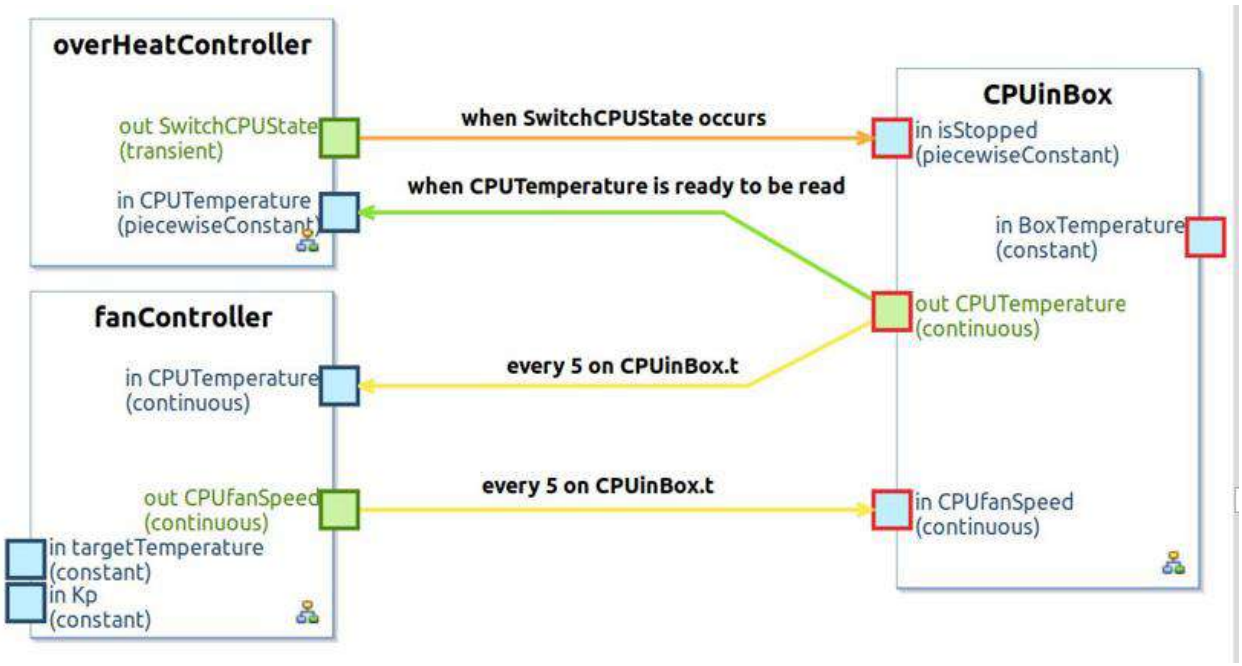

Рисунок 3.7 – Система охолодження ЦП

На рисунку 3.8, першим моментом часу є той, коли кінцевий автомат перемикається з нормальної температури до стану "занадто жарко". Це сталося в момент 14679.

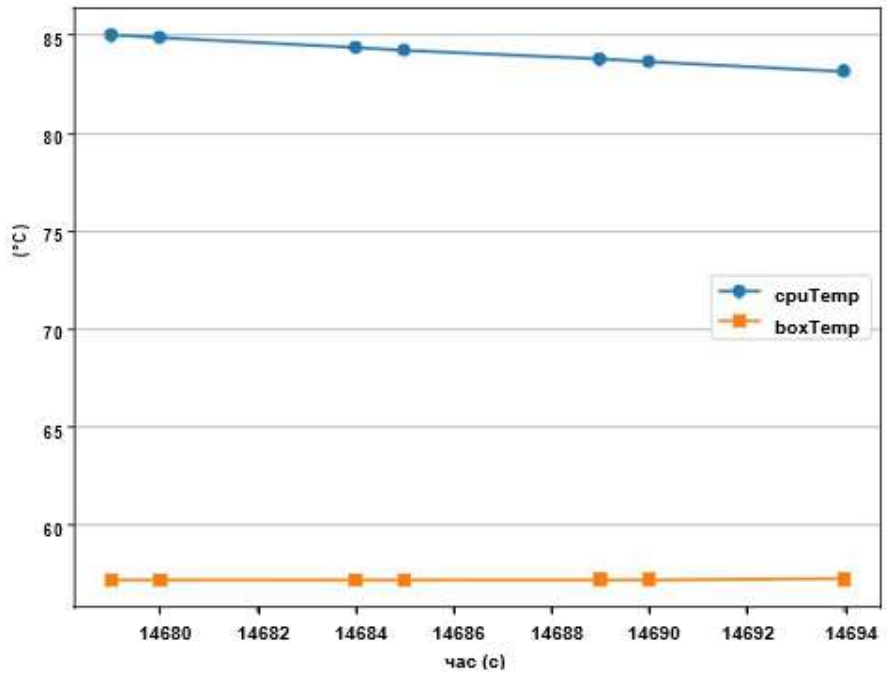

Рисунок 3.8 – Результати моделювання, отримані на вході контролера в стані "занадто жарко"

Отже, поки кінцевий автомат залишається в цьому стані, дані отримуються кожні 5 секунд, як зазначено в тимчасових з’єднувачах, і в період читання з кінцевого автомата. Однак, оскільки система увійшла в стан "занадто жарко" в момент часу 14679, модуль моделювання було призупинено через 5 секунд, тобто в 14684. Тут ми бачимо, що внутрішня семантика симуляції послідовно розкривається та враховується.

Запропонований інтерфейс є розширюваним, ефективним та інтуїтивно зрозумілим у використанні. Стосовно реалізації можна звернути увагу на два основні моменти. По-перше, її ефективність сильно залежить від внутрішньої реалізації API. У нашому випадку ми змінили генерацію коду, щоб за потреби генерувати паузу. Наприклад, для оновлення, усі присвоєння створено для створення паузи. Це лише незначно впливає на продуктивність. Однак, якщо реалізація виконується в обгортці, де всі мікрокроки перевіряються, щоб побачити, чи змінна була оновлена, тоді виконання може страждати від сповільнення. Якщо взяти вибірку змінної для перевірки перетину, виконання буде сповільнено, і точний момент часу, коли відбудеться перетин, може бути пропущено. Краще вводити фактичні переходи через нуль у модель (зазвичай у набір рівнянь), щоб забезпечити кращу продуктивність і точність. У нашій реалізації ми покладалися на анотації, щоб забезпечити гнучкість відкритої семантики. Отже, розробник інструменту відповідає за надання очікуваної семантики.

На рисунку 3.9 наведено результати виконання координаційного алгоритму.

Стосовно розширення предиката, слід звернути увагу на два незначні моменти. По-перше, важливо покладатися на механізм, щоб чітко визначити, який предикат підтримується для конкретної одиниці моделювання. Це можна, наприклад, зробити в артефактному еквіваленті опису моделі. ризик неконтрольованої еволюції предикатів, що призведе до вавилонської вежі предикатів. Це довгострокова проблема, але ризиків, що це станеться, мало.

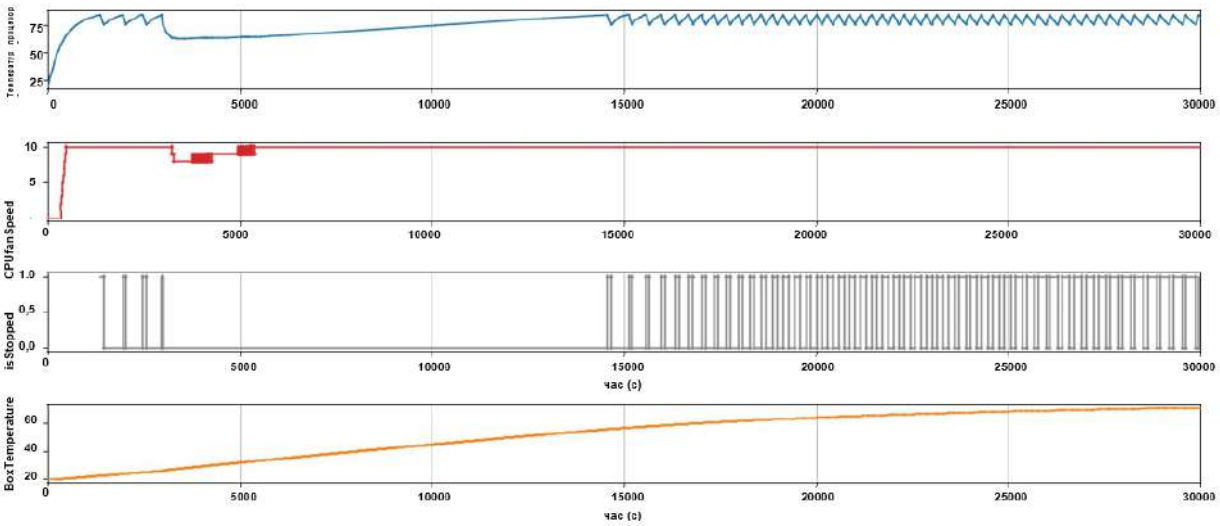

Рисунок 3.9 – Результати виконання координаційного алгоритму

Якщо буде прийнято шлях до цієї ситуації, може бути цікаво надати офіційний набір сховища розширень предикатів, де люди зможуть шукати існуючий предикат перед створенням власного, і де всі предикати зібрані разом.

Що стосується розміру варіанту використання, ми вирішили використовувати простий, але репрезентативний варіант використання, щоб зосередитися на взаємодії між модулями моделювання. Це дає нам змогу вивчати конкретну поведінку та моделі координації, не вводячи складності розробки промислового випадку використання.

ВИСНОВКИ

Розвиток кіберфізичних систем створює важливі виклики, пов’язані зі зростанням складності самих систем і процесів їх розробки. В атестаційній роботі розглядалася фаза моделювання розробки з особливим акцентом на інтеграції різнорідних моделей. Методи спільного моделювання дозволяють підтвердити та перевірити систему на ранній стадії розробки, але можуть виникнути деякі обмеження: внутрішня неоднорідність CPS призводить до використання різних спеціалізованих інструментів і мов, а співпраця між різними зацікавленими сторонами та підприємствами зростає. У контексті спільного моделювання було запропоновано декілька рішень для вирішення цих проблем, які забезпечують безперебійну співпрацю та обміну моделями між об’єктами (тобто підприємствами та зацікавленими сторонами). Стандарт FMI та стандарт HLA пропонують однорідні інтерфейси для моделювання безперервної та дискретної подій відповідно. Однак, у спільному моделюванні кіберфізичної системи його неоднорідність обмежує можливість використання однорідних стандартів спільного моделювання через їх обмеження з точки зору адаптивності та правильності.

Було проілюстровано основну семантику, яка склала кіберфізичні системи, і координаційну семантику, яка використовується для моделювання та спільного моделювання цих систем щодо їх природній семантики. Підходи спільного моделювання, засновані на безперервному часі та на основі дискретних подій, мають обмеження на інтеграцію гетерогенних систем через семантичний розрив між рідною семантикою та адаптованою семантикою. Ми показали, що цей проміжок може призвести до помилок спільного моделювання через невідповідність семантики моделей. Ми зазначили, що поточні підходи спираються на однорідні інтерфейси та забезпечують адаптацію між семантиками. Однак ці підходи не враховують

власну семантику моделей для визначення координаційної моделі, здатної правильно використовувати елементи семантики базових моделей.

У програмній і системній інженерії підходи, орієнтовані на координацію, пропонують чітко визначену інтеграцію для моделей, які відповідають різним семантикам. Зокрема, ми представили мови опису архітектури та мови координації, які зосереджені на вираженні явних моделей координації між різними обчислювальними об’єктами. Щоб отримати доступ до основної семантики, ці мови покладаються на спеціальні інтерфейси, які розкривають елементи семантики та синтаксису моделей. Було зазначено, що ці інтерфейси відповідають певній семантиці. Наприклад, стандарт FMI підтримує лише безперервні моделі, а запропонований інтерфейс прив’язує свою семантику до базової семантики моделі. У неоднорідному контексті інтерфейс повинен мати можливість виражати неоднорідність системи, одночасно забезпечуючи IP-захист моделі.

Було запропоновано структуру, присвячену моделюванню спільної симуляції. Фреймворк складається з двох доменно-специфічних мов та інтегрованого середовища моделювання, яке їх вбудовує. Він надає текстові та візуальні редактори з доповненням, перевіркою синтаксису та графічним редагуванням та підтримує дві запропоновані мови. MCL надає набір з’єднувачів, що дозволяє визначити правильну координацію між модулями моделювання. На основі цих визначень структура може автоматично генерувати код для розподіленого спільного моделювання. Дослідження показує різні переваги запропонованої структури. По-перше, вона дозволяє дизайнеру визначити правильну координацію, тобто координацію, для якої спільне моделювання не створює неочікуваних затримок. По-друге, це зменшує зв’язок між блоками до суворого мінімуму для забезпечення коректності. Менше спілкування означає кращу продуктивність моделювання. Нарешті, високий ступінь автоматизації в структурі усуває трудомістке завдання написання правильної координації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Edward A Lee. ‘Cyber physical systems: Design challenges’. In: Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on. IEEE. 2008, pp. 363–369.

2. Stefan Klikovits, Rima Al-Ali, Moussa Amrani, Ankica Barisic Fernando Barros, Dominique Blouin, Etienne Borde, Didier Buchs, Holger Giese, Miguel Goulão, Mauro Iacono, Florin Leon, Eva Navarro, Patrizio Pelliccione, and Ken Vanherpen. State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development. Jan. 2019. doi: 10.5281/zenodo.2533455.

3. Harinder Jagdev and Jim Browne. ‘The Extended Enterprise - A Context for Manufacturing’. In: Production Planning & Control - PRODUCTION PLANNING CONTROL 9 (Apr. 1998), pp. 216–229. doi: 10.1080/095372898234190

4. Modelisar. FMI for Model Exchange and Co-Simulation. July 2014. url: https://fmi-standard.org/downloads%5C#version2

5. Judith S Dahmann. ‘High level architecture for simulation’. In: Proceedings First International Workshop on Distributed Interactive Simulation and Real Time Applications. IEEE. 1997, pp. 9–14.

6. Jens Bastian, Christoph Clauß, SusannWolf, and Peter Schneider. ‘Master for Co-Simulation Using FMI’. In: 8th International Modelica Conference. 2011.

7. Tom Schierz, Martin Arnold, and Christoph Clauß. ‘Co-simulation with communication step size control in an FMI compatible master algorithm’. In: Proceedings of the 9th International MODELICA Conference; Munich; Germany. 076. Linköping University Electronic Press. 2012, pp. 205–214.

8. David Broman, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. ‘Requirements for Hybrid Cosimulation Standards’.

In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control. HSCC ’15. Seattle,Washington: Association for Computing Machinery, 2015, pp. 179–188. doi: 10.1145/2728606.2728629

9. David Broman, Christopher Brooks, Lev Greenberg, Edward A Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. ‘Determinate composition of FMUs for co-simulation’. In: Proceedings of the Eleventh ACM International Conference on Embedded Software. IEEE Press. 2013, p.2

10. Julien Deantoni, Cédric Brun, Benoît Caillaud, Robert France, Gabor Karsai, Oscar Nierstrasz, and Eugene Syriani. ‘Domain Globalization: Using Languages to Support Technical and Social Coordination’. In: Globalizing Domain-Specific Languages. Ed. by Combemale, Benoit, Cheng, Betty H.C., France, Robert B., Jézéquel, Jean-Marc, Rumpe, and Bernhard. Vol. 9400. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 70–87. doi: 10.1007/978-3-319-26172-0_5.

11. Giovanni Liboni, Julien Deantoni, Antonio Portaluri, Davide Quaglia, and Robert De Simone. ‘Beyond Time-Triggered Co-simulation of Cyber-Physical Systems for Performance and Accuracy Improvements’. In: 10th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools. Manchester, United Kingdom, Jan. 2018.

12. Sadaf Mustafiz, Cláudio Gomes, Hans Vangheluwe, and Bruno Barroca. ‘Modular design of hybrid languages by explicit modeling of semantic adaptation’. In: 2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS). Apr. 2016, pp. 1–8. doi: 10.23919/TMS.2016.7918835

13. Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. ‘Cosimulation: a Survey’. In: ACM Computing Surveys 51.3 (2018), Article 49. doi: 10.1145/3179993

14. Johan Eker, Jorn W Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, Yuhong Xiong, and Stephen Neuendorffer. ‘Taming heterogeneity - the Ptolemy approach’. In: Proceedings of the IEEE 91.1 (2003),

pp. 127–144.

15. Bert Van Acker, Joachim Denil, Hans Vangheluwe, and Paul De Meulenaere. ‘Generation of an Optimised Master Algorithm for FMI Co-simulation’. In: Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium. DEVS ’15. Alexandria, Virginia: Society for Computer Simulation International, 2015.

16. Stavros Tripakis, David Broman, and Computer Sciences. Bridging the Semantic Gap Between Heterogeneous Modeling Formalisms and FMI. Tech. rep. 2014.

17. Casper Thule, Cláudio Gomes, Julien Deantoni, Peter GormLarsen, Jörg Brauer, and Hans Vangheluwe. ‘Towards the Verification of Hybrid Co-simulation Algorithms’. In: Workshop on Formal Co-Simulation of Cyber-Physical Systems (SEFM satellite). Toulouse, France, June 2018.

18. Jean-Philippe Tavella, Mathieu Caujolle, Stephane Vialle, Cherifa Dad, Charles Tan, Gilles Plessis, Mathieu Schumann, Arnaud Cuccuru, and Sebastien Revol. ‘Toward an accurate and fast hybrid multi-simulation with the FMI-CS standard’. In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). 2016, pp.1–5.doi: 10.1109/ETFA.2016.7733616

19. Jean-Philippe Tavella, Mathieu Caujolle, Charles Tan, Gilles Plessis, Mathieu Schumann, Stephane Vialle, Cherifa Dad, Arnaud Cuccuru, and Sebastien Revol. ‘Toward an Hybrid Co-simulation with the FMI-CS Standard’. In: 2016.

20. David Garlan and Mary Shaw. ‘Introduction to software architecture’. In: Advanced Topics in Science and Technology in China January (1994), pp. 1–33. doi: 10.1007/978-3-540-74343-9_1

21. Nenad Medvidovic and Richard N Taylor. ‘A framework for classifying and comparing architecture description languages’. In: ACM SIGSOFT Software Engineering Notes 22.6 (1997), pp. 60–76.

22. George A Papadopoulos and Farhad Arbab. ‘Coordination models and languages’. In: Advances in computers 46 (1998), pp. 329–400.

23. Edward A Lee and Alberto Sangiovanni-Vincentelli. ‘A framework for comparing models of computation’. In: IEEE Transactions on computer-aided design of integrated circuits and systems 17.12 (1998), pp. 1217–1229.

24. Cécile Hardebolle and Frédéric Boulanger. ‘Modhel’x: A component-oriented approach to multiformalism modeling’. In: International Conference on Model Driven Engineering Languages and Systems. Springer. 2007, pp. 247–258.

25. Matias Ezequiel Vara Larsen, Julien Deantoni, Benoit Combemale, and Frédéric Mallet. ‘A Behavioral Coordination Operator Language (BCOoL)’. In: International Conference on Model Driven Engineering Languages and Systems (MODELS). Ed. by Timothy Lethbridge, Jordi Cabot, and Alexander Egyed. 18. to be published in the proceedings of the Models 2015 conference. Ottawa, Canada: ACM, Sept. 2015, p. 462

26. Erwan Bousse, Thomas Degueule, Didier Vojtisek, Tanja Mayerhofer, Julien Deantoni, and Benoit Combemale. ‘Execution framework of the gemoc studio (tool demo)’. In: Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering. ACM. 2016, pp. 84–89

27. Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert B France, Jean-Marc Jézéquel, and Jeff Gray. ‘Globalizing Modeling Languages’. In: Computer 47.6 (June 2014), pp. 68–71. doi: 10.1109/MC.2014.147

28. Fabio Cremona, Marten Lohstroh, David Broman, Edward A. Lee, Michael Masin, and Stavros Tripakis. ‘Hybrid co-simulation: it’s about time’. In: Software & Systems Modeling 18.3 (2019), pp. 1655–1679. doi:10.1007/s10270-017-0633-6

29. International Council on Systems Engineering, ed. INCOSE Systems Engineering Handbook. Vol. 2.0. 2000

30. Joel Moses. ‘Flexibility and Its Relation to Complexity and Architecture’. In: Complex Systems Design & Management. Ed. by Marc Aiguier, Francis Bretaudeau, and Daniel Krob. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 197–206.

31. Julio MOttino. ‘Engineering complex systems’. In: Nature 427.6973 (2004), pp. 399–399. doi: 10.1038/427399a.

32. Nicolai Pedersen., Kenneth Lausdahl., Enrique Vidal Sanchez., Peter Gorm Larsen., and Jan Madsen. ‘Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS’. In: Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH, INSTICC. SciTePress, 2017, pp. 73–82. doi: 10.5220/0006412700730082 (cited on page 10).

33. Cláudio Gomes, Hans Vangheluwe, and Paul De Meulenaere. ‘Property preservation in co-simulation’. PhD thesis. Ph. D. thesis, University of Antwerp, 2019.

34. Getachew F. Belete, Alexey Voinov, and Gerard F. Laniak. ‘An overview of the model integration process: From pre-integration assessment to testing’. In: Environmental Modelling Software 87 (2017), pp. 49–63. doi: https://doi.org/10.1016/j.envsoft.2016.10.013.

35. Jonathan L. Goodall, Bella F. Robinson, and Anthony M. Castronova. ‘Modeling water resource systems using a service-oriented computing paradigm’. In: Environmental Modelling Software 26.5 (2011), pp. 573–582. doi:https://doi.org/10.1016/j.envsoft.2010.11.013.

36. Bernd Neumayr, Michael Schrefl, and Bernhard Thalheim. ‘Modeling Techniques for Multi-level Abstraction’. In: Jan. 2008, pp. 68–92. doi: 10.1007/978-3-642-17505-3_4.

37. Heejung Chang and Kangsun Lee. ‘Applying Web Services and Design Patterns to Modeling and Simulating Real-World Systems’. In: Artificial Intelligence and Simulation. Ed. by Tag Gon Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 351–359.

38. Vadim Lisitsa, Vladimir Cheverda, and V. Volianskaia. ‘Numerical Simulation of Geological Faults by Discrete Elements Method’. In: June 2019. doi:

10.3997/2214-4609.201901680.

39. John A. Stankovic. ‘Misconceptions about real-time computing: a serious problem for next-generation systems’. In: Computer 21.10 (1988), pp. 10–19. doi: 10.1109/2.7053.

40. Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. ‘Model-integrated development of embedded software’. In: Proceedings of the IEEE 91.1 (2003), pp. 145–164.

41. Severin Sadjina, Lars Tandle Kyllingstad, Martin Rindarøy, Stian Skjong, Vilmar, and Eilif Pedersen. ‘Distributed Co-simulation of Maritime Systems and Operations’. In: Journal of Offshore Mechanics and Arctic Engineering 141.1 (Sept. 2018). 011302. doi: 10.1115/1.4040473

42. Robert M Argent. ‘An overview of model integration for environmental applications–components, frameworks and semantics’. In: Environmental Modelling Software 19.3 (2004). Concepts, Methods and Applications in Environmental Model Integration, pp. 219–234. doi: https://doi.org/10.1016/S1364-8152(03)00150-6.

43. W Ross Ashby. An introduction to cybernetics. Chapman & Hall Ltd, 1961.

44. BOSE Debayan. ‘Component Based Development-Application In Software Engineering’. In: Indian Statistical Institute (2011).

45. Jean Bézivin and Olivier Gerbé. ‘Towards a precise definition of the OMG/MDA framework’. In: Dec. 2001, pp. 273–280. doi: 10.1109/ASE.2001.989813.

46. Jean Bézivin. ‘From Object Composition to Model Transformation with the MDA.’ In: Jan. 2001, pp. 350–354. doi: 10.1109/TOOLS.2001.10021.

47. Jean Bézivin. ‘Model Driven Engineering: An Emerging Technical Space’. In: vol. 4143. Jan. 2005, pp. 36–64. doi: 10.1007/11877028_2 (cited on page 12).

48. Vicente García Díaz, Edward Núñez Valdez, Jordán Espada, B. Pelayo

García-Bustelo, Juan Cueva Lovelle, and Carlos Marín. ‘A brief introduction to model-driven engineering’. In: 18 (Apr. 2014), pp. 127–142.

49. Alberto Rodrigues da Silva. ‘Model-driven engineering: A survey supported by the unified conceptual model’. In: Computer Languages, Systems & Structures 43 (2015), pp. 139–155. doi: https://doi.org/10.1016/j.cl.2015.06.001.

50. Object Management Group. Object Management Group. http://www.omg.org/. Nov. 2020.

51. Denis Merigoux, Raphaël Monat, and Jonathan Protzenko. A Modern Compiler for the French Tax Code. 2020.

52. Douglas C. Schmidt. ‘Guest Editor’s Introduction: Model-Driven Engineering’. In: Computer 39.2 (2006), pp. 25–31. doi: 10.1109/MC.2006.58.

53. Gordon D Plotkin. ‘A structural approach to operational semantics’. In: (1981).

54. Robert W Floyd. ‘Assigning meanings to programs’. In: Program Verification. Springer, 1993, pp. 65–81.

55. Robert D. Tennent. ‘The Denotational Semantics of Programming Languages’. In: Commun. ACM 19.8 (Aug. 1976), pp. 437–453. doi:10.1145/360303.360308.

56. Frank Budinsky, Raymond Ellersick, David Steinberg, Timothy J Grose, and Ed Merks. Eclipse modeling framework: a developer’s guide. Addison-Wesley Professional, 2004.

57. Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ‘ATL: A model transformation tool’. In: Science of computer programming 72.1-2 (2008), pp. 31–39.

58. Krzysztof Czarnecki and Simon Helsen. ‘Classification of model transformation approaches’. In: Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture. Vol. 45. 3. USA. 2003, pp. 1–17.

59. Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro.

EMF: eclipse modeling framework. Pearson Education, 2008.

60. Zoé Drey, Cyril Faucher, Franck Fleurey, Vincent Mahé, and Didier Vojtisek. ‘Kermeta language’. In: Reference Manual (2009).

Jeff C Jensen, Danica H Chang, and Edward A Lee. ‘A model-based design methodology for cyberphysical systems’. In: 2011 7th International Wireless Communications and Mobile Computing Conference. 2011, pp. 1666–1671. doi: 10.1109/IWCMC.2011.5982785.