Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Кваліфікаційна наукова
праця на правах рукопису

ЛУК’ЯНЕЦЬ МИХАЙЛО ОЛЕКСАНДРОВИЧ

УДК 004.51 : 004.67

ДИСЕРТАЦІЯ

МЕТОДИ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВІЗУАЛІЗАЦІЇ МУЛЬТИМОДАЛЬНИХ ТЕМПОРАЛЬНИХ ПОТОКОВИХ ДАНИХ ПРИСТРОЇВ ІОТ

121 Інженерія програмного забезпечення
12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії
Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

М. О. Лук’янець

Науковий керівник: Сулема Євгенія Станіславівна, доктор технічних наук, доцент

Київ – 2026

АНОТАЦІЯ

Лук'янець М.О. Методи та програмне забезпечення візуалізації мультимодальних темпоральних потокових даних пристроїв IoT. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії в галузі знань 12 Інформаційні технології за спеціальністю 121 Інженерія програмного забезпечення. – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2025.

Стрімкий розвиток екосистем IoT та кіберфізичних систем спричинив створення нової парадигми цифрової взаємодії з фізичним світом, згідно з якою безперервно генеруються й циркулюють потоки мультимодальних темпоральних даних, які відображають сенсорні вимірювання, відео й аудіопотоки, дані телеметрії та подієвих журналів, що характеризуються різною частотою дискретизації, нерівномірними затримками, пропусками та шумами. Особливістю таких даних є їхня висока динамічність та неоднорідність, що ускладнює процес інтеграції, синхронізації та подальшої інтерпретації. Постійне зростання кількості джерел інформації вимагає від програмних систем здатності до масштабування під час інтенсивного навантаження. Візуалізація цих потоків у програмних системах виконує не лише функцію відображення, а й є ключовим інструментом оперативного аналізу, виявлення аномалій і підтримки прийняття рішень у режимі реального часу, з урахуванням людського фактору та вимог до інтерпретованості. Процеси візуалізації виступають проміжною ланкою між отриманням мультимодальних потоків «сирих» даних від пристроїв IoT і аналізом інформації про об’єкти моніторингу користувачем, забезпечуючи перехід від неструктурованих числових масивів до більш зрозумілих графічних моделей.

Попри істотний прогрес у створенні графічних інтерфейсів для візуалізації потокових даних залишається низка викликів: гетерогенність

форматів і протоколів, відсутність уніфікованих моделей представлення мультимодальних потоків, складність часової синхронізації за умов запізнілих і невпорядкованих подій, а також брак стандартизованих архітектур і шаблонів проєктування для систем візуалізації. Це ускладнює проєктування програмних систем, здатних адаптуватися до специфіки конкретних пристроїв без суттєвої зміни існуючих частин систем.

Окремою проблемою є забезпечення стабільності відображення раніше візуалізованих даних під час їх безперервного оновлення та ресемплінгу, що напряму впливає на довіру користувача до аналітичних матеріалів. Стабільність відображення у візуальному представленні зменшує кількість помилок оператора системи при тривалій роботі з візуалізованими даними. Ситуацію ускладнюють обмеження обчислювальних ресурсів (особливо на місцях роботи операторів) і когнітивне перевантаження операторів при роботі з багатоподієвими потоками високої інтенсивності.

Аналіз наявних програмних рішень свідчить про їхню фрагментованість і вузькоспеціалізованість, орієнтацію на окремі вертикалі пристроїв або конкретних виробників, обмежену переносимість і повторне використання компонентів. Подолання зазначених бар'єрів потребує створення уніфікованих, гнучких і масштабованих програмних підходів та рішень, які поєднують подієво-орієнтоване оброблення, механізми надійної часової узгодженості та архітектурні рішення для стабільної, інтерпретованої візуалізації мультимодальних потокових даних у реальному часі. Розв'язанню саме цієї науково-технічної задачі й присвячено дисертаційну роботу.

Метою дисертаційної роботи є підвищення ефективності процесів створення програмного забезпечення візуалізації потокових даних пристроїв IoT.

У першому розділі виконано аналіз методів оброблення та візуалізації IoT-даних, підходів до часової синхронізації та архітектур

потокового аналізу, сформульовано вимоги до програмного забезпечення і критерії якості. Окрему увагу приділено особливостям візуальної аналітики у доменах кібербезпеки та радіомоніторингу. Додатково проаналізовано обмеження відомих рішень щодо масштабованості та стабільності відображення даних, що визначило напрям подальших досліджень.

У другому розділі розроблено метод групування мультимодальних потокових даних з урахуванням темпорального параметру даних на основі підходів реактивного програмування. Запропонований підхід надає змогу обробляти події асинхронно, використовуючи механізми потокового планування та оброблення у реальному часі, що робить його придатним для високонавантажених систем IoT. Описано механізми узгодження часових міток і роботи з запізнілими подіями, що забезпечують коректне поєднання даних із різних каналів, навіть у разі різної частоти надходження інформації або імовірних затримок спричинених мережевими проблемами. Також перевірено швидкодію розробленого методу декількома замірами часу виконання для різної кількості вхідних даних.

У третьому розділі запропоновано метод стабільного відображення точкових темпоральних даних із даунсемплінгом у межах змінної частини даних, що не порушує уже відображені дані. На відміну від повного оновлення відображуваних даних, де зміна масштабів чи ресемплінг можуть призводити до перемальовування всієї сцени, у цьому підході введено принцип поділу буфера на застарілу, стабільну та активну зони, що уможливлює мінімізацію обчислювальних витрат та надає можливість зберегти візуальну узгодженість.

У четвертому розділі представлено узагальнену архітектуру програмного забезпечення, що підтримує централізовані та децентралізовані розгортання, наведено перелік виділених функціональних та нефункціональних вимог до програмного забезпечення IoT, запропоновані варіанти реалізації частин архітектури залежно від потреб, а також розроблено архітектурний шаблон проєктування програмного

забезпечення Multimodal Stream Stable Visualization (MSSV) для відображення мультимодальних потокових даних, який визначає організацію презентаційного шару та взаємодію його ключових компонентів між собою. Для цього архітектурного шаблону розроблено узагальнену схему користувацького інтерфейсу, що ілюструє узгоджене подання багатовимірних потоків і перехід від розрізненого відображення даних до інтегрованого інтерфейсу. Розглянуто можливості інтеграції архітектури з наявними інфраструктурами та шляхи масштабування під різні сценарії використання.

У дисертаційній роботі отримано низку **нових наукових результатів**, зокрема, **уперше** розроблено метод подієво-орієнтованого групування наборів мультимодальних темпоральних даних, особливістю якого є використання попередньо заданих часових вікон і правил відкладеного злиття пізніх подій та врахування заданої максимально допустимої розбіжності у часі надходження даних, що уможливлює групування даних за наявності лише частини потоків, які безперервно надходять з IoT пристроїв.

Уперше розроблено метод візуалізації, який, на від відміну від відомих методів, забезпечує незмінність вже візуалізованих фрагментів потоку під час надходження нових даних і застосування до них ресемплінгу, що дає змогу знизити навантаження на комп’ютерну систему в умовах високої інтенсивності передавання даних з IoT пристроїв.

Уперше розроблено узагальнену архітектуру програмного забезпечення візуалізації потокових даних пристроїв IoT, характерними рисами якої є: подієво-орієнтоване групування темпоральних мультимодальних даних та стабільне відображення даних великих обсягів у реальному часі, що забезпечує розширення функціональних можливостей візуалізації для систем оброблення даних IoT та уможливлює розроблення користувацьких інтерфейсів нового типу – подієво-орієнтованих інтерфейсів систем підтримки прийняття рішень.

Уперше розроблено архітектурний шаблон проєктування MSSV, який, на відміну від відомих, призначений для оперування згрупованими потоками даних, що робить можливим спрощення процесів розроблення програмних засобів візуалізації потокових даних пристроїв IoT із забезпеченням подієво-орієнтованого та синхронізованого відображення даних великих обсягів.

Основні наукові результати дисертаційної роботи **опубліковано** у 8 наукових працях, зокрема у 4 наукових статтях, опублікованих у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б», та у 4 матеріалах наукових конференцій.

Ключові слова: інженерія програмного забезпечення, Інтернет речей, кіберфізичні системи, потоки даних, мультимодальні темпоральні дані, архітектура програмних систем, візуалізація, прийняття рішень.

ABSTRACT

Lukianets M.O. Methods and Software for the Visualization of Multimodal Temporal Streaming Data from IoT Devices. – Qualification academic work, manuscript.

PhD thesis in the field of knowledge 12 Information technologies in speciality 121 Software Engineering. – National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2025.

The rapid development of IoT ecosystems and cyber-physical systems has led to the emergence of a new paradigm of digital interaction with the physical world, according to which continuous streams of multimodal temporal data are generated and circulate. These streams reflect sensor measurements, video and audio flows, telemetry data, and event logs characterized by different sampling rates, irregular latencies, missing values, and noise. A distinctive feature of such data is their high dynamism and heterogeneity, which complicates the processes of integration, synchronization, and subsequent interpretation. The constant growth in the number of information sources requires software systems to be scalable under intensive loads. Visualization of these streams in software systems performs not only a display function but also serves as a key instrument for operational analysis, anomaly detection, and real-time decision support, taking into account human factors and interpretability requirements. Visualization processes act as an intermediate layer between the acquisition of multimodal raw data streams from IoT devices and the user's analysis of information about monitored objects, ensuring the transition from unstructured numerical arrays to more comprehensible graphical models.

Visualization of these streams thus goes beyond simple rendering and becomes a crucial tool for real-time analytical reasoning, anomaly identification, and decision support, with due consideration of human factors and interpretability constraints. It serves as an intermediary between complex raw data streams and

the user, enabling the transformation of unstructured numerical data into intuitive graphical representations.

Despite significant progress in the development of graphical interfaces for streaming data visualization, a number of challenges remain: heterogeneity of formats and protocols, the lack of unified models for representing multimodal streams, the complexity of temporal synchronization under conditions of delayed and out-of-order events, as well as the shortage of standardized architectures and design patterns for visualization systems.

Despite this progress, these challenges persist, including heterogeneous data formats and protocols, the absence of unified representation models for multimodal streams, difficulties in temporal synchronization when events arrive late or out of order, and the lack of standardized architectures and design patterns for visualization systems. This complicates the design of software systems capable of adapting to the specifics of particular devices without significant changes to existing system components.

A separate challenge is ensuring the stability of previously visualized data during continuous updates and resampling, which directly affects user trust in analytical results. The stability of previously visualized data allows for minimizing operator errors during prolonged work with visualized data. This situation is further complicated by limited computational resources (especially at operator workstations) and the cognitive overload experienced by operators when working with high-intensity, multi-event data streams.

An analysis of existing solutions indicates fragmentation and narrow specialization, a focus on individual verticals or specific vendors, and limited portability and reusability of components. Many systems rely on ad hoc dashboards that do not generalize across modalities, and they lack principled mechanisms for temporal alignment, scalable data reduction, and reproducible visual states. Overcoming these barriers requires unified, flexible, and scalable software approaches that combine event-driven processing, robust mechanisms for temporal consistency, and architectural solutions for stable, interpretable real-

time visualization of multimodal data. This dissertation is devoted to addressing this scientific and technical problem.

The aim of this dissertation is to improve the efficiency of the processes of creating software for visualizing streaming data from IoT devices.

In the first chapter, a review is provided of methods for processing and visualizing IoT data, approaches to time synchronization, and streaming analytics architectures, and the software requirements and quality criteria are formulated. Special attention is given to the specifics of visual analytics in the domains of cybersecurity and radio monitoring. Additionally, the limitations of existing solutions regarding scalability and stability of data visualization are analyzed, which determined the direction of further research.

In the second chapter, a method is developed for grouping multimodal streaming data that accounts for the temporal parameter of the data, based on reactive programming approaches. The proposed approach enables asynchronous event handling using stream scheduling and real-time processing mechanisms, making it suitable for high-throughput IoT systems. Mechanisms for timestamp alignment and handling of late events are described, ensuring the correct combination of data from different channels, even when arrival rates differ or delays arise due to network issues. The performance of the developed method was also verified through several execution time measurements for varying amounts of input data.

In the third chapter, a method is proposed for stable visualization of point-based temporal data with downsampling applied within the variable portion of the data, without affecting data that have already been displayed. Unlike full updates of the display, where rescaling or resampling may trigger redrawing of the entire scene, this approach introduces a three-zone buffer-outdated, stable, and active- which minimizes computational costs and preserves visual consistency.

In the fourth chapter, a generalized software architecture is presented that supports centralized and decentralized deployments; implementation options for architectural components are proposed depending on requirements; and an

architectural software design pattern Multimodal Stream Stable Visualization (MSSV) for visualizing multimodal streaming data is developed, defining the organization of the presentation layer and the interaction of its key components. For this architectural pattern, a generalized user interface scheme is provided, illustrating the coordinated presentation of multidimensional streams and the transition from fragmented data views to an integrated interface. The possibilities for integrating the architecture with existing infrastructures and scaling paths for different usage scenarios are also considered.

In this dissertation, a **number of novel scientific results were obtained**, in particular: **for the first time**, an event-driven method for grouping sets of multimodal temporal data has been developed, characterized by the use of predefined time windows and rules for deferred merging of late-arriving events, as well as accounting for a specified maximum permissible discrepancy in data arrival time, which enables grouping even when only part of the data streams continuously arriving from IoT devices is available.

For the first time, a specialized visualization method has been established which, in contrast to existing methodologies, guarantees the persistence and immutability of previously rendered stream segments during the arrival of new data and the subsequent application of resampling techniques, thereby substantially mitigating the computational overhead within systems characterized by high-intensity IoT data transmission.

For the first time, a generalized software architecture for IoT streaming data visualization has been developed, characterized by event-driven grouping of temporal multimodal data and stable real-time display of large data volumes, which expands visualization functionality for IoT systems and enables the creation of a new type of user interface – event-driven interfaces for decision support systems.

For the first time, the MSSV architectural design pattern has been introduced, specifically engineered to manage and manipulate aggregated data streams – unlike conventional patterns –thereby streamlining the developmental

processes of IoT streaming data visualization software while ensuring strictly synchronized and event-driven rendering of high-volume data.

The main scientific results of the dissertation are published in 8 scientific works, including 4 articles in specialized journals recognized by the list of scientific specialized editions of Ukraine (category “Б”) and 4 conference proceedings.

Keywords: software engineering, Internet of Things, cyber-physical systems, data streams, multimodal temporal data, software system architecture, visualization, decision making.

Список публікацій здобувача / List of publications of the applicant:

Статті у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б» / the articles published in scientific journals included in the list of scientific journals of Ukraine in category «Б»:

1. Mykhailo Lukianets, Yevgeniya Sulema. Real-time data visualization for IoT network systems: challenges and strategies for performance optimization // System technologies. – 2023. – Vol. 5. № 148. – С. 52–61. Внесок М. О. Лук'янця: аналіз існуючих методів та систем візуалізації та виявлення можливих напрямків оптимізації роботи систем оброблення даних в мережах IoT. Внесок Є. С. Сулеми: визначено теоретичні засади оптимізації в межах систем IoT реального часу.
2. М.О. Лук'янець, Є.С. Сулема. Метод угрупування темпоральних мультимодальних потокових даних пристроїв IoT на основі реактивного програмування // Системні технології. 2024. –Т. 4. – № 153. – С. 123–132. Внесок М. О. Лук'янця: розроблення методу групування даних, його алгоритмізація та перевірка ефективності. Внесок Є. С. Сулеми: обґрунтування підходу до застосування парадигми реактивного програмування для агрегації мультимодальних даних.
3. Лук'янець М., Сулема Є. Метод відображення точкових темпоральних мультимодальних потокових даних пристроїв IoT зі збереженням стабільності відображення попередньо візуалізованих даних // Herald of khmelnytskyi national university. Technical sciences. – 2025. – Т. 347. – №. 1. – С. 245-250. Внесок М. О. Лук'янця: розроблення алгоритму стабілізації відображення та проведення серії експериментів для підтвердження роботи методу. Внесок Є. С. Сулеми: розроблення концепції забезпечення візуальної стабільності при відображенні високоінтенсивних потоків даних.

4. Лук’янець М., Сулема Є. Високорівнева архітектура системи відображення потокових темпоральних мультимодальних даних мереж пристроїв IoT // Herald of khmelnytskyi national university. Technical sciences. – 2025. – Т. 355. – №. 4. – С. 303-308. Внесок М. О. Лук’янця: розроблення концепції побудови архітектури системи відображення, розроблення функціональних модулів та механізмів оброблення потокових даних. Внесок Є. С. Сулеми: визначення принципів взаємодії компонентів.

Публікація у матеріалах наукової конференції / the publication in the proceedings of the scientific conference:

5. Лук’янець, М. (2024). Угрупування темпореальних мультимодальних потокових даних пристроїв IoT: особливості часової синхронізації // *Матеріали конференцій МЦНД.* - Харків:10.05.2024. – С. 117–118.
6. Лук’янець , М. (2024). Проблеми та можливі рішення оптимізації відображення даних мереж IoT у реальному часі // *Collection of Scientific Papers «ΛΌГОΣ».* – Zurich: May 24, 2024. – С. 190–191.
7. М.О. Лук’янець, Є.С. Сулема. (2024) Алгоритм оптимізації процесу відображення темпоральних мультимодальних потокових даних пристроїв IoT // Прикладна математика та комп’ютинг ПМК' 2024 : збірник тез доповідей Сімнадцятої конференції магістрантів та аспірантів (20-22 листопада 2024 р. Київ, Україна). – Київ, 2024. – С. 159-163. Внесок М. О. Лук’янця: розроблення алгоритму оптимізації процесу відображення даних пристроїв IoT. Внесок Є. С. Сулеми: визначення теоретичних засад оптимізації продуктивності систем візуалізації в реальному часі.
8. Лук’янець, М. (2026). Шаблон проєктування Multimodal Stream Stable Visualization (MSSV) для відображення мультимодальних потокових даних у системах реального часу // 3 Міжнародна науково-практична конференція Innovative Research in Science and Economy. – Брюссель: Січень 2026.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 16

ВСТУП 18

РОЗДІЛ 1. АНАЛІЗ ВІДОМИХ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ ВІЗУАЛІЗАЦІЇ ТЕМПОРАЛЬНИХ МУЛЬТИМОДАЛЬНИХ ДАНИХ 24

1.1. Сучасний стан програмного забезпечення візуалізації даних 24

1.2. Критерії оцінювання візуалізації даних IoT 35

1.3. Аналіз вимог до програмного забезпечення візуалізації та порівняльний аналіз відомих програмних рішень 38

1.4. Висновки до першого розділу 44

РОЗДІЛ 2. РОЗРОБЛЕННЯ МЕТОДУ ПОДІЄ-ОРІЄНТОВАНОГО ГРУПУВАННЯ ПОТОКОВИХ ДАНИХ ПРИСТРОЇВ ІОТ 46

2.1. Задача групування потоків даних та шляхи її розв’язання 46

2.2. Метод групування темпоральних мультимодальних даних 51

2.3. Програмна реалізація методу групування 60

2.4. Оцінювання ефективності розробленого методу 67

2.5. Візуалізація згрупованих подієвих даних 73

2.6. Висновки до другого розділу 75

РОЗДІЛ 3. РОЗРОБЛЕННЯ МЕТОДУ ВІЗУАЛІЗАЦІЇ ТЕМПОРАЛЬНИХ МУЛЬТИМОДАЛЬНИХ ПОТОКОВИХ ДАНИХ ПРИСТРОЇВ ІОТ ЗІ ЗБЕРЕЖЕННЯМ СТАБІЛЬНОСТІ ВІДОБРАЖЕННЯ 77

3.1. Формулювання вимог до візуалізації IoT-даних 77

3.2. Розроблений метод візуалізації 79

3.3. Програмна реалізація розробленого методу 88

3.4. Оцінювання ефективності розробленого методу 92

3.5. Висновки до третього розділу 101

РОЗДІЛ 4. РОЗРОБЛЕННЯ УЗАГАЛЬНЕНОЇ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІДОБРАЖЕННЯ ПОТОКОВИХ ДАНИХ ПРИСТРОЇВ IoT 103

4.1. Загальні вимоги до архітектури програмного забезпечення для роботи з потоковими даними пристроїв IoT 103

4.2. Функціональні та нефункціональні вимоги 107

4.3. Розроблена узагальнена архітектура програмного забезпечення 110

4.4. Аналіз розробленої архітектури 124

4.5. Впровадження програмного забезпечення візуалізації даних пристроїв IoT 128

4.6. Шаблон проєктування програмного забезпечення для відображення потокових даних Multimodal Stream Stable Visualization (MSSV) 133

4.7. Узагальнена схема та принципи побудови користувацького інтерфейсу для розробленого шаблону проєктування 137

4.8. Висновки до четвертого розділу 143

ВИСНОВКИ 145

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ 148

ДОДАТКИ 159

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

2D – Two-Dimensional (двовимірний формат)

3D – Three-Dimensional (тривимірний формат)

.NET – Microsoft .NET (кросплатформна програмна платформа)

API – Application Programming Interface (інтерфейс програмування застосунків)

AR – Augmented Reality (доповнена реальність)

Apache Kafka –розподілена стримінгова платформа, черга повідомлень

AWS – Amazon Web Services (веб-сервіси Amazon)

BenchmarkDotNet –фреймворк бенчмарків для .NET

CPU – Central Processing Unit (центральний процесор)

DirectX – Microsoft DirectX (набір мультимедійних і графічних API)

Docker –платформа контейнеризації

GPU – Graphics Processing Unit (графічний процесор)

gRPC – високопродуктивний фреймворк віддалених викликів процедур

HTTP – Hypertext Transfer Protocol (протокол передавання гіпертексту)

IIoT – Industrial Internet of Things (промисловий Інтернет речей)

IoT – Internet of Things (Інтернет речей)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

Kubernetes –платформа оркестрації контейнерів

LINQ – Language Integrated Query (вбудована в мову платформа запитів)

MQTT – Message Queuing Telemetry Transport (легковаговий протокол публікації–підписки для телеметрії)

Protobuf – Protocol Buffers (формат серіалізації даних Google)

REST – Representational State Transfer (архітектурний стиль веб-служб)

Rx.NET – Reactive Extensions for .NET (бібліотека реактивного програмування для .NET)

SciChart – SciChart (бібліотека візуалізації даних для .NET)

SQL – Structured Query Language (мова структурованих запитів)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (користувацький досвід взаємодії)

Vulkan –низькорівневий графічний API

WPF – Windows Presentation Foundation (фреймворк побудови інтерфейсів у .NET)

WebSocket –протокол двонапрямленого з’єднання в реальному часі

Ресемплінг – процедура зміни частоти дискретизації та/або приведення часових рядів до спільної часової шкали й кроку вибірки

Даунсемплінг – різновид ресемплінгу, за якого кількість точок даних навмисно зменшується для зниження обчислювального навантаження і збереження ключових трендів та екстремумів під час аналізу чи візуалізації в реальному часі

ВСТУП

Актуальність теми. Стрімкий розвиток Інтернету речей (IoT) та промислового Інтернету речей (IIoT) за останні 5–10 років зумовив появу масивів темпоральних мультимодальних даних, що безперервно надходять від різнорідних передавачів, мобільних і вбудованих пристроїв. У таких сферах, як промислова автоматизація, Smart City, eHealth, енергетика та екологічний моніторинг, оперативність і якість прийняття рішень безпосередньо залежать від здатності систем у реальному часі об'єднувати, обробляти та відображати потоки даних. Візуалізація відіграє значну роль у цьому процесі. Саме вона забезпечує швидке сприйняття стану об'єктів, виявлення аномалій і тенденцій. Також відповідає за формування пояснювальних інтерпретацій даних для фахівців різних доменів.

Технології потокового оброблення та візуалізації темпоральних мультимодальних даних пристроїв IoT уже знайшли широке застосування у промисловості, виробництві, логістиці, енергетиці та міській інфраструктурі. Зокрема, у виробничій сфері такі рішення використовуються для оперативного нагляду за технологічними процесами, підвищення надійності устаткування, раннього виявлення аномалій і профілактичного обслуговування. У логістиці аналітика потокових даних дає змогу вдосконалювати керування ланцюгами постачання, оптимізувати маршрути, зменшувати витрати та підвищувати точність прогнозування запасів і термінів поставок. В енергетиці інструменти візуальної аналітики підтримують балансування навантажень, керування розподіленою генерацією та виявлення неефективностей у мережах. Крім того, системи візуалізації сприяють сталому розвитку, оскільки вони надають можливість оцінювати вплив різних сценаріїв експлуатації технічних об'єктів і міської інфраструктури на довкілля в режимі близькому до реального часу.

Водночас існує значний потенціал для ширшого застосування візуальної аналітики потокових IoT-даних у людиноцентричних галузях,

наприклад, у медицині такі підходи можуть підтримувати дистанційний моніторинг пацієнтів, формування попереджувальних клінічних сигналів, точний аналіз результатів обстежень і прогнозування стану на основі мультимодальних даних.

Сучасні системи візуалізації стикаються з низкою викликів попри значний прогрес у засобах збору та транспортування даних. Серед них висока швидкість надходження даних (кількість надходжень на секунду), їхнім обсягом, різнорідністю форматів і мультимодальністю самих даних, а також необхідністю роботи з особливостями темпоральних даних, мінімізації когнітивного навантаження користувачів і, відповідно, забезпечення стабільності відображення. Поширені реалізації часто орієнтовані на вузькі випадки використання, не пропонують уніфікованої архітектури та не забезпечують масштабованості для зростаючих навантажень, включно з гібридними сценаріями, що включають розміщення частково на хмарних рішеннях, а частково на власне робочих місцях операторів систем.

Таким чином, актуальною науково-технічною задачею є вдосконалення теоретичних засад і програмних рішень для візуалізації темпоральних мультимодальних потокових даних пристроїв IoT у реальному часі з низьким ресурсним споживанням і підтримкою масштабованої архітектури. Розв’язання цієї задачі створює підґрунтя для розроблення надійних систем моніторингу та прогнозування в промисловості, міській інфраструктурі та медичних застосунках.

Зв’язок роботи з науковими програмами, планами, темами. Дослідження виконані у рамках держбюджетної НДР «Математичні та програмні методи оброблення мультимодальних даних моніторингу медико-біологічних об’єктів для діагностики стану здоров’я пацієнтів» (№ держреєстрації 0120U102134), яка виконувалася у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського».

Мета і задачі дослідження. Метою дисертаційної роботи є підвищення ефективності процесів створення програмного забезпечення візуалізації потокових даних пристроїв IoT.

Для досягнення поставленої мети вирішено наступні основні задачі.

1. Аналіз сучасних підходів до роботи з потоковими мультимодальними даними та їх відображення.
2. Аналіз вимог до програмного забезпечення відображення даних пристроїв IoT.
3. Розроблення методів подіє-орієнтованого групування та стабільного відображення потокових даних.
4. Проєктування узагальненої архітектури програмного забезпечення візуалізації потокових даних пристроїв IoT.
5. Розроблення та забезпечення якості програмного забезпечення візуалізації потокових даних пристроїв IoT.
6. Експериментальне дослідження ефективності розроблених методів та програмного забезпечення.

Об’єкт дослідження: процеси аналізу вимог, розроблення, забезпечення якості та впровадження програмного забезпечення візуалізації потокових даних пристроїв IoT.

Предмет дослідження: методи, способи та програмні засоби для візуалізації потокових даних.

Методи дослідження: теорія програмування та програмних систем, методи потокового оброблення даних і реактивного програмування, методи проєктування архітектур програмних систем, теорія паралельних та розподілених обчислень.

Наукова новизна одержаних результатів полягає у наступному:

1. **Уперше** розроблено метод подієво-орієнтованого групування наборів мультимодальних темпоральних даних, особливістю якого є використання попередньо заданих часових вікон і правил відкладеного злиття пізніх подій та врахування заданої

максимально допустимої розбіжності у часі надходження даних, що уможливлює групування даних за наявності лише частини потоків, які безперервно надходять з IoT пристроїв.

2. **Уперше** розроблено метод візуалізації, який, на від відміну від відомих методів, забезпечує незмінність вже візуалізованих фрагментів потоку під час надходження нових даних і застосування до них ресемплінгу, що дає змогу знизити навантаження на комп'ютерну систему в умовах високої інтенсивності передавання даних з IoT пристроїв.
3. **Уперше** розроблено узагальнену архітектуру програмного забезпечення візуалізації потокових даних пристроїв IoT, характерними рисами якої є: подієво-орієнтоване групування темпоральних мультимодальних даних та стабільне відображення даних великих обсягів у реальному часі, що забезпечує розширення функціональних можливостей візуалізації для систем оброблення даних IoT та уможливлює розроблення користувацьких інтерфейсів нового типу – подієво-орієнтованих інтерфейсів систем підтримки прийняття рішень.
4. **Уперше** розроблено архітектурний шаблон проєктування MSSV, який, на відміну від відомих, призначений для оперування згрупованими потоками даних, що робить можливим спрощення процесів розроблення програмних засобів візуалізації потокових даних пристроїв IoT із забезпеченням подієво-орієнтованого та синхронізованого відображення даних великих обсягів.

Практичне значення одержаних результатів полягає у створенні методів, архітектурних рішень та узагальненої схеми користувацького інтерфейсу, які спрощують процеси розроблення програмних засобів для візуалізації темпоральних мультимодальних потокових даних пристроїв IoT. Зокрема, результати дослідження можуть бути використані під час проєктування систем моніторингу промислових процесів, міської

інфраструктури, медичних застосунків та освітніх платформ. Запропоновані рішення забезпечують стабільність відображення даних у реальному часі, знижують навантаження на обчислювальні ресурси, що робить їх корисними для розробників програмного забезпечення, що створюють IoT рішення для підприємств і організацій, зацікавлених у впровадженні сучасних технологій оброблення та візуалізації потокових даних.

Особистий внесок здобувача. Усі основні результати дисертаційного дослідження, які представлені до захисту, одержані автором особисто. У публікаціях, написаних у співавторстві, здобувач є основним автором результатів, пов'язаних із методами.

У роботі [1] здобувачем проаналізовано проблеми та запропоновано стратегії оптимізації продуктивності під час візуалізації потокових даних у системах IoT у реальному часі. У роботі [2] здобувачем запропоновано метод групування темпоральних мультимодальних потокових даних пристроїв IoT на основі реактивного програмування. У роботі [3] здобувачем розроблено метод відображення точкових темпоральних мультимодальних потокових даних зі збереженням стабільності вже візуалізованих сегментів. У роботі [4] здобувачем запропоновано узагальнену архітектуру системи відображення потокових темпоральних мультимодальних даних пристроїв IoT.

Апробація результатів дисертації. Основні результати дисертаційного дослідження доповідалися та обговорювалися на наукових конференціях:

1. VI Міжнародна науково-практична конференція «Grundlagen der modernen wissenschaftlichen forschung», Цюрих, Швейцарія, 2024.
2. IV Міжнародна наукова конференція «Стратегічні напрямки розвитку науки: фактори впливу та взаємодії», Харків, Україна, 2024.
3. Сімнадцята конференція магістрантів та аспірантів Прикладна математика та комп'ютинг ПМК-2024. Київ, Україна, 2020.

4. 3 Міжнародна науково-практична конференція Innovative Research in Science and Economy. Брюссель, Бельгія, 2026.

Публікації. Основні наукові результати дисертаційної роботи опубліковано у 8 наукових працях, зокрема у 4 наукових статтях, опублікованих у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б», та у 4 матеріалах наукових конференцій.

Структура роботи. Дисертація складається із вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг становить 175 сторінок, включаючи 158 сторінок основного тексту, 25 рисунків та 10 таблиць.

РОЗДІЛ 1. АНАЛІЗ ВІДОМИХ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ ВІЗУАЛІЗАЦІЇ ТЕМПОРАЛЬНИХ МУЛЬТИМОДАЛЬНИХ ДАНИХ

1.1. Сучасний стан програмного забезпечення візуалізації даних

Однією з найскладніших задач у 21 столітті є збирання великих даних. Швидкий розвиток пристроїв збору даних і засобів зберігання даних робить можливим збирання величезних обсягів інформації з різних джерел.

Наприклад, до 2003 року інформація про 20 000-25 000 генів у ДНК людини та послідовності 3 000 000 000 пар хімічних основ, які складають ДНК людини, була зібрана у рамках проєкту геному людини, щоб допомогти зрозуміти, як функціонує людський організм [1-5]. У застосунку Europe Media Monitor щодня збирається близько 100 000 статей із понад 2500 джерел новин 43 мовами, щоб узагальнювати світові новини та надсилати сповіщення про них у стислій формі. Ці дані є багатими джерелами інформації, які можуть допомогти підтримати наукові відкриття та допомогти у прийнятті рішень. Однак отримати значущі знання з таких даних складно. Люди часто стикаються з різнорідною, суперечливою та динамічною інформацією з багатьох різноманітних джерел і стикаються з проблемою відсутності можливості ефективно її аналізувати. Це добре відома проблема перевантаження інформацією. Для її розв’язання доцільно застосовувати візуальну аналітику, що поєднує автоматизований аналіз даних з інтерактивною візуалізацією для ефективного розуміння, інтерпретації та прийняття рішень на основі надвеликих складених наборів даних. Для її реалізації доцільно використовувати не лише обчислювальні потужності комп’ютерів, а і враховувати людську здатність візуально сприймати закономірності та тенденції, щоб допомагати користувачам робити значущі висновки з даних, які є великими або складними, щоб їх можна було обробляти лише автоматизованими методами аналізу даних або візуалізацією. Основна ідея полягає в тому, щоб розробляти програмні

системи, які нададуть змогу фахівцям-аналітикам досліджувати масивні інформаційний потоки на потрібному рівні абстракції за допомогою відповідних візуальних представлень і вживати ефективних дій у режимі реального часу.

Візуальна аналітика – це мультидисциплінарний підхід, який використовує переваги різних суміжних галузей досліджень, таких як візуалізація, інтелектуальний аналіз даних, керування даними, об'єднання даних та статистичний аналіз [6]. По-перше, неоднорідні джерела даних необхідно обробляти та інтегрувати. Далі для створення візуалізації даних можна застосувати методи автоматизованого аналізу моделей вихідних даних. Ці моделі можна візуалізувати у подальшому. Окрім перевірки моделей, візуальні представлення можна абстрагувати від даних за допомогою методів інтерактивної візуалізації, які найкраще підходять для конкретного типу даних, структури та розмірності. У процесі візуальної аналітики можна отримати висновки за допомогою візуалізації, автоматичного аналізу, а також взаємодії між візуалізацією, моделями та фахівцями-аналітиками.

Цикл зворотного зв'язку передбачає зберігання знань, отриманих під час поглибленого аналізу системи, що допомагає аналітику швидше й якісніше формувати висновки в подальшій роботі. Важливим аспектом візуальної аналітики є взаємодія з користувачем. Аналітики повинні мати можливість формулювати запити та досліджувати дані, часто використовуючи кілька абстрактних подань. Швидкий зворотний зв'язок від цих подань дає змогу керувати процесом аналізу шляхом зміни параметрів або вибору альтернативних методів аналізу.

Основна мета аналізу візуальних даних полягає в представленні, навігації, агрегації та перегляді деталей таких складних даних.

Візуальна аналітика як загальний підхід потребує вдосконалення, що пояснюється наступними проблемами [7–11].

1. *Проблема масштабованості*. Створення масштабованої візуальної аналітики є нетривіальною задачею, оскільки масштабування має здійснюватися одночасно за різними критеріями (розміром, розмірністю, типами даних, рівнями якості тощо). Для вирішення цієї проблеми потрібні ефективні методи для роботи із вхідними даними високої роздільності за наявності шуму в даних, а також безперервним надходженням потоків даних зі значною кількістю інформації.
2. *Проблема невизначеності*. Ця проблема пов'язана з наявністю шуму та пропущених значень у даних, що надходять із різнорідних джерел, а також із спотвореннями, які можуть виникати внаслідок застосування методів автоматизованого аналізу та особливостей людського сприйняття. Для її розв'язання поняття якості даних і надійності алгоритмів аналізу повинні бути належним чином представлені аналітикам. У свою чергу, аналітики мають усвідомлювати наявність невизначеності та вміти оцінювати вхідні дані, зокрема їх повноту, точність і достовірність.
3. *Проблема апаратного забезпечення*. Більш ефективні та потужні обчислювальні методи потребують апаратного забезпечення для підтримки оброблення даних в режимі реального часу та для візуалізації великих потоків даних. Крім настільних дисплеїв високої роздільної здатності, передових пристроїв відображення, таких як екрани великої роздільної здатності, також мають підтримуватися портативні пристрої з відносно невеликими розмірами та малою роздільною здатністю. Системи візуальної аналітики повинні адаптуватися до характеристик доступних пристроїв виведення, що підтримують робочий процес візуальної аналітики на всіх рівнях.
4. *Проблема взаємодії*. Для забезпечення інтуїтивно зрозумілої візуальної роботи з поданням даних необхідні адекватні методи взаємодії між користувачем і системою. Аналітик повинен мати змогу зосередитися безпосередньо на розв'язанні аналітичного завдання, не

відволікаючись на надмірно складний або перевантажений користувацький інтерфейс. Ефективно спроєктовані інтерактивні механізми мають підтримувати навігацію та аналіз даних, сприяти фіксації проміжних висновків і забезпечувати прийняття обґрунтованих рішень.

5. *Проблема оцінювання*. Міждисциплінарна природа та складність візуального аналізу суттєво обмежують можливості уніфікованого оцінювання якості відповідних рішень. Це вимагає формування теоретичної бази для верифікації результативності та ергономічності нових аналітичних методів. Розроблення такої системи оцінювання дозволить не лише систематизувати підходи до аналізу користувацького досвіду, а й визначити вектори подальшого вдосконалення інструментарію візуалізації.
6. *Проблема інфраструктури*. Сучасний ринок візуальної аналітики представлений переважно вузькоспеціалізованими інфраструктурами, що розроблені під конкретні прикладні сценарії. Попри наявність механізмів інтеграції в окремих системах, існує розрив між наявнми платформами та вимогами до оперативності й точності оброблення даних. Це обґрунтовує необхідність переходу до архітектур вищого рівня абстракції, які б уніфікували сервіси та процеси для різних доменів. Крім того, критично важливим є формування репозиторіїв типових рішень, що дозволить реалізувати принцип повторного використання компонентів і знизити витрати на розробку.

Для візуалізації потоків даних часто використовуються різноманітні графіки, такі як, наприклад, лінійні, кругові, частотні, температурні та інші діаграми, що можуть відобразити зміну даних з часом [12]. Також для певних задач можуть використовуватися такі діаграми як водоспад, що являє собою історію зміни певного лінійного графіку.

Розглянемо застосування візуалізації для різних видів аналізу даних.

Аналіз просторово-часових даних передбачає роботу з даними, які містять обидва просторові та часові виміри, наприклад, телекомунікаційні дані дорожнього руху або дані супутникового зображення [11, 13]. Просторові дані містять координати разом із посиланнями на географічне розташування. Знаходження просторових відносин та моделі всередині даних представляє особливий інтерес, що вимагає розвитку відповідних методів керування, візуалізації та аналізу.

Серед важливих завдань у цій сфері загалом можна навести такі завдання, як виявлення закономірностей, тенденцій і кореляцій елементів даних з часом. Аналіз даних з як просторовими, так і часовими вимірами створює необхідність більш складних візуалізацій для відображення таких даних.

Аналіз даних радіочастотного спектру є складовою процедур радіомоніторингу та вимагає значного фахового досвіду в цій галузі. Робота оператора відповідних систем супроводжується істотним когнітивним навантаженням, що в окремих випадках може призводити до помилкового оцінювання сигналів і неточного розуміння поточної ситуації [14]. За умов ускладнення електромагнітного середовища аналітикам можуть бути необхідні додаткові джерела даних і спеціалізовані аналітичні засоби, спрямовані на підвищення рівня ситуаційної обізнаності та формування обґрунтованих висновків щодо стану електромагнітного спектра.

Сучасні підходи до радіомоніторингу розглядають аналіз радіочастотного спектра як задачу поєднання масштабованого збору вимірювань, потокового оброблення та інтерпретованої візуалізації результатів.

Ефективність такого аналізу значною мірою залежить від методів аналізу темпоральних даних. Дослідження користувацького досвіду візуалізацій часових рядів додатково підкреслюють важливість інтерактивності для зменшення когнітивного навантаження та підвищення якості інтерпретації. Практичні аспекти аналітики даних реального часу для

потокових даних, включно з вимогами до низької затримки та інкрементальності обчислень, узагальнено в роботах, присвячених аналізу потокових даних [15-19]. У різних прикладних доменах візуальна аналітика потокових темпоральних даних поєднує автоматизоване виявлення структур у часових рядах з інтерактивними методами їх інтерпретації. Дослідження демонструють ефективність таких підходів для аналізу даних руху, мережевого трафіку, комунікаційних потоків і геоприв'язаних даних, а також підкреслюють роль узгодженого подання часових подій і зменшення когнітивного навантаження під час аналізу [20–24].

Аналітикам необхідно постійно переглядати традиційні спектральні діаграми, наприклад, амплітудно-частотні та частотно-часові діаграми [12]. Моніторинг радіомереж потребує значного фахового досвіду в предметній галузі, а робота операторів радіоспектрального моніторингу зазвичай супроводжується високим когнітивним навантаженням. У повсякденній практиці це може призводити до помилкових оцінок сигналів і, як наслідок, до неточного розуміння поточної ситуації. Тому залучення додаткових джерел даних у процесі моніторингу радіомереж є об'єктивною тенденцією розвитку відповідних систем.

Дані радіосигналів являють собою структуровану інформацію, що формується на основі спектральних вимірювань, а в окремих випадках доповнюється даними послідовностей аналого-цифрового перетворення, отриманими внаслідок застосування складних процедур оброблення сигналів. Використання даних радіосигналів як додаткового джерела дає змогу попередньо ідентифікувати типи сигналів та їхні характеристики, зменшуючи вимоги до рівня спеціалізованого досвіду операторів і знижуючи когнітивне навантаження.

Водночас на практиці зберігається низка проблем. По-перше, унаслідок безперервного моніторингу один і той самий немиттєвий радіосигнал може багаторазово фіксуватися в потоці сигнальних даних. По-друге, традиційні спектральні діаграми не забезпечують адекватного

відображення частотно-часового розподілу та багатовимірних характеристик радіосигналів, що змінюються з часом. У зв'язку з цим виникає потреба в застосуванні більш розвинених аналітичних і візуальних засобів, які дають змогу узгоджено використовувати сигнальні та спектральні дані для аналізу складних електромагнітних ситуацій.

Аналіз потоків даних безпеки комп'ютерних систем. У багатьох практичних сценаріях кібербезпеки аналіз потоків даних є критичним для виявлення підозрілої поведінки, моніторингу стану інфраструктури та розуміння поточних дій у системі. До типових джерел належать системні й мережеві журнали (syslog, журнали подій ОС і застосунків), телеметрія маршрутизаторів і серверів, показники продуктивності та повідомлення засобів захисту [25]. Такі дані надходять безперервно, мають високу швидкість надходження, різні формати, тому завданням аналітики часто стає не лише «виявити інцидент», а й зменшити складність потоку, тобто виконати фільтрацію, нормалізацію, агрегування та виділити найбільш релевантні фрагменти для подальшого аналізу користувачем.

Аналітики можуть здійснювати моніторинг у реальному часі або працювати після інциденту, але в обох випадках повний перегляд усіх подій з однаковою деталізацією є практично неможливим. Подібні вимоги виникають і в суміжних прикладних ситуаціях (зокрема при роботі з потоками повідомлень із відкритих джерел), де важливо оперативно узгоджувати різнорідні сигнали для підтримки ситуаційної обізнаності. Для цього використовують підходи потокової аналітики, які дають змогу об'єднувати та агрегувати потоки, формувати узагальнені події, виділяти тренди, аномалії та потенційні джерела інцидентів, придатні для подальшої інтерпретації.

Попри значну кількість досліджень, повністю автоматизовані методи не гарантують достатньої надійності в реальних умовах експлуатації. Дані можуть бути неповними, зашумленими, містити суперечності, а поведінка атак і типова поведінка інфраструктури змінюються з часом. Тому візуальна

аналітика розглядається як спосіб поєднати алгоритмічне оброблення з експертним контролем. Автоматичні модулі зменшують інформаційне перевантаження і пропонують варіанти на розгляд, а аналітик зберігає контекст, формує гіпотези та оцінює достовірність висновків у межах задачі. У таких системах особливо важливо надавати вказівки користувачу до релевантних частин потоку, оскільки візуальні подання самі по собі можуть бути складними і вимагати інтерактивного дослідження (рис. 1.1).

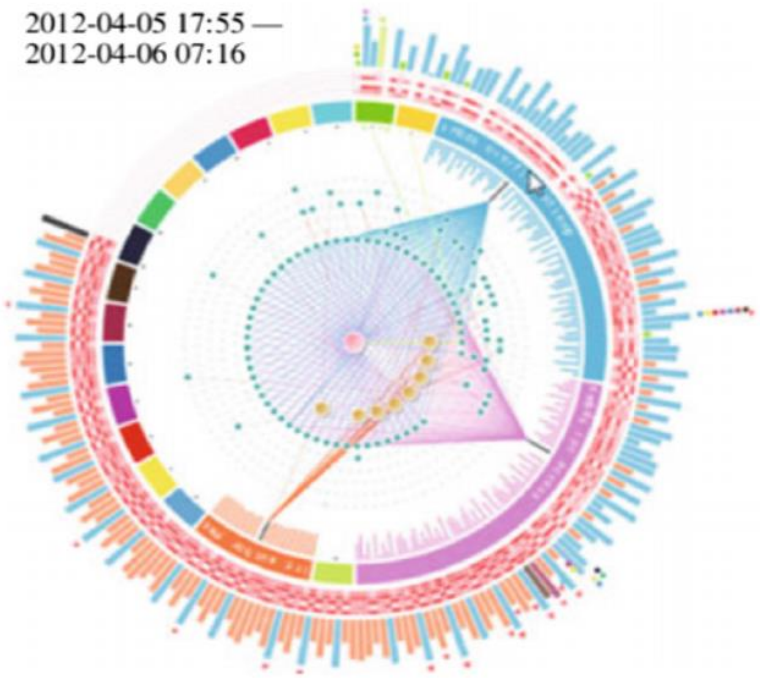


Рис. 1.1. Приклад кругової діаграми візуалізації даних кібербезпеки [26]

Також варто виділити алгоритмічний аналіз потоків даних, зокрема задачі виявлення подій (event detection) та кластеризацію потоків. Кінцева мета систем візуальної аналітики потоків полягає в підвищенні ситуаційної обізнаності та підтримці прийняття рішень, тобто сприйнятті елементів середовища в межах часу й простору, інтерпретації їхнього значення та прогнозуванні найближчих змін.

Кластерний аналіз у цій парадигмі розглядається як допоміжний інструмент візуальної аналітики. Він підтримує попереднє структурування

великих і різнорідних масивів подій, наприклад, групування схожих записів журналів, профілів мережевої активності або повторюваних шаблонів поведінки. Застосування кластеризації у зв'язці з візуалізацією звичайно дає змогу швидше виявляти закономірності та відхилення за рахунок наочних візуалізацій, виконувати сегментацію для детальнішого аналізу окремих груп. Також дає змогу підтримувати інтерактивні сценарії, коли користувач змінює параметри кластеризації і спостерігає, як це впливає на структуру даних та видимість потенційних аномалій [27–32].

Розглянемо особливості візуалізації даних пристроїв IoT. Для сучасних підприємств візуалізація даних IoT відкриває широкі можливості для підвищення ефективності операцій, оптимізації процесів і прийняття рішень на основі реальних даних, зокрема в інформаційних та кіберфізичних системах. Вона дає змогу контролювати ключові показники в режимі реального часу, оперативно виявляти потенційні проблеми та використовувати приховані можливості. IoT надає змогу створювати більш інформовані стратегії, адаптовані до конкретних потреб і обставин. Це включає можливість моніторингу продуктивності обладнання, аналізу стану виробничих процесів та виявлення закономірностей поведінки споживачів, які раніше не були очевидними.

Розвиток візуалізації IoT-даних пов'язаний із переходом від поодиноких графіків до інтерактивних систем, у яких візуальне подання поєднується з попереднім обробленням, агрегуванням і механізмами взаємодії з потоком даних. З огляду на безперервність надходження та різнорідність параметрів, такі системи мають забезпечувати як оперативний моніторинг, так і можливість дослідницького аналізу та уточнення гіпотез. У сучасних підходах особливу увагу приділяють гнучким фреймворкам і технологіям візуалізації, де важливими є придатність до практичного впровадження й масштабованість [40–44].

Одним із найважливіших інструментів для візуалізації даних IoT є дашборди (інтерактивні панелі керування), які виступають у ролі цифрових

центрів контролю та керування. Вони дають змогу в реальному часі відображати ключові метрики у зручному, зрозумілому для користувача форматі. Візуалізовані дані не лише сприяють кращому розумінню стану систем, але й дають змогу оперативно реагувати на зміни та запобігати потенційним проблемам, таким як марнування ресурсів чи вихід з ладу обладнання.

Техніки візуалізації даних IoT дають змогу виявляти закономірності, тенденції та аномалії, які були б важкодоступними для аналізу в їхньому «сирому» вигляді [45].

Попри значний прикладний потенціал, візуалізація даних IoT супроводжується низкою взаємопов'язаних особливостей, які ускладнюють отримання коректних і своєчасних аналітичних висновків.

Передусім, існує *проблема масштабу*. Велика кількість пристроїв генерує безперервні потоки подій, що швидко перевищують можливості традиційних засобів відображення. У таких умовах візуальний шум зростає, важливі деталі можуть бути загублені й пропущені у масиві точок, а аналітичний процес потребує спеціальних підходів до зменшення когнітивного навантаження та збереження інформативності навіть за високих швидкостей надходження даних [12].

Не менш суттєвою є *різнорідність IoT-даних*. У реальних системах одночасно співіснують числові ряди, категоріальні та текстові атрибути, геопросторові координати, сигнали, зображення або відео, що надходять від сенсорів різної природи та точності. Отже, візуалізація має не просто відобразити окремі канали, а забезпечити цілісне уявлення про стан системи, узгодивши часові шкали, семантику вимірювань і контекст походження даних.

Додаткову складність створює *якість даних*. Сенсорні вимірювання часто містять пропуски, збої, дрейф, дублікати або аномальні значення, які без попереднього оброблення здатні призводити до хибних візуальних патернів і, як наслідок, до помилкових управлінських рішень. У цьому

контексті особливо важливо враховувати, що неякісні дані можуть бути підсилені автоматизованими методами аналізу. Некоректно налаштовані алгоритми фільтрації, згладжування, класифікації чи виявлення аномалій здатні спотворювати висновки, навіть якщо сама візуальна форма виглядає переконливо.

Поряд із технічними аспектами постає питання коректного добору *способу візуалізації*. Різні типи даних та аналітичні завдання потребують різних візуальних відображень. Те, що добре працює для порівняння категорій, може бути непридатним для аналізу трендів або взаємозв'язків. Невідповідність між типами й походженням даних і обраною технікою відображення підвищує ризик неправильного тлумачення, особливо коли користувачеві потрібно діяти швидко й на основі обмеженого простору відображення на екрані.

Також існує проблема *ступеню спрощення* і зберігання точності. Візуалізація абстрагує деталі, однак надмірна агрегація або агресивне згладжування здатні приховати важливі локальні події (наприклад, короткі збої, пікові навантаження чи рідкісні аномалії). Отже потрібні механізми, які дають змогу зберігати оглядовість без втрати можливості переглянути первинні дані і перевірити джерело.

У випадку систем реального часу ці питання підсилюються *вимогами продуктивності*. Висока частота оновлення, значні обсяги даних що надійшли, складні візуалізації та обмеження апаратного забезпечення створюють ситуацію, коли затримки оброблення або рендерингу впливають не лише на кількість кадрів на секунду, а й на коректність сприйняття поточного стану. Саме тому управління обсягом і швидкістю доступу до даних стає невід'ємною частиною візуальної аналітики IoT.

Розглянемо найбільш розповсюджені способи підвищення продуктивності. Варто зазначити, що підвищення продуктивності звичайно спирається на комбінацію стратегій. Застосовується агрегація, яка узагальнює дані у більші часові інтервали, зменшуючи кількість елементів

для відображення та водночас підвищуючи читабельність. Важливу роль відіграє попереднє оброблення, а саме очищення, усунення пропусків, нормалізація та перетворення у формат, більш придатний для подальшої візуалізації, дають змогу зменшити частку артефактів і підвищити релевантність показаного. Стиснення зменшує витрати на передавання і зберігання, а отже, опосередковано знижує затримку відображення даних.

Поряд із цим широко використовуються прийоми прискорення доступу. Кешування зберігає результати проміжних обчислень і повторно використовує їх під час оновлень, тоді як індексація скорочує час вибірки потрібних фрагментів без повного сканування наборів даних.

1.2. Критерії оцінювання візуалізації даних IoT

Оцінювання візуалізації даних у системах Інтернету речей є важливим етапом, що дозволяє оцінити аналітичну цінність представлених даних для прийняття обґрунтованих управлінських рішень. Для верифікації якості функціонування програмного забезпечення реального часу застосовується комплексний підхід, що базується на низці ключових критеріїв [46].

Одним із таких критеріїв є *продуктивність* програмного забезпечення, яка визначається можливістю оброблювати потоки даних без значних часових затримок та кількість даних, які вони здатні опрацювати за одиницю часу. Зазначені характеристики безпосередньо визначають можливість ефективного вилучення інформації з візуальних форм, оскільки специфіка систем реального часу вимагає максимально швидкої доставки вичерпних відомостей до операційного інтерфейсу. У такому технологічному контексті сумарна затримка процесів опрацювання та рендерингу не має перевищувати 100 мс, що є критичним для забезпечення цілісного та плавного зорового сприйняття динамічних змін. Водночас пропускна здатність візуалізаційного конвеєра повинна бути узгоджена з піковою інтенсивністю джерел даних (щонайменше понад 1000 подій на секунду), що дозволяє підтримувати стабільність відображення та

запобігати виникненню черг у буферах даних.

Для підходів, орієнтованих на зменшення обсягу даних є важливим *збереження візуальної цілісності відображення*, доцільним є впровадження оцінювання на підставі порівняльного аналізу схожості зображень. Допустимим порогом збереження інформативності при застосуванні алгоритмів ущільнення визначено показник подібності не нижче 90% відносно еталонного кадру, сформованого на базі повного масиву даних. Це гарантує оператору можливість ідентифікації аномалій та трендів з імовірністю, еквівалентною роботі з необробленими потоками, одночасно досягаючи оптимізації навантаження на обчислювальні ресурси. У межах даної процедури здійснюється співставлення референтних кадрів візуалізації та результатів застосування методів узагальнення за умови ідентичності параметрів рендерингу. Зокрема, фіксація масштабів, товщини ліній, конфігурацій згладжування та кольорових палітр є обов'язковою, оскільки це дозволяє відокремити безпосередні інформаційні втрати від технічних артефактів візуалізаційного процесу. Збереження стабільності відображення забезпечує зменшення когнітивного навантаження [47].

Суттєвим критерієм також виступає *рівень споживання апаратних ресурсів*. Фундаментальним етапом оцінювання є проведення бенчмаркінгу та детальне вимірювання продуктивності візуалізаційного конвеєра. У межах аналізу досліджуються такі параметри, як пропускна спроможність, затримка оброблення та рендерингу, стабільність тривалості кадру (фреймтайм), обсяг задіяної оперативної пам'яті, а також завантаженість CPU та GPU. Крім того, вивчається чутливість цих індикаторів до варіацій системних налаштувань, кількості джерел даних, інтенсивності вхідних подій та обраної глибини генералізації інформації. Позитивно оцінити роботу можливо при здатності системи підтримувати стабільну частоту кадрів на рівні 30-60 FPS при зростанні кількості IoT-пристроїв. При цьому навантаження на центральний процесор має залишатися в межах 20-30% для типових сценаріїв. Для забезпечення валідності порівняльного аналізу при

бенчмаркінгу необхідно застосовувати детерміновані набори даних, уніфіковані апаратні конфігурації. Обов'язковими є процедури попереднього «прогріву» системи. Оскільки реальні системи чутливі до конфігурації й апаратного середовища, результати слід подавати разом із параметрами тестового стенду, характеристиками CPU та GPU, обсягом оперативної пам’яті, версіями драйверів і бібліотек, а також із фіксованими налаштуваннями візуального подання та режимами прискорення.

Важливим є критерій наявності *логічного групування* потоків даних. У контексті IoT здатність системи об’єднувати дані дозволяє оператору ефективно отримувати логічно поєднані дані, що відбулися в один момент часу. Це передбачає не лише створення статичних груп, а й динамічну агрегацію показників у межах цих груп, що мінімізує інформаційну перенасиченість і полегшує перехід від загального огляду системи по окремим параметрам.

Важливим параметром є *різноманітність та адаптивність типів відображення*. Сучасні системи повинні виходити за межі стандартних двомірних графіків, пропонуючи спеціалізовані віджети для різних типів даних від теплових карт і просторових шарів на ГІС-платформах до складних 3D-моделей або спектрограм. Гнучкість у виборі методу представлення дозволяє адаптувати інтерфейс під специфіку конкретного фізичного процесу, що безпосередньо впливає на можливість інтерпретації даних оператором для приняття рішень.

Критерій *підтримки роботи в реальному часі* оцінюється через наявність механізмів потокової трансляції даних і їх подальшої візуалізації без необхідності повного оновлення сторінки чи виконання дискретних запитів. Це реалізується на базі технологій, що забезпечують безперервне надходження подій з мінімальною латентністю та технологій, що забезпечують візуалізацію даних у реальному часі.

Також важливою є можливість *роботи з мультимодальними даними*, яка характеризує здатність системи одночасно оперувати різнорідними

потоками (відео, аудіо, сенсорною телеметрією тощо).

Загалом доцільність застосування конкретних критеріїв залежить від об’єкту оцінки. Наприклад для оцінювання методів, що мають за мету оптимізацію навантаження, доцільно буде застосувати критерії споживання апаратних ресурсів.

1.3. Аналіз вимог до програмного забезпечення візуалізації та порівняльний аналіз відомих програмних рішень

В результаті проведеного аналізу вимог до програмного забезпечення візуалізації було виділено критерії, представлені в табл. 1.1.

Таблиця 1.1. – Критерії оцінювання візуалізації

Критерій оцінювання	Технічні вимоги та функціональні особливості	Цільові показники / Метрики
1	2	3
Пропускна здатність оброблення	Здатність візуалізаційного конвеєра опрацьовувати потоки без інерційних черг у буферах	Затримка менше 100 мс, Пропускна здатність > 1000 подій/с
Візуальна цілісність	Схожість зображення після даунсемплінгу порівняно з еталонним кадром (повний набір даних)	Показник подібності ≥ 90% при ідентичних параметрах рендерингу
Ресурсоємкість	Ефективність використання апаратних потужностей при масштабуванні кількості пристроїв та частоти надходженя даних	Навантаження CPU до 20-30%, частота кадрів 30-60 FPS
Підтримка роботи в реальному часі	Потокова трансляція та оновлення інтерфейсу	Використання технологій потокової передачі, відсутність дискретних запитів для оновлення

Продовження табл. 1.1

1	2	3
Адаптивність відображення	Різноманітність типів представлення даних	Наявність 2D/3D графіків, ГІС-шари, теплові карти, спектрограми
Затримка відображення	Здатність до швидкого оновлення відображення	Для відображення в реальному часі затримка має складати < 100мс
Групування потоків даних	Наявність механізмів об'єднання потоків	Підтримка оброблення потоків не менше ніж 20 пристроїв та не менше ніж 1000 даних на секунду
Мультимодальність	Здатність одночасно оброблювати та синхронізувати різнорідні типи вхідних даних	оОброблення та часова кореляція відео, аудіо, телеметрії тощо

Розглянемо приклади існуючих програмних рішень візуалізації даних IoT. Зокрема, Tableau і Power BI використовуються для побудови інтерактивних панелей, де результати кластеризації можуть бути представлені через фільтри.

Платформа *Tableau* має можливості створення дашбордів та підтримку значних обсягів даних. Однак Tableau має суттєві обмеження у роботі з даними реального часу. Стандартні методи оновлення орієнтовані на періодичні отримання даних, а часте автоматичне оновлення створює надмірне навантаження на сервер. Стандартні методи оновлення в Tableau базуються на механізмі періодичного опитування джерел даних, що вносить неминучу дискретність у процес візуалізації. У контексті систем реального часу, де затримка не повинна перевищувати 100 мс, такий підхід виявляється неприпустимим, оскільки навіть при мінімально можливих інтервалах оновлення виникає значна латентність між моментом генерації події на IoT-пристрої та її відображенням на екрані. Спроби реалізувати

псевдореальний час через часте автоматичне оновлення візуалізацій призводять до зростання навантаження на серверний компонент та мережеву інфраструктуру. Крім того, графічний конвеєр Tableau не забезпечує нативної підтримки мультимодальності на рівні потокової синхронізації. Платформа не має вбудованих інструментів для вирівнювання часових міток між медіаданими та сенсорною телеметрією в реальному часі, що критично для достовірної інтерпретації подій. Також відсутня підтримка низькорівневих протоколів передачі даних, таких як MQTT. Таким чином, Tableau демонструє високу ефективність для ретроспективного аналізу великих даних, проте її використання для візуальної аналітики високочастотних мультимодальних IoT-потоків в реальному часі обмежене.

Power BI має інтеграцію з екосистемою Microsoft та, відповідно, доступність для користувачів, що вже використовують дану екосистему для отримання та оброблення даних від пристроїв. Однак, дана система має обмежені можливості візуалізації, оскільки для використання всіх можливостей частота оновлення обмежена, а відображення даних у реальному часі має обмежені можливості візуалізації. Для веборієнтованих рішень подібні сценарії реалізовують засобами бібліотек на кшталт D3.js, що дає змогу створювати кастомні інтерактивні подання кластерів і пов’язаних подій. У середовищах прототипування аналітичних конвеєрів і експериментів з методами групування також застосовують KNIME та Orange, які підтримують побудову робочих процесів і візуальне дослідження результатів кластеризації [33–37].

Grafana є однією з найбільш популярних платформ з відкритим кодом для візуалізації метрик та часових рядів. Її архітектура орієнтована на багатоджерельність, що дозволяє інтегрувати дані з Prometheus, InfluxDB, Elasticsearch, а також реляційних баз даних, таких як MySQL або PostgreSQL. Ключовою перевагою системи є механізм створення дашбордів із використанням різноманітних панелей. Наприклад графіків, теплових

карт, гістограм, геокарт тощо. Grafana підтримує горизонтальне масштабування та кластеризацію, що забезпечує роботу системи візуалізації при підвищенні навантаження. Також підтримує розширення можливостей візуалізації за допомогою плагінів.

Kibana, що є частиною Elastic Stack, розроблена переважно для дослідження логів та подій, індексованих в Elasticsearch. Вона демонструє значні результати у повнотекстовому пошуку та аналізі неструктурованих даних. На відміну від Grafana, Kibana пропонує глибшу інтеграцію з інструментами машинного навчання для виявлення аномалій безпосередньо в потоках даних. Основним архітектурним обмеженням Kibana є її жорстка прив'язка до Elasticsearch. Для візуалізації даних з інших джерел їх необхідно попередньо імпортувати в Elastic Stack, що вносить додаткові затримки [38].

ThingsBoard позиціонується як універсальна платформа для управління пристроями та збору даних в екосистемах IoT. Вона надає розвинені можливості управління життєвим циклом пристроїв, підтримку протоколів MQTT, CoAP, HTTP та побудову складних логічних ланцюжків оброблення через Rule Engine. Візуалізація в ThingsBoard реалізована через систему віджетів, які можуть відображати дані в режимі реального часу або історичні значення. Однак, при роботі з високочастотною телеметрією віджети можуть не встигати коректно оброблювати чергу вхідних повідомлень. Крім того, спеціалізовані віджети в ThingsBoard можуть мати артефакти автооновлення, що змушує розробників шукати альтернативні способи стабілізації відображення, які не передбачені стандартною архітектурою [39].

Також важливо розглянути можливість візуалізації даних з використанням спеціалізованих графічних бібліотек комерційного рівня, серед яких однією з найбільш технологічно розвинутих є *SciChart*. На відміну від комплексних платформ моніторингу, SciChart функціонує як високорівневий набір засобів розроблення, що зосереджений виключно на

оптимізації рендерингу екстремально великих обсягів даних. Архітектурною основою даної бібліотеки є власний кросплатформений графічний рушій Visual Xccelerator, який максимально використовує можливості апаратного прискорення через низькорівневі API, такі як DirectX, OpenGL та WebGL. Це дозволяє досягати значної швидкодії, де оброблення мільйонів точок даних відбувається без втрати плавності інтерфейсу, що є критично важливим для систем реального часу. Бібліотека демонструє стабільну частоту оновлення екрана на рівні шістдесяти кадрів на секунду навіть при значній інтенсивності вхідного потоку, що значно перевищує встановлений поріг у 1000 подій за секунду. Також SciChart забезпечує мінімальну латентність відображення, яка значно нижча за межу у 100 мс. Однак, незважаючи на досконаліть графічного підсистеми, SciChart залишається виключно шаром представлення і не містить вбудованих засобів для розв'язання проблем вищого рівня архітектури. Вона не забезпечує нативної підтримки протоколів передачі даних, таких як MQTT 5.0, та не має готових механізмів для автоматичної синхронізації мультимодальних потоків, наприклад, вирівнювання часових міток відеоряду та телеметрії, що покладається на розробника кінцевого програмного забезпечення.

Сучасний стан програмного забезпечення для візуалізації потокових даних характеризується наявністю загальних платформ та систем моніторингу, таких як Grafana, Power BI, Kibana, ThingsBoard тощо та окремих бібліотек візуалізації значних потоків даних, таких як SciChart (табл. 1.2.). Аналіз архітектурних особливостей Grafana дозволяє стверджувати, що незважаючи на наявний інструментарій для відображення часових рядів, система має обмежені можливості щодо забезпечення відображення в режимі реального часу. У свою чергу, платформа Kibana, орієнтована на аналіз логів у середовищі Elasticsearch, демонструє низьку візуальну стабільність через механізми повного оновлення дашбордів, що спричиняє ефект мерехтіння інтерфейсу та ускладнює роботу операторів у

сценаріях тривалого моніторингу. Спеціалізовані IoT-платформи, зокрема ThingsBoard стикаються з проблемами масштабування при роботі з потоками даних, що потребують складної часової синхронізації безпосередньо на стороні клієнта.

Таблиця 1.2. – Аналіз існуючих програмних рішень

Критерій порівняння	Tableau	Power BI	Grafana	Kibana	ThingsBoard	SciChart
Протоколи даних	SQL, API	REST Azure Hub	TSDB	Elastic search	MQTT, CoAP, HTTP	Native Array
Затримка відображення	Секунди (>1с)	Секунди (>1с)	Мілісекунди (<1с)	Секунди (>1с)	< 100 мс	< 16 мс (60 FPS)
Пропускна здатність оброблення	Низька	1 млн рядків/год	Висока	Дуже висока	До 20 тисяч поівідомлень с	1 млрд точок
Групування потоків даних	Відсутнє	Відсутнє	Відсутнє	Відсутнє	Відсутнє	Рівень додатка
Мультимодальність	Відсутня	Обмежена	Через плагіни	Через додатки	Розширена (Custom Widgets)	-
Підтримка реального часу	Відсутня	Обмежена	Наявна	Обмежена	Повна	Повна
Ресурсоємкість	Висока	Висока	Низька	Висока	Низька	Низька

За результатами проведеного аналізу встановлено, що ThingsBoard є найбільш пристосованою до наведених вимог системою серед розглянутих систем для цілей комплексної візуалізації та управління даними IoT-пристроїв. При цьому варто зазначити, що для задач реалізації високпродуктивного відображеня даних у реальному часі застосування SciChart може надати кращі результати по швидкодії. Хоча платформа ThingsBoard наразі є одним із найбільш функціональних та універсальних

рішень для управління пристроями в екосистемах IoT, вона все ж має низку суттєвих обмежень, які не дають змогу повною мірою вирішити специфічні задачі візуалізації високоінтенсивних потоків даних. Зокрема, при роботі з високочастотною телеметрією стандартні віджети платформи можуть не встигати коректно опрацьовувати чергу вхідних повідомлень. Крім того, під час відображення динамічних даних у спеціалізованих віджетах можуть виникати артефакти автооновлення, що змушує розробників шукати сторонні методи стабілізації зображення, не передбачені стандартною архітектурою системи. Варто зауважити, що у проаналізованих програмних рішень не знайдено таких особливостей, як стабільність візуалізації попередньо відображених даних та подієво-орієнтованих інтерфейсів.

Таким чином, незважаючи на те, що ThingsBoard переважає багато аналогів за рівнем інтеграції та підтримки протоколів, ця платформа не забезпечує необхідної візуальної стабільності та продуктивності при екстремальних навантаженнях, що підтверджує актуальність розроблення нових методів та архітектурних рішень для візуалізації мультимодальних темпоральних потокових даних.

1.4. Висновки до першого розділу

У першому розділі розглянуто сучасний стан програмного забезпечення для візуалізації даних, наведено особливості візуальної аналітики у різних галузях застосування. Зауважено, що вибір типу подання визначається не лише форматом даних, а й аналітичним завданням, рівнем деталізації та вимогами до взаємодії з користувачем. Розглянуто використання кластерного аналізу у візуальній аналітиці.

Показано, що стрімке зростання кількості джерел інформації спричиняє задачі, пов’язані із високою динамічністю, гетерогенністю форматів та необхідністю синхронізації потоків мультимодальних даних у реальному часі. Встановлено, що візуалізація в таких системах є критично важливим інструментом для оперативної підтримки прийняття рішень. У

ході роботи було проаналізовано та обґрунтовано комплекс вимог до якості візуалізації, що охоплюють показники продуктивності, візуальної цілісності та ресурсоємності. Виявлено наявні проблеми та завдання для візуалізацій та показано доцільність та необхідність розроблення методів, що можуть бути використані для візуалізації потокових даних пристроїв IoT, а також запропоновано напрямки розроблення таких методів.

Проведено порівняльний аналіз сучасних програмних засобів та аналітичних платформ. Виявлено, що навіть найбільш розвинені системи мають недоліки при роботі з високочастотними мультимодальними даними. Виявлені обмеження наявних засобів обґрунтовують необхідність подальшого розроблення нових методів та архітектурних рішень для створення спеціалізованого програмного забезпечення, здатного ефективно працювати з темпоральними мультимодальними потоковими даними.

РОЗДІЛ 2. РОЗРОБЛЕННЯ МЕТОДУ ПОДІЄ-ОРІЄНТОВАНОГО ГРУПУВАННЯ ПОТОКОВИХ ДАНИХ ПРИСТРОЇВ ІОТ

2.1. Задача групування потоків даних та шляхи її розв'язання

Розглянемо основні поняття, що є важливими для вирішення задачі групування потоків даних.

Потік даних – це послідовність незмінних записів, до якої постійно додаються нові дані. Кожен запис представлений у вигляді пари «ключ-значення», доповненої часовою міткою. Пари «ключ-значення» дають змогу здійснювати паралельне оброблення даних, оскільки записи з різними ключами можуть оброблятися одночасно. Ця концепція є основою для створення високонавантажених та масштабованих додатків для оброблення потоків даних.

Темпоральність потоків даних означає, що кожен елемент потоку має невід'ємну часову характеристику, тобто часову мітку. Темпоральна природа таких потоків вимагає систему враховувати не лише значення вимірювань, а й їхню позицію в часі, що є важливим для коректного оброблення та візуалізації потоків у режимі реального часу.

Багатопотоковими є дані, які надходять одночасно з кількох незалежних джерел, утворюючи множину паралельних потоків з можливими різними частотами надходження.

Групування потоків даних IoT у реальному часі – це процес, важливий для оброблення, аналізу та організації значних і безперервних потоків даних, що генеруються пристроями IoT. До таких пристроїв належать давачі, камери, носимі пристрої та інші пристрої, які постійно передають дані в реальному часі. Групування цих потоків даних необхідне для реалізації оброблення багатьох потоків даних, спрощення аналізу та, у результаті, отримання корисної інформації від даних, що надійшли.

Розглянемо основні аспекти, пов'язані зі задачею групування потоків даних IoT у реальному часі.

Агрегація та пакетування потоків. Першим кроком у процесі групування потоків даних IoT є їх агрегація або пакетування. Оскільки пристрої IoT часто передають постійний потік даних, оброблення кожного окремого повідомлення може бути неефективним. Пакетування даних за часовими вікнами (наприклад, щосекунди або щохвилини) допомагає зменшити навантаження на системи, зберігаючи при цьому режим реального часу. Ці згруповані потоки можуть оброблятися паралельно, що підвищує масштабованість системи.

Просторове та часове групування. Пристрої IoT часто розташовані в різних географічних місцях, і групування даних за географічною близькістю є корисним. Наприклад, дані з температурних давачів в одному регіоні можна згрупувати для аналізу місцевого клімату. Групування потоків даних за часовими інтервалами також є поширеним підходом. Наприклад, вимірювання, зроблені протягом 10-хвилинного періоду, можуть бути згруповані для аналізу трендів у реальному часі.

Контекстне групування. Для групування потоків даних можна використовувати контекстуальну інформацію. Наприклад, у системі «розумного дому» потоки даних з давачів температури, вологості та освітлення можуть бути згруповані відповідно до контексту кімнати. Це дає змогу проводити інтелектуальний аналіз, виявляючи шаблони, що стосуються певного середовища.

Групування за типом пристрою. Потоки даних можуть бути згруповані за типами пристроїв або давачів. Наприклад, у промисловому середовищі потоки від давачів вібрації можуть бути згруповані окремо від температурних датчиків. Це оптимізує оброблення, оскільки різні потоки можуть вимагати різних видів аналізу.

Групування за подіями. У деяких додатках IoT потоки даних групуються на основі подій. Наприклад, у «розумному місті» потоки з давачів руху, метеостанцій і систем громадського транспорту можуть бути

згруповані, коли відбувається значуща подія, така як затор на дорозі. Таке групування дає змогу швидко реагувати на критичні ситуації [48].

Розподіл для балансування навантаження. У великих системах IoT необхідно розподіляти потоки даних між кількома серверами або вузлами оброблення для балансування навантаження. Групування через розподіл допомагає рівномірно розподілити навантаження, що запобігає утворенню «вузьких місць» у системі. Це особливо важливо для систем реального часу, де затримки критичні.

Питання безпеки та конфіденційності. Групування потоків даних IoT у реальному часі також передбачає вирішення питань безпеки та конфіденційності. Деякі дані можуть бути конфіденційними, що вимагає їхнього шифрування або ізоляції від інших потоків.

При цьому постають наступні основні проблеми.

1. *Великий обсяг і швидкість*. Пристрої IoT генерують значні обсяги даних на високій швидкості, що ускладнює їхнє групування та оброблення в реальному часі.
2. *Гетерогенність даних*. Пристрої IoT генерують різні типи даних (числові, категоріальні, відео, аудіо), що ускладнює їхнє групування.
3. *Масштабованість*. Збільшення кількості пристроїв IoT вимагає масштабованих рішень для групування й оброблення потоків даних у реальному часі.

У контексті потокового оброблення даних і аналітики в реальному часі, важливим концептом, який уможливлює групування та оброблення нескінченних потоків даних протягом певного періоду часу, є поняття вікна – кінцевої підмножини подій із нескінченного потоку даних, що задається часом початку та часом кінця вікна.

Застосування вікон дає змогу розділити безперервні потоки на менші, більш керовані частини для аналізу, агрегації або інших операцій. Існує кілька типів вікон, що використовуються в обробленні потоків, включаючи

ковзаючі вікна (hopping windows), які забезпечують гнучкий спосіб управління даними на основі їхньої темпоральної ознаки.

Оскільки дані в реальних потокових застосунках звичайно є безперервними і не мають природного завершення, вікна дають можливість виконувати обчислення або агрегацію на підмножинах потоку протягом певного періоду часу. Це допомагає відповідати на запити виду: «Яка загальна кількість подій за останні п’ять хвилин?» або «Яка середня температура за останню годину?».

Розрізняють наступні типи вікон, які часто використовуються в обробленні потоків [49].

1. *Постійні вікна* (Tumbling Windows). Це вікна фіксованого розміру, що не перекривають одне одного. Як тільки одне вікно завершується, відразу починається наступне. Кожен запис у потоці належить лише одному вікну.
2. *Ковзні вікна* (Sliding Windows). Це вікна, що перекриваються, в яких нові вікна створюються через регулярні інтервали. Один і той же запис може потрапити в кілька вікон.
3. *Крокуючі вікна* (Hopping Windows). Це особливий тип ковзних вікон, де вікна «крокують» вперед через фіксований інтервал, створюючи визначений ступінь перекриття між послідовними вікнами. Крокуюче вікно визначається двома параметрами:
   - розмір вікна, тобто його тривалість (наприклад, 5 хвилин);
   - розмір кроку (або інтервал кроку) – інтервал, через який починається нове вікно (наприклад, щохвилини).

У крокуючих вікнах кілька вікон можуть перекриватися, що означає, що одна подія може бути включена в кілька вікон. Перекриття між вікнами контролюється розміром кроку. Якщо крок менший за розмір вікна, вікна будуть перекриватися. Якщо крок дорівнює розміру вікна, це буде як постійне вікно.

Однією з важливих проблем є часова синхронізація між пристроями в мережі. Існують різноманітні механізми синхронізації – найбільш відомим стандартом є Network Time Protocol (NTP), який дає змогу синхронізувати годинники в межах кількох мілісекунд у глобальних мережах [50]. Для промислових систем з високими вимогами до точності застосовують Precision Time Protocol, який забезпечує мікросекундну синхронізацію, але потребує спеціалізованого обладнання та підтримки в мережевих комутаторах.

Виділяють кілька джерел проблем часової синхронізації [49, 50]. Давачі часто мають неточні генератори тактових сигналів, що спричиняє дрейф годинника та розходження часових міток між пристроями, тому необхідно задавати допустиму часову похибку. Транспортні мережі (Wi-Fi, 4G/5G тощо) створюють джитер, тобто змінну затримку доставки пакетів, через що події, що відбулися майже одночасно, надходять у довільному порядку. Наявність дрейфу годинників і мережевих затримок призводить до того, що події, які належать до однієї групи, можуть опинитися поза межами допустимого вікна синхронізації. Якщо поріг малий – групи будуть неповними, якщо великий – зросте ризик помилкового поєднання подій, що відносяться до різних процесів.

Виходячи з вищезазначеного аналізу проблем, можна виділити наступні вимоги до процедур групування потоків даних [51-55].

Багаторівневе групування. Ця вимога передбачає ієрархічне групування, тобто спочатку дані з окремих сенсорів агрегуються в групи, які, у свою чергу, можна згрупувати в більші групи і так далі.

Переконфігурація ієрархії під час виконання програми. Системи для оброблення потоків часто повинні залишатися працездатними під час зміни наявних пристроїв. Можливість зміни під час виконання дає змогу уникнути простоїв системи під час змін у структурі потоків або логіці оброблення за рахунок відсутності необхідності перезапускати систему.

Збереження масштабованості та надійності. Оброблення потоків звичайно використовується для великих обсягів даних, відповідно навантаження повинно розподілятися між кількома ядрами процесора або навіть на кілька обчислювальних вузлів. Тому метод, який виконує групування потоків, повинен забезпечувати масштабованість і швидкодію, щоб не стати «вузьким місцем» загального процесу оброблення даних. У середовищі IoT з тисячами давачів, які генерують дані, модуль групування повинен бути здатен обробляти це навантаження без значного зниження продуктивності при збільшенні кількості пристроїв.

Реалізація аналітики в реальному часі для чутливих до часу застосунків. Деякі IoT-системи вимагають аналізу даних у реальному часі, щоб забезпечити своєчасну реакцію на події. Наприклад, системи моніторингу здоров'я пацієнтів або автоматизоване управління промисловими процесами потребують швидкого оброблення даних і прийняття рішень. Для цього рішення повинно підтримувати оброблення потоків у реальному часі з низькою затримкою.

Оброблення невизначеності та прийняття рішень у випадку відсутності даних. Реальні IoT-системи часто мають справу з неповними, недостовірними або невизначеними даними. Події з фізичного світу можуть бути непередбачуваними або містити певний рівень похибки. Тому рішення має включати механізми для оброблення цієї невизначеності.

Вимоги до процедур групування потоків даних IoT у реальному часі підкреслюють необхідність у гнучких та масштабованих рішеннях. Вони мають бути здатні обробляти великі обсяги сенсорних даних, підтримуючи кілька ієрархій та можливість переконфігурації під час виконання.

2.2. Метод групування темпоральних мультимодальних даних

Метою розроблення методу групування темпоральних мультимодальних даних є спрощення процесів оброблення даних, що надходять з пристроїв IoT.

Метод реалізує подієво-орієнтований підхід виділення подій з мультимодальних даних, що ґрунтується на принципі трансформації безперервних або високочастотних інформаційних потоків у послідовність точкових даних з інформацією про час події. Оскільки кожна окрема модальність, зокрема відео, аудіо чи телеметрія, характеризується власною специфікою та частотою дискретизації, їх безпосереднє поєднання створює надмірне обчислювальне навантаження на систему. Застосування подієвого підходу дозволяє уніфікувати ці гетерогенні дані, представляючи їх як окремі значущі події на єдиній часовій осі.

Процес такої трансформації базується на аналізі вхідних потоків за допомогою спеціалізованих функцій детекції суттєвих змін. Під час оброблення аудіосигналу в реальному часі можливо фіксувати моменти перевищення заданого порогу інтенсивності або виявлення характерних частотних характеристик, що дозволяє ігнорувати фоновий шум і генерувати подію лише для критичних фрагментів. При роботі з відеопотоками замість оброблення кожного кадру можливо використовувати детекцію змін сцени або виявлення руху, де подією вважається часова мітка кадру, на якому зафіксовано появу об'єкта або іншу динаміку контенту. Для сенсорної телеметрії подія ініціюється у випадках детекції аномалій або перетину встановлених порогових значень числових параметрів, таких як температура чи тиск. Загалом пропонується використовувати відомі методи виділення події з мультимодальних даних. Таким чином, в подальшому можливо оперувати послідовностями даних у вигляді окремих подій.

Розроблений метод групування передбачає застосування реактивного програмування, зокрема, оператора GroupJoin. Спершу розглянемо його простіший варіант, а саме оператор Join, приклад роботи якого зображений на рис. 2.1 та продемонстрований у лістингу 2.1. В результаті роботи цього оператора кожна подія з лівого (left) потоку даних при знаходженні відповідної події з правого (right) потоку утворює нову подію. На цьому

прикладі показаний принцип формування груп за часом надходження, завдяки чому події з лівого потоку отримують всі відповідні їм майбутні події з правого потоку.

Як показано на рис. 2.1, в результаті об’єднання потоків різних типів даних може сформуватися третій тип, що, наприклад, є структурою даних, що об’єднує вихідні типи даних. Час існування події є обмеженим, завдяки чому події з різних потоків можуть бути об’єдані протягом певного часового вікна. При цьому, кожна подія може мати більш ніж одну відповідну їй подію або не мати жодної [56-58].

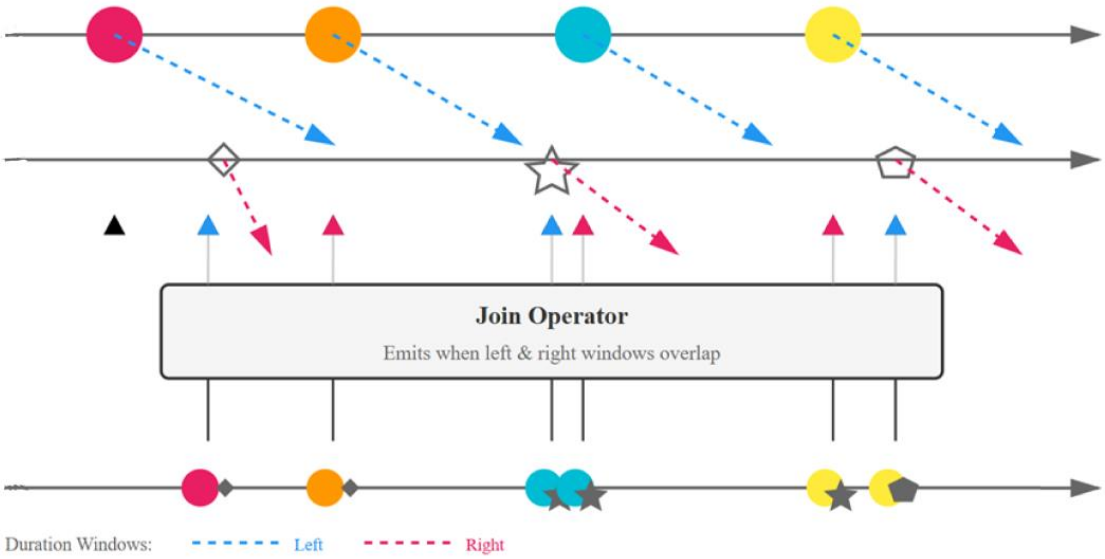


Рис. 2.1. Робота оператору Join

Складнішим варіантом оброблення подій є оператор GroupJoin, приклад роботи якого зображений на рис. 2.2. Його ключова відмінність від оператора Join полягає у тому, що результат роботи цього оператора являє собою не окремі пари подій, а об’єднання події з лівого потоку зі списком подій з правого потоку, що задовольняють критеріям, що видно на лістингу 2.2.

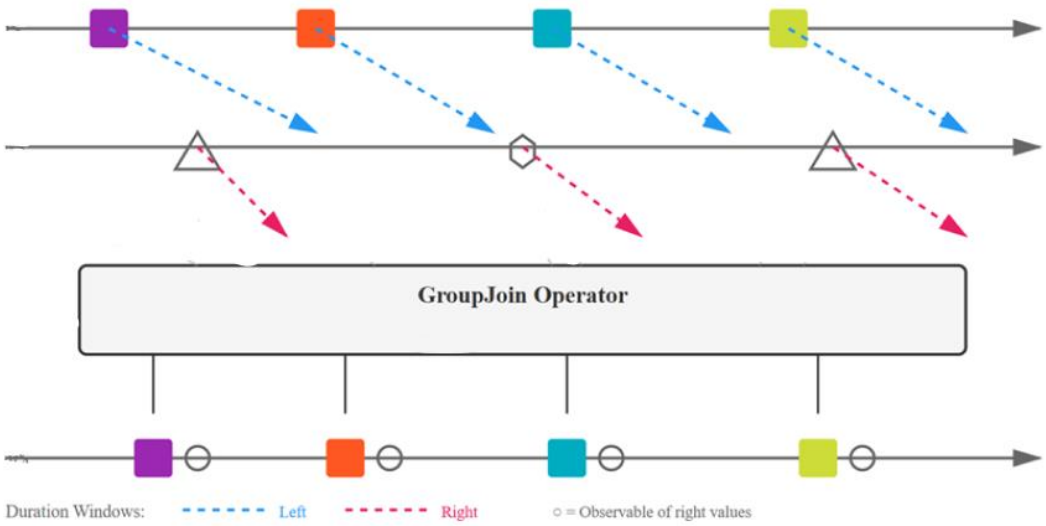


Рис. 2.2. Робота оператору GroupJoin

Лістинг 2.2. Робота оператора GroupJoin з заданими обмеженнями щодо часу існування подій – лівий (left) безкінечно, правий (rigth) зникає відразу

```
left                |-0-1-2-3-4-5|
right               |---A---B---C|
0th window values     --A---B---C|
1st window values       A---B---C|
2nd window values         --B---C|
3rd window values           B---C|
4th window values             --C|
5th window values               C|
```

Ґрунтуючись на вищезазначеному, розглянемо розроблений метод групування темпоральних мультимодальних даних. Метод складається з трьох основних етапів.

Етап 1. Ініціалізація групування потоків

На першому етапі виконується налаштування механізму об’єднання даних, що визначає подальшу логіку роботи всього методу. До оператора GroupJoin передаються два вхідні потоки. Головний потік даних, який вважається опорним, та підлеглий потік даних, що може сформувати групи з даними головного потоку. Одночасно задаються параметри їхньої

взаємодії – максимальний час очікування формування групи Θ, допустима похибка між часовими мітками Δ. Параметри тривалості очікування й значення допустимої похибки підбираються відповідно до умов роботи конкретної системи. А саме, можливих затримок передавання, розбіжності годинників, характеру фізичних процесів, що вимірюються. На цьому ж етапі відбувається підписка на події появи нових елементів у потоках, що дає змогу організувати подальше реактивне оброблення, коли кожен новий елемент автоматично запускає відповідні процедури синхронізації. У результаті виконання цього етапу формується сконфігурований компонент, готовий приймати та групувати дані в режимі реального часу. Аналогічним чином надалі добудовується ієрархічна структура з потоками даних від інших пристроїв.

Етап 2. Отримання, очікування та синхронізація даних

Другий етап розпочинається з моменту надходження чергового елемента від головного пристрою. Цей елемент фіксується як опорний і для нього відкривається інтервал очікування відповідних даних із потоку підлеглого пристрою. У межах заданого вікна часу кожен отриманий елемент підлеглого потоку порівнюється з опорним за часовою міткою та, за необхідності, іншими передбаченими критеріями, з урахуванням максимально допустимої похибки синхронізації. Якщо така відповідність встановлюється, елемент вважається, таким, що може сформувати групу з опорним елементом. У випадку, коли протягом відведеного часу жоден елемент підлеглого потоку не задовольнив критеріям, фіксується факт завершення інтервалу очікування без надходження даних від підлеглого.

Етап 3. Формування групи даних та генерація сповіщення

На третьому етапі, за результатами попередніх етапів, безпосередньо формується група даних, яка представляє набір згрупованих елементів. Якщо для опорного елемента знайдено один або кілька сумісних за параметрами елементів з підлеглого потоку, створюється повна група, що містить дані всіх пристроїв. У разі, якщо відповідних даних не виявлено в

межах заданого інтервалу, формується група, яка включає лише дані від головного пристрою. Після побудови групи генерується подія формування групи, яка передається для подальшого оброблення.

Розглянемо алгоритмічну реалізацію розробленого методу.

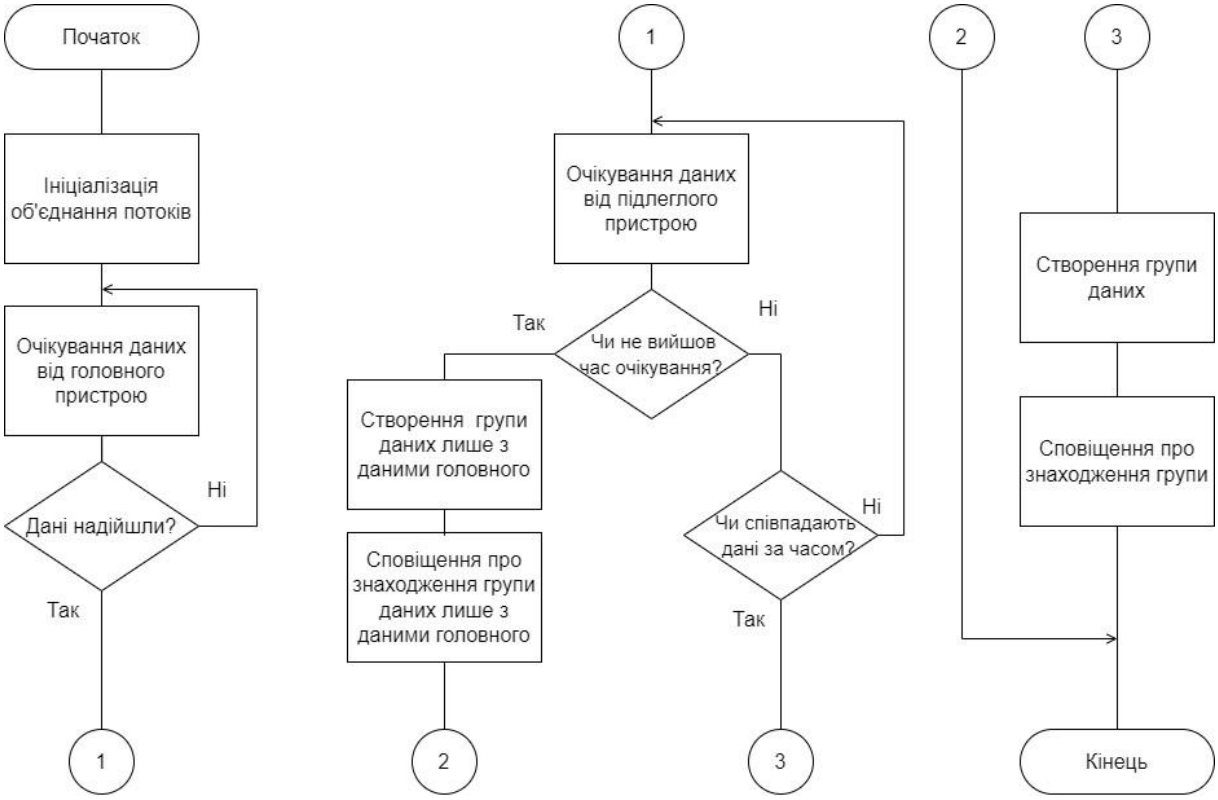


Рис. 2.3. Блок-схема алгоритму групування потокових даних з використанням реактивного програмування

На блок-схемі (рис. 2.3) зображений алгоритм групування темпоральних потокових даних з використанням реактивного програмування. Його основними кроками є наступні.

Ініціалізація групування потоків даних. На цьому кроці до оператора GroupJoin передаються два потоки даних (головний на підлеглий) разом із параметрами роботи. Обмеження до двох потоків пов’язане з тим, що GroupJoin працює саме з парою потоків, які утворюють одну ланку ієрархії.

Потік даних можна описати наступним чином:

$$F_i = \{e_k^i\}_{k=1}^n, \quad (2.1)$$

де $e_k^i = (t_k^i, v_k^i)$ це дані, в який відповідно t_k^i є часовою міткою, а v_k^i є отриманим від пристрою значенням, а $i = [1 \dots N]$, що відповідає кількості пристроїв, від яких отримуються дані.

Параметри інтервалу очікування підлеглих даних Θ і допустимої похибки часових міток Δ задаються з урахуванням очікуваних умов роботи системи. Вони встановлюються залежно від мережевих затримок, можливого дрейфу годинників пристроїв і типового часу надходження даних. Коректний вибір цих параметрів визначає баланс між повнотою груп і затримкою візуалізації. Параметри також мають задовольняти наступній умові:

$$\begin{aligned} \Theta &\geq 0 \\ \Delta &\geq 0 \end{aligned} \quad (2.2)$$

Очікування даних від головного потоку. Система переходить у стан очікування надходження даних від головного потоку. Як тільки з'являється новий елемент даних $e_k^0 = (t_k^0, v_k^0)$, він фіксується разом із його часовою міткою t_k^0, та іншими його даними v_k^0. Після цього для цього елемента починається очікування відповідних даних у підлеглому потоці протягом заданого інтервалу часу Θ. Вікно очікування задається за формулою:

$$W = [t_k^0, t_k^0 + \Theta] \quad (2.3)$$

Очікування даних від підлеглого потоку. Після надходження події від головного потоку система переходить у режим очікування відповідних даних від підлеглого потоку. Протягом цього часу відстежуються всі події, що надходять від підлеглого потоку даних із перевіркою їхніх часових міток. Очікування триває доти, доки не буде виконано умови за часом або не сплине максимально допустимий інтервал очікування Θ.

Перевірка чи не вийшов час надходження даних. Після кожного нового кроку очікування перевіряється, чи не перевищено встановлений параметр максимальної тривалості очікування Θ за умовою:

$$t_{cur} \leq t_k^0 + \Theta, \tag{2.4}$$

де t_{cur} є часовою міткою поточного моменту часу.

Якщо поточний час виходить за межі вікна, пов'язаного з подією головного потоку, подальший пошук підлеглих даних для цього елементу даних головного потоку припиняється. Це дає змогу уникнути нескінченного очікування запізнілих або втрачених повідомлень, що вже є неактуальними та відповідно знизити навантаження за рахунок обмеження часового вікна, впродовж можуть бути потрібні нові групи даних.

Створення групи з даними головного потоку. Якщо час очікування вийшов і придатних підлеглих даних не було знайдено, залежно від попередньої конфігурації формується група, що містить лише дані з головного потоку. Це дає змогу не блокувати подальше оброблення й візуалізацію через нестачу підлеглих даних, а надалі працювати з даними головного пристрою, у разі, якщо такі групи даних вважаються валідними у кожному конкретному випадку. Групи мають наступний вигляд:

$$G_{main} = \{e_k^0\}. \tag{2.5}$$

Сповіщення про групу з даними лише головного потоку. Після формування групи тільки з даними головного потоку генерується відповідна подія або повідомлення в системі. Інші компоненти, що підписані на ці сповіщення, отримують інформацію про сформовану групу та можуть одразу робити її оброблення. Таким чином відбувається робота потокового конвеєра оброблення при роботі з даними у реальному часі.

Перевірка чи збігаються дані за часовою міткою. Якщо дані від підлеглого потоку $e_m^j = (t_m^j, v_m^j)$ надійшли в межах вікна очікування Θ, виконується порівняння їхньої часової мітки з часовою міткою події головного потоку:

$$\Delta \geq |t_m^j - t_k^0|. \tag{2.6}$$

Обчислюється різниця між цими значеннями й перевіряється, чи не перевищує вона задану допустиму похибку Δ. У разі виконання цієї умови дані вважаються придатними для включення до однієї групи.

Створення групи з даними головного та підлеглого потоку даних. Коли часові мітки головної та підлеглої подій задовольняють умову допустимої різниці, формується об'єднана група даних. До неї входять дані із головного потоку та відповідний дані із підлеглого потоку. Такі групи мають наступний вигляд:

$$G_{total} = \{e_k^0\} \bigcup \{v_m^j\}_{j=1}^n, \tag{2.7}$$

де v_m^j відповідає даним, часова мітка яких задовольняє умові 2.6.

Сповіщення про знаходження групи. Після успішного формування повної групи відбувається генерація сповіщення про її створення. Потік оброблення передає цю групу на наступні модулі, такі як компоненти візуалізації або аналітичні блоки. Завдяки цьому інші частини системи можуть одразу використовувати узгоджені за часом дані для побудови графіків, виявлення подій або подальших розрахунків.

Для подальшого групування даних відбувається аналогічне об'єднання даних зі сформованих груп. Також можливо створити реалізацію, що буде групувати дані між потоком сформованих груп від декількох пристроїв та потоком отримання даних від пристрою.

Для зменшення витрат пам’яті та часу при подальшому групуванні дані додаються до списку вже сформованої групи замість формування нової. Таким чином можливо згрупувати дані від довільної кількості пристроїв, сформувавши ієрархію групування. Завдяки цьому можливо розподілити процес групування між різними процесами, забезпечивши масштабованість та розподілення ресурсів системи для обчислення як між ядрами процесора, так можливо і різними серверами залежно від налаштування та розгортання у кожному окремому випадку.

2.3. Програмна реалізація методу групування

Реалізація розробленого методу групування здійснювалася з використанням принципів та інструментів реактивного програмування, зокрема бібліотеки Rx.NET (Reactive Extensions for .NET) [59, 60] для асинхронного і подійного програмування на платформі .NET, яка дає змогу розробникам управляти асинхронними потоками даних та подіями за допомогою функціонального підходу, що спрощує роботу з потоками подій і зменшує складність коду.

Розглянемо компоненти Rx.NET, що використовувалися для реалізації розробленого методу.

Observable. Observable – це основний компонент Rx.NET, який представляє потік подій або даних, який можна «спостерігати». Observable може генерувати значення асинхронно і дає змогу підписуватися на ці значення.

Observer. Observer – це об’єкт, який підписується на Observable для отримання і оброблення даних. Observer реалізує інтерфейс IObserver<T>, який включає методи для оброблення нових значень, помилок і завершення роботи з потоком даних.

LINQ-подібні запити. Rx.NET використовує синтаксис, схожий на LINQ (Language Integrated Query), для роботи з потоками даних. Це дає

змогу створювати, комбінувати і трансформувати потоки даних, використовуючи методи розширення, такі як Select, Where, Merge, і Zip.

Планувальник. Rx.NET використовує планувальники (scheduler) для управління, коли і де будуть виконуватися обробники подій.

Observer є основою подієвої парадигми. Джерела (давачі, мікросервіси) публікують події, а вузли оброблення й візуалізації підписуються на них. У типових IoT-сценаріях потоки «гарячі», тобто генерація не залежить від наявності споживача, отже, потрібні контрольована підписка та відписка на події, що дасть змогу тимчасову роботу систему без обов'язкової наявності споживача. Роз'єднання продукування та споживання дає змогу незалежно масштабувати обидві сторони.

На лістингу 2.3 наведений приклад використання методу GroupJoin з Rx.NET разом з Observer та Observable.

Лістинг 2.3. Приклад роботи з методом GroupJoin [61]

```
var leftList = new List<string[]>();
leftList.Add(new string[] { "2013-1-1 02:0:0", "Batch1" });
leftList.Add(new string[] { "2013-1-1 03:0:0", "Batch2" });
leftList.Add(new string[] { "2013-1-1 04:0:0", "Batch3" });
var rightList = new List<string[]>();
rightList.Add(new string[]{ "2013-1-1 01:0:0", "Prod=2" });
rightList.Add(new string[]{ "2013-1-1 02:0:0", "Prod=0" });
rightList.Add(new string[]{ "2013-1-1 03:0:0", "Prod=3" });
var l = leftList.ToObservable();
var r = rightList.ToObservable();
var q = l.GroupJoin(r,
     _ => Observable.Never<Unit>(), // windows from each left event
going on forever
     _ => Observable.Never<Unit>(), // windows from each right event
going on forever
     (left, obsOfRight) => Tuple.Create(left, obsOfRight)); // create
tuple of left event with observable of right events
// e is a tuple with two items, left and obsOfRight
q.Subscribe(e =>{
    var xs = e.Item2;
    xs.Where(
     x => x[0] == e.Item1[0]) // filter only when datetime matches
     .Subscribe(
     v => {
// оброблення результатів
        ));
     });
});
```

Компоненти програмної системи, розроблені для реалізації методу групування, зображені на рис. 2.4 (модулі, що стосуються даних пристроїв) та рис. 2.5 (модулі, що відповідають за групування даних). Розглянемо їх докладніше.

Модуль опису пристроїв та їхніх даних є першим важливим елементом реалізації розробленої програмної підсистеми потокового групування даних. У системі визначено абстракцію пристрою через інтерфейс і базовий клас, які інкапсулюють логіку генерації або отримання даних та сповіщення про їх надходження за допомогою події. Базові класи для даних пристрою та для груп даних (BaseDeviceData та DataGroup) описують загальну структуру даних, отриманих від пристрою (часова мітка та власне виміряні дані) та групи подій.

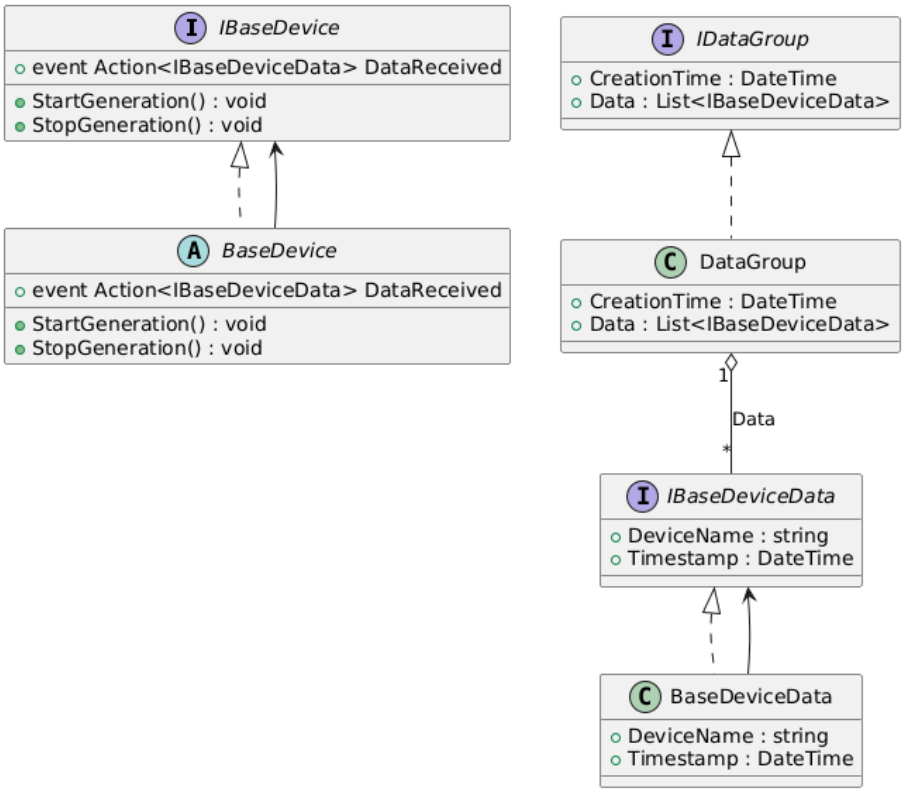


Рис. 2.4. Діаграма класів узагальнених даних пристроїв та груп даних

При цьому вони спроєктовані як універсальні моделі, від яких у подальшому можуть успадковуватися спеціалізовані класи для конкретних

типів пристроїв. Такий підхід дає змогу адаптувати структуру даних до особливостей конкретного застосування. За потреби розробник може додавати нові поля для різних типів сенсорів, зберігати додаткові метадані або агреговані показники, не змінюючи базову архітектуру підсистеми та не порушуючи існуючі механізми оброблення. Таким чином, модуль опису даних забезпечує узагальнену основу для подальшого розширення системи під специфіку окремих IoT-пристроїв.

Модуль базових механізмів групування становить наступний логічний блок реалізації і визначає інфраструктуру для побудови конкретних алгоритмів синхронізації потоків. У цьому модулі визначено інтерфейс і абстрактний клас групування, які задають спільну базу для всіх реалізацій класів групування. Базовий клас групування оперує набором підключених пристроїв і визначає уніфікований контракт взаємодії із зовнішніми компонентами через події, що сповіщають про формування групи даних з включенням даних підлеглого потоку, про створення групи без знаходження даних підлеглого у часовому вікні, а також про створення будь-якої групи.

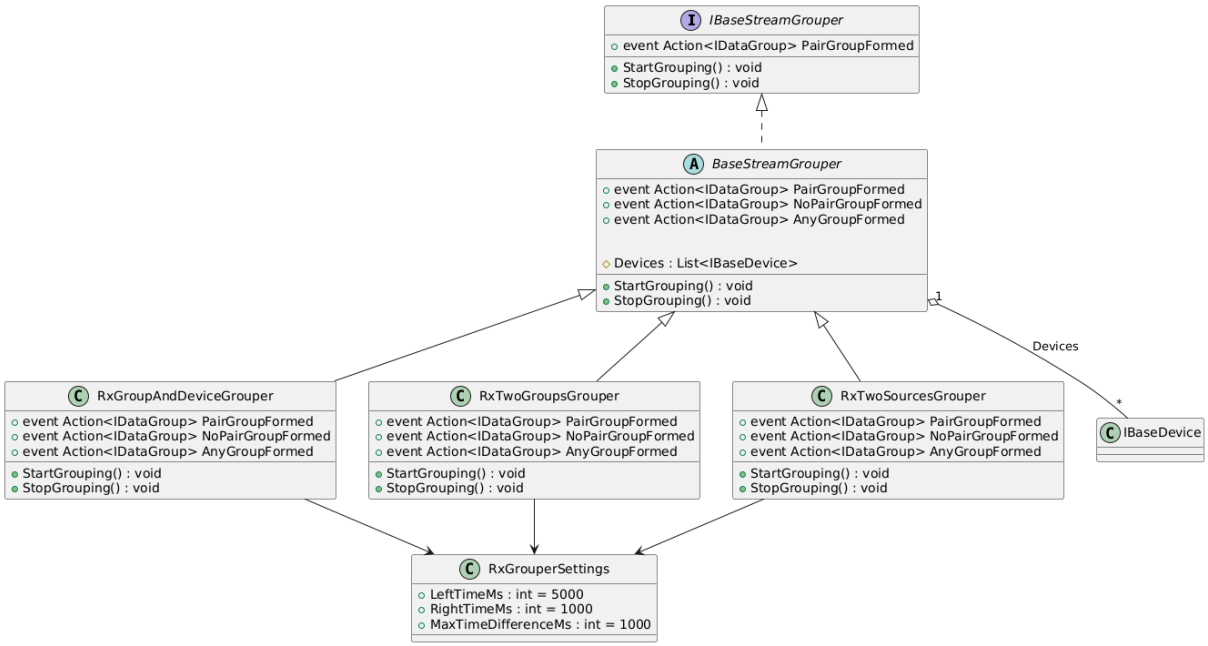


Рис. 2.5. Діаграма класів групування даних пристроїв

Завдяки цьому різні варіанти алгоритмів можуть будуватися поверх єдиного каркаса без дублювання коду, повторно використовуючи загальну логіку керування пристроями, реєстрації підписників та маршрутизації результатів. У цьому модулі також розміщено кілька реалізацій, що працюють поверх базового каркаса. Одна з них орієнтована на об’єднання двох потоків даних від пристроїв, інша поєднує вже сформовану групу з даними окремого пристрою, а ще одна реалізація виконує об’єднання двох попередньо агрегованих груп. Усі реалізації використовують спільну модель подій та спираються на механізми реактивного програмування для організації асинхронного оброблення потоків даних, а також можливість масштабування за кількістю пристроїв формуючи ієрархічну структуру групування даних.

Модуль налаштувань групування відповідає за параметрування алгоритмів та забезпечує їхню гнучкість. Для цього введено спеціальний клас налаштувань, який зберігає часові характеристики процесу групування, зокрема тривалість вікон очікування даних, а також максимально допустиму різницю між часовими мітками подій, що мають вважатися групою. Конфігурація винесена в окрему сутність, унаслідок чого алгоритми працюють із єдиною моделлю параметрів, а зміна налаштувань не потребує втручання в їхній внутрішній код. Це дає змогу адаптувати підсистему до різних режимів роботи IoT-пристроїв, змінюючи допустимі затримки без перероблення логіки групування.

Розглянемо потік даних в розробленій реалізації. Діаграма послідовності, зображена на рис. 2.6, ілюструє роботу групування даних у випадку взаємодії двох пристроїв з використанням компонента RxTwoSourcesGrouper. На ній показано основних учасників процесу – головний пристрій, підлеглий пристрій, модуль групування, механізм реактивного об’єднання потоків GroupJoin, обробник подій OnNext та підсистема візуалізації або подальшого оброблення результатів, яка власне і використовує згруповані дані. На початковому етапі головний і підлеглий

пристрої запускають генерацію або ж отримання даних, після чого модуль групування ініціалізує процес роботи, створюючи реактивне об’єднання двох потоків за допомогою GroupJoin і підписуючись на результати через виклик Subscribe з переданням обробника OnNext.

Після ініціалізації алгоритму на діаграмі відображається типовий цикл оброблення вхідних подій. Головний пристрій передає до модуля групування чергове вимірювання, яке маршрутизується у потік master. Аналогічним чином підлеглий пристрій надсилає свої дані, що потрапляють у потік slave. Компонент GroupJoin синхронізує ці події відповідно до заданих часових вікон, формуючи для кожної події з головного потоку пару у вигляді об’єкта, що містить master-дані та реактивну послідовність потенційних slave-подій. Ця пара передається до обробника OnNext, який реалізує прикладну логіку співставлення подій.

Подальша поведінка системи залежить від того, чи надійшли дані від підлеглого пристрою з відповідною часовою міткою. Обробник перевіряє, чи існує подія підлеглого потоку, що потрапляє до визначеного вікна часу, і чи не перевищує різниця між часовими мітками значення максимально допустимого відхилення. У випадку, коли така подія знайдена і часові умови виконуються, формується об’єднана група даних, після чого генерується подія PairGroupFormed, а також, залежно від початкової конфігурації, загальна подія AnyGroupFormed, сповіщаючи підсистему візуалізації або оброблення про успішне формування пари.

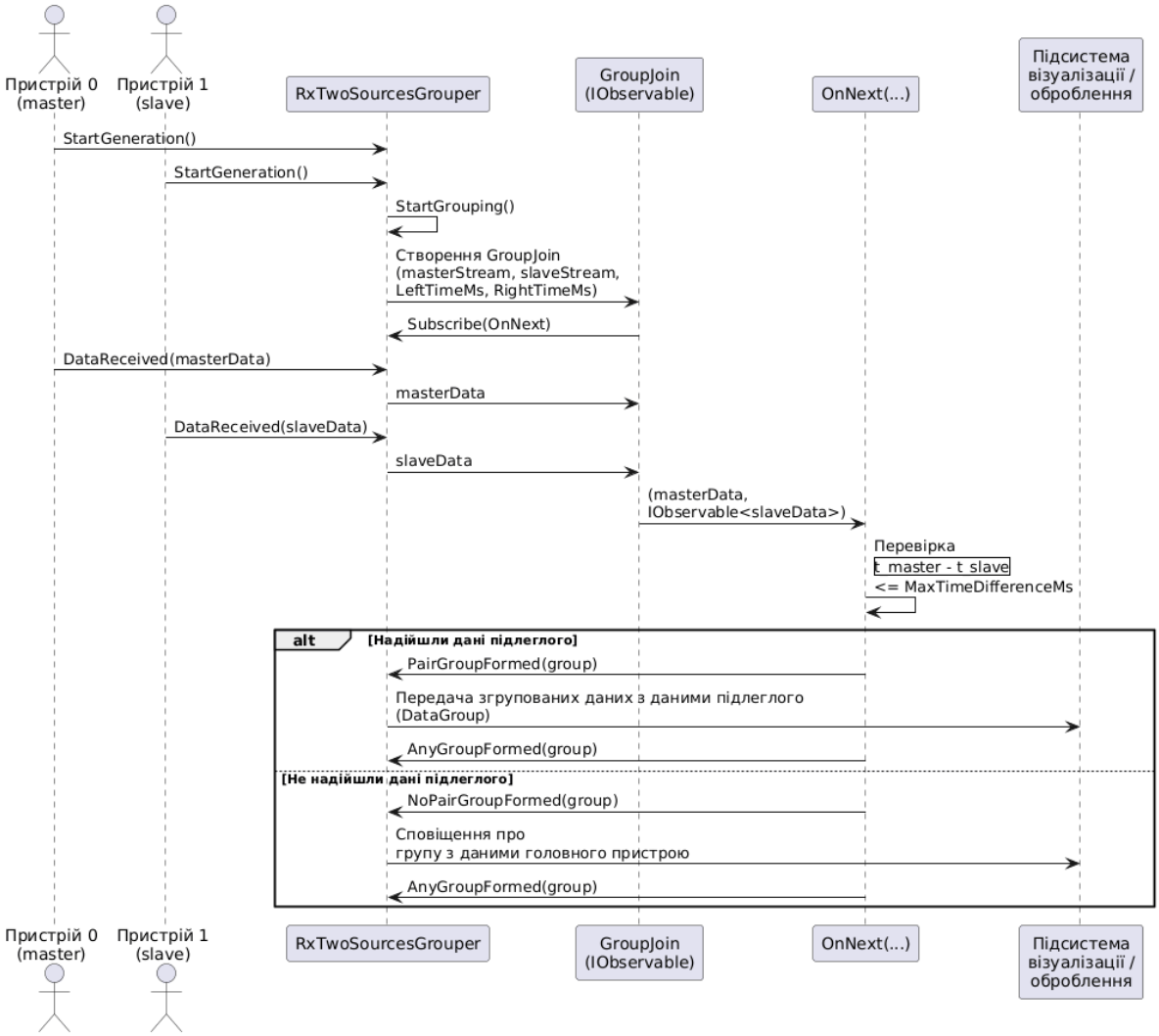


Рис. 2.6. Діаграма послідовності реалізації методу групування

Якщо ж відповідної події підлеглого потоку не знайдено або вона не задовольняє часовим обмеженням, формується група лише з даними головного пристрою. У такому разі він генерує подію NoPairGroupFormed і знову ж таки подію AnyGroupFormed, передаючи інформацію про результат. Таким чином, діаграма послідовності демонструє повний життєвий цикл оброблення однієї події від головного пристрою від моменту її надходження до формування групи з даними підлеглого пристрою або закінчення часу очікування даних від підлеглого потоку.

2.4. Оцінювання ефективності розробленого методу

Вимірювання продуктивності проведені з використанням BenchmarkDotNet. Точність BenchmarkDotNet досягається завдяки ретельному урахуванню різних факторів, які можуть спотворювати результати бенчмарків. Бібліотека автоматично нейтралізує вплив таких аспектів, як оптимізації JIT-компілятора, адаптивна робота GC (Garbage Collector, «збирача сміття»), а також шум, що створюється фоновими процесами операційної системи. Для зменшення випадкових похибок кожен тест виконувався багаторазово, результати усереднювалися, а статистичний аналіз дає змогу побачити як середнє значення, так і розкид, медіану, стандартне відхилення та інші метрики. Такий підхід гарантує стабільність і відтворюваність результатів навіть у різних середовищах виконання.

Окремо у процесі тестування враховувалися особливості середовища виконання, оскільки продуктивність алгоритмів, що працюють з потоками даних та асинхронними обчисленнями, суттєво залежить від платформи, реалізації «збирача сміття» та налаштувань оптимізації.

Тому під час експериментів використовувалися однакові сценарії запуску, уніфіковані вхідні дані та фіксовані параметри конфігурації, що дало змогу мінімізувати вплив сторонніх чинників і зосередитися на оцінці власне запропонованого методу. Отримані результати далі використовуються для аналізу розробленої реалізації та формулювання висновків щодо придатності методу для застосування в задачах візуалізації потокових IoT-даних.

Таблиця 2.1. – Порівняння швидкодії наївного та Rx методу

Кількість згенерованих даних, од	Час виконання Rx, мс	Час виконання Naive, мс
30	15	17
60	20	23
125	25	27
250	30	35
500	46	56
1000	109	120
2000	347	311

На рис. 2.7 зображений результат бенчмарку, швидкодія для невеликої кількості є схожою, проте при зростанні кількості даних для групування швидкодія для групування потокових даних зменшується. Цей та наступні бенчмарки були запущені на такій конфігурації: win10-x64, .NET 8, Intel i7- 7700HQ. Детальні результаті бенчмарку зазначені в табл. 2.1. Приклад базового бенчмарку наведений в лістингу 2.4.

Лістинг 2.4. Приклад створення базової частини для подальших бенчмарків

```
    public static RxTwoGroupsGrouper
Get4GeneratorsGrouper(List<LDBTG<BaseDeviceData>> generators,
List<BaseStreamGrouper> groupers, int masterIntervalMs, int count) {
        var dgsr = new LDBTG<BaseDeviceData>(100, "SlaveDevice", count /
2);
        var tsg2 = new RxTwoSourcesGrouper(new List<IBaseDevice> { left,
leftS }, new RxGrouperSettings { MaxTimeDifferenceMs = 5000 });
        var tgg = new RxTwoGroupsGrouper(new List<IBaseDevice>(), tsg1,
tsg2, new RxGrouperSettings { MaxTimeDifferenceMs = 5000 });
 generators.Add(dg); generators.Add(left); generators.Add(leftS);
generators.Add(dgsr); groupers.Add(tsg1); groupers.Add(tsg2);
groupers.Add(tgg);
        return tgg;
    }
```

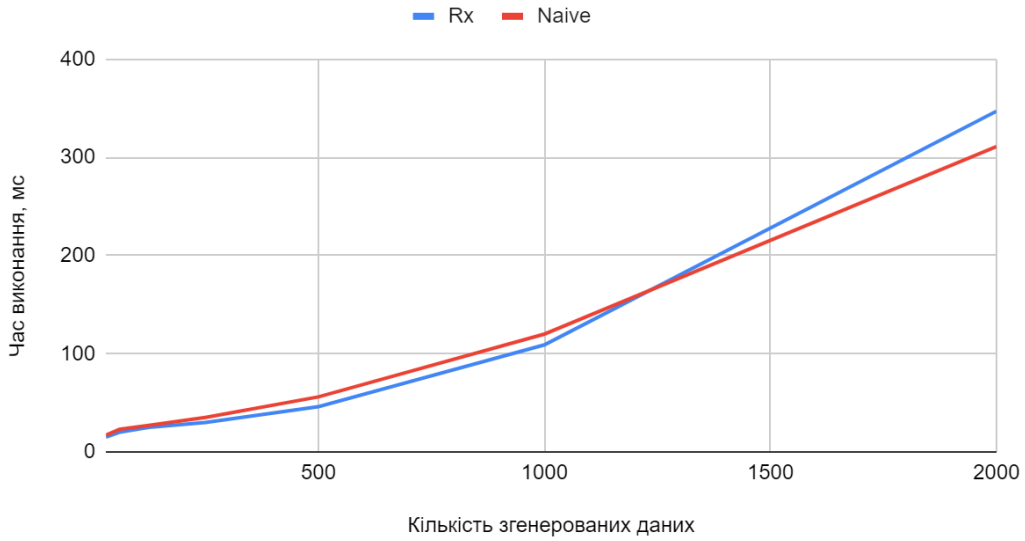


Рис. 2.7. Результати перевірки роботи реалізацій методів з використанням Rx та наївної реалізації

Очікувано, при зростанні кількості пристроїв зростає і використана пам’ять. Проте, при подвоєнні кількості пристроїв коефіцієнт збільшення використаної менше, ніж у два рази. Детальні результаті бенчмарку зазначені в табл. 2.2.

Таблиця 2.2. – Вимірювання швидкодії

Генератори даних, од	Середній час виконання, мс	Виділено пам’яті, МБ	Згенеровано даних, од
2	27	0,5	250
4	43	25	510
8	116	32	1020
16	208	26	2500
32	250	32	5700
64	337	43	11200
128	503	71	23500

На рис. 2.8 зображений результат бенчмарку для запропонованої реалізації для різної кількості емуляторів пристроїв, що генерують дані. Швидкість генерування даних сягає 40 тисяч за секунду сумарно для всіх пристроїв.

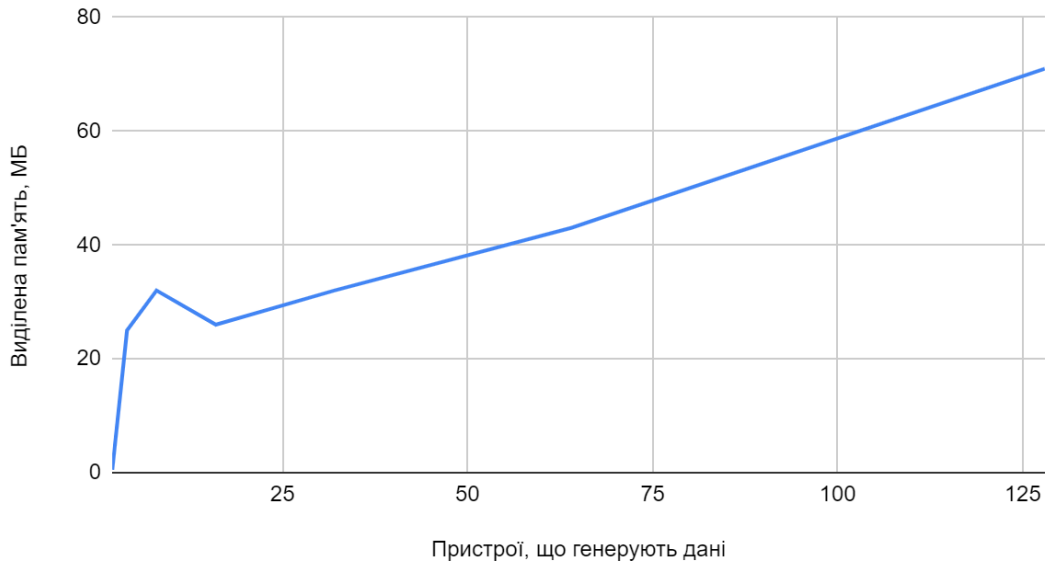


Рис. 2.8. Результати перевірки роботи реалізації методу з використанням Rx по виділеній пам’яті

На рис. 2.9 зображений аналогічний бенчмарк до проведеного на рис. 2.8, але вимірювався час виконання групування. Помітне зростання витраченої оперативної пам’яті до попереднього бенчмарку, тобто при подвоєнні кількості пристроїв час виконання зростає менше, ніж у два рази.

Отримані результати бенчмарків дають змогу зробити наступні висновки.

Однією з ключових переваг методу є толерантність до мережевих затримок та неточності годинників пристроїв. Використання часових вікон і допусків під час формування груп подій дає змогу коректно об’єднувати дані навіть тоді, коли події від різних пристроїв надходять із запізненням або в іншому порядку, ніж були згенеровані, що є типовою ситуацією для розподілених IoT пристроїв.

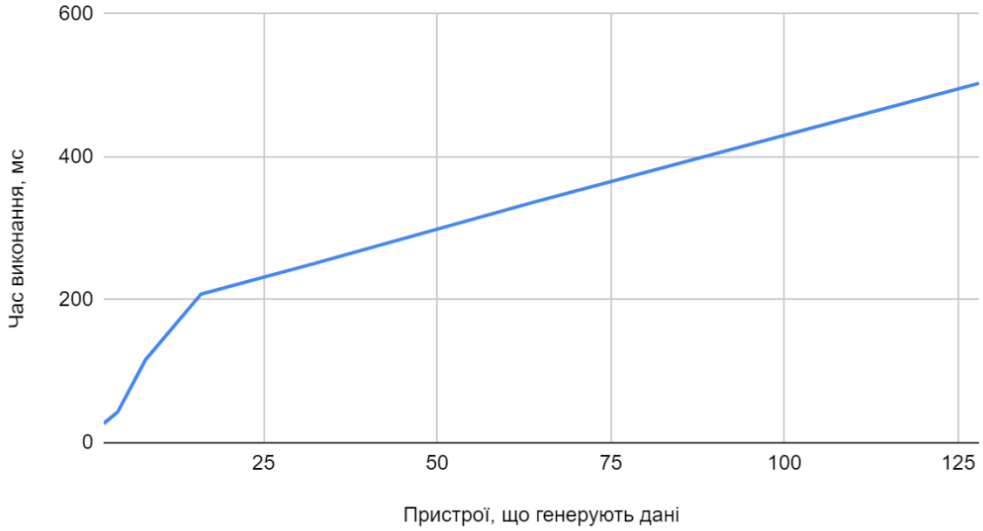


Рис. 2.9. Результати перевірки роботи реалізації методу з використанням Rx за часом виконання

Ще одна важлива перевага полягає у використанні реактивного підходу та фреймворку Rx.NET. Формування груп відбувається без необхідності проміжного збереження великих обсягів даних безпосередньо у реалізації. Оброблення виконується безпосередньо в потоці подій, що сприяє підвищенню продуктивності та спрощує побудову складних конвеєрів оброблення.

Також особливістю є використання реактивної моделі на базі Rx.NET, яка має аналоги та портовані реалізації на інших платформах і мовах програмування. Завдяки цьому логіку розробленої реалізації можна переносити в інші технологічні стеки, відтворюючи ті самі оператори та принципи оброблення потоків даних без суттєвої зміни загальної схеми групування.

Завдяки узагальненим інтерфейсам і базовим класам для даних та груп поточна реалізація не прив’язана до конкретного домену чи модальності вимірювань, тобто розробник може розробити відповідні класи, що наслідуються від базових типів для різних предметних галузей без зміни основної логіки групування.

Серед особливостей запропонованого методу варто відзначити можливість побудови ієрархії групування, що дає змогу гнучко реалізовувати необхідні вимоги групування з урахуванням особливостей як самих пристроїв, так і їх підключення до мережі. Відповідно, таке налаштування вимагає часу і ресурсів для аналізу та встановлення необхідних параметрів для конфігурації ланок методу.

Водночас якість групування істотно залежить від правильної конфігурації параметрів. Значення часових допусків, тривалість вікон і правила оброблення запізнілих подій впливають на те, чи будуть коректно виділені логічні залежності між подіями. Некоректно підібрані параметри можуть спричинити або об’єднання несуміжних подій в одну групу, або, навпаки, розрив логічно пов’язаних подій між різними групами, що знижує надійність аналітичних висновків у граничних режимах при некоректному попередньому встановленні параметрів конфігурації.

Додатковим обмеженням є залежність від припущень щодо якості часової синхронізації між пристроями. За значних розбіжностей годинників або за відсутності механізмів корекції навіть гнучкі часові допуски можуть виявитися недостатніми для формування коректних груп. Крім того, для забезпечення толерантності до затримок необхідно тимчасово зберігати всі дані головного пристрою до завершення вікна групування, що за інтенсивних потоків даних або довгих вікон очікування призводить до збільшення споживання пам’яті.

Подальше вдосконалення методу може бути спрямоване на впровадження адаптивних механізмів автоматичного налаштування параметрів групування. На основі поточних характеристик потоків (інтенсивність подій, статистика затримок) система могла би динамічно коригувати часові вікна, значення допусків, що зменшило би залежність від ручного підбору параметрів.

Перспективним є також використання методів машинного навчання для оцінювання якості сформованих груп та виявлення аномальних

сценаріїв, таких як систематичні запізнення окремих пристроїв, нетипові інтервали між подіями або надмірна кількість «порожніх» груп. Такі моделі можуть як пропонувати кращі значення параметрів, так і сигналізувати про необхідність технічного обслуговування чи переналаштування пристроїв.

Доцільним є й подальший розвиток інструментарію моніторингу й діагностики, тобто розроблення засобів візуалізації структури груп у реальному часі, детального журналювання (логування) рішень щодо включення чи виключення подій до груп, а також введення метрик якості групування (частка незгрупованих подій, середній час очікування формування групи тощо). Це полегшить налагодження, експлуатацію та вдосконалення відповідного програмного забезпечення.

Для спрощення налаштування можливе групування даних на кожній ланці одразу від багатьох пристроїв, що дасть змогу зменшити налаштування кожної окремої ланки та спростити побудову ієрархії.

2.5. Візуалізація згрупованих подієвих даних

Реалізація запропонованого подієво-орієнтованого методу групування створює фундаментальне підґрунтя для реалізації якісно нових типів користувацьких інтерфейсів. Такі інтерфейси можна класифікувати як подієво-орієнтовані, оскільки основною одиницею візуальної інформації в них стає не окремий потік від датчика, а подія. У цій парадигмі подія фактично перетворюється на самостійний тип інтерфейсного елемента, який інкапсулює в собі дані різної природи, пов’язані спільним часовим контекстом. Це дозволяє змінити спосіб взаємодії оператора з системою, зміщуючи акцент з безперервного спостереження за показниками на аналіз конкретних інцидентів та значущих станів об’єкта моніторингу.

Розроблений інтерфейс, представлений на рис. 2.10, демонструє практичне втілення цієї концепції. На графічному полі замість традиційних лінійних графіків відображаються дискретні візуальні маркери, кожен з яких відповідає події. Кожна така точка на часовій шкалі є результатом

роботи методу групування, який виділив події з мультимодальних потоків. Це забезпечує високу щільність інформації. Оператор бачить не просто факт спрацювання датчика, а цілісну подію, що має часову мітку та асоційований набір даних від усіх активних пристроїв у системі.

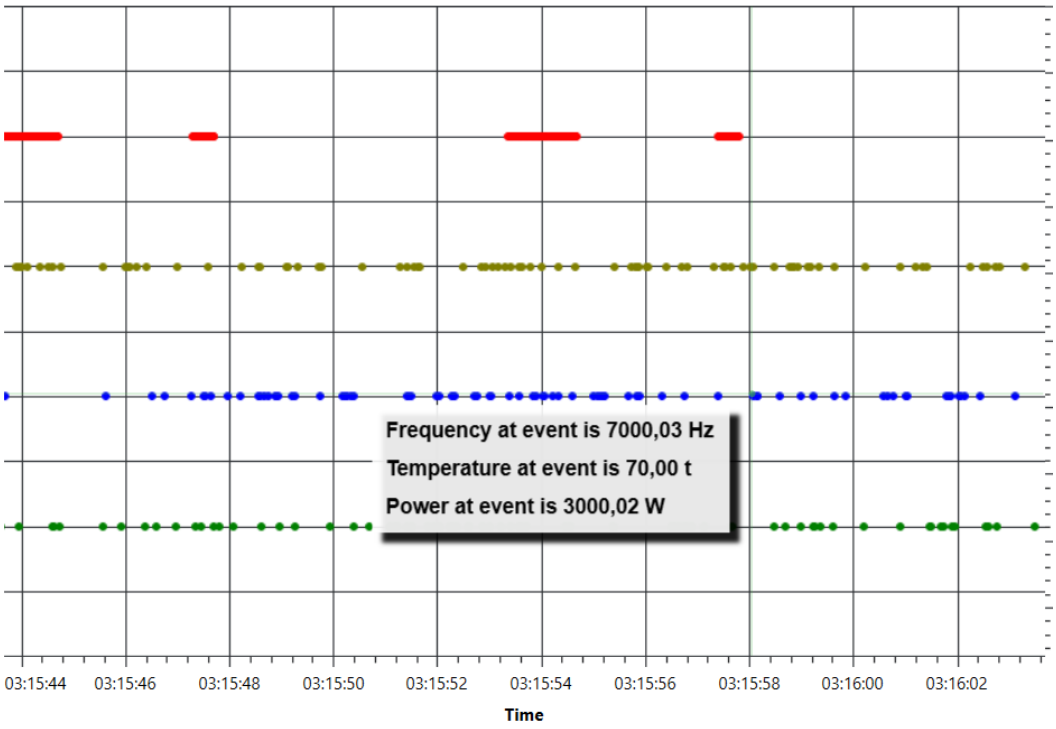


Рис. 2.10. Приклад реалізації подієво-орієнтованого інтерфейсу візуалізації згрупованих даних

Можливим напрямом розвитку таких інтерфейсів є впровадження глибокої інтерактивності, де подія стає активним об’єктом маніпуляції. Наступним кроком у вдосконаленні програмних механізмів є реалізація функцій швидкого доступу до деталізованих даних через безпосередню взаємодію з візуальним маркером події. Наприклад, при виконанні користувачем кліку мишкою на точку, що відповідає появі події, система повинна автоматично ініціювати відкриття контекстного вікна або додаткової панелі. У цьому вікні відображаються специфічні для даної події дані. Відеофрагмент, зафіксований у момент виникнення події, відтворюється аудіозапис або виводяться точні цифрові показники

телеметрії. Таким чином, подія на головній шкалі часу виступає як інтерактивний ключ до всього масиву мультимодальної інформації, що дозволяє користувачу переходити від загального огляду до детального аналізу конкретного інциденту без необхідності ручного пошуку по різних каналах даних.

Ще одним напрямом подальшого розвитку таких інтерфейсів є впровадження механізмів візуалізації подій у вигляді часових смужок, що дозволить відображати тривалість події від моменту її фактичного початку до завершення. Такий підхід відкриває можливості для якісно нового представлення даних, які мають певну протяжність у часі, як-от процеси поступового підвищення температури або тривалість перебування об’єкта в зоні відеоспостереження. Теоретично, використання як точкових маркерів, так і часових інтервалів дозволить оператору візуально оцінювати не лише факт виникнення події, а й інтенсивність та тривалість інцидентів безпосередньо на часовій осі. Це створить умови для наочної демонстрації перекриття різних подій у часі, що є критично важливим для глибшого розуміння причинно-наслідкових зв’язків між мультимодальними потоками.

2.6. Висновки до другого розділу

У другому розділі сформульовано вимоги до групування даних, такі як можливість перебудування ієрархічної структури групування, що може складатися з декількох рівнів ієрархії, швидкодія, робота з потоковими темпоральними гетерогенними даними та наведено розроблений метод групування потокових мультимодальних темпоральних даних.

Метод групування розроблено з використанням операторів реактивного програмування. Програмна реалізація методу перевірена з використанням BenchmarkDotNet.

Показано, що розроблений метод уможливлює групування даних від багатьох пристроїв з можливістю виконувати групування на різних серверах

завдяки особливостям цього методу, що дає змогу об’єднувати входи і виходи різних шарів групування. Також завдяки особливостям обраної бібліотеки Rx.NET цю ієрархічну структуру можливо перебудувати за необхідності у разі додавання нових пристроїв або видалення старих.

Запропоновано якісно новий тип користувацьких інтерфейсів – подієво-орієнтований інтерфейс, де кожен маркер представляє собою подію. Наведено підхід до візуалізації згрупованих даних у формі дискретних маркерів подій замість безперервних потоків. Запропоновано напрями подальшого розвитку таких інтерфейсів.

РОЗДІЛ 3. РОЗРОБЛЕННЯ МЕТОДУ ВІЗУАЛІЗАЦІЇ ТЕМПОРАЛЬНИХ МУЛЬТИМОДАЛЬНИХ ПОТОКОВИХ ДАНИХ ПРИСТРОЇВ ІОТ ЗІ ЗБЕРЕЖЕННЯМ СТАБІЛЬНОСТІ ВІДОБРАЖЕННЯ

3.1. Формулювання вимог до візуалізації IoT-даних

Візуалізація потокових темпоральних мультимодальних даних, отриманих від пристроїв Інтернету речей (IoT), є однією з найбільш актуальних задач сучасних інформаційних систем. Значний обсяг інформації, різноманітність типів даних і необхідність відображення у реальному часі ставлять перед розробниками специфічні вимоги до методів візуалізації, які мають забезпечити не тільки ефективну презентацію даних, але й їхню інтерпретацію, аналіз та підтримку процесу прийняття рішень [62-65].

Одним із ключових аспектів візуалізації потокових темпоральних мультимодальних даних є забезпечення стабільності візуального представлення попередньо відображених даних. У разі постійного оновлення інформації без належних методів стабілізації відображення, користувачі можуть зіткнутися з проблемою когнітивного перевантаження, спричиненого частими та неконтрольованими змінами візуального представлення. Така ситуація особливо критична для сфер, де прийняття швидких та точних рішень на основі даних є ключовим фактором успіху, зокрема в індустріальному інтернеті речей (IIoT), розумних містах (Smart City) та електронному здоров'ї (eHealth).

Таким чином, процедури візуалізації IoT-даних мають задовольняти таким основним вимогам.

1. Стабільність візуалізації:
   – забезпечення незмінності або мінімальної змінності раніше візуалізованих даних у випадку оновлення інформації;

- мінімізація негативного впливу динамічних оновлень на сприйняття інформації користувачами.

2. Робота з великими потоками даних у реальному часі:
   - можливість ефективно обробляти великі масиви інформації, що надходять від численних пристроїв IoT;
   - можливість візуалізації інформації у режимі, близькому до реального часу, без значних затримок, які можуть негативно вплинути на прийняття оперативних рішень.
3. Зменшення когнітивного навантаження на користувачів:
   - використання методів, які дають змогу уникнути інформаційного перевантаження за рахунок агрегування та фільтрації інформації;
   - представлення даних таким чином, щоб користувачі могли швидко їх аналізувати.
4. Оптимізація використання обчислювальних ресурсів:
   - мінімізація споживання ресурсів CPU і GPU завдяки вибірковому зменшенню обсягів даних, які підлягають візуалізації;
   - забезпечення ефективності роботи системи в умовах обмежених апаратних ресурсів, зокрема у вбудованих системах або портативних пристроях.
5. Адаптивність до змінних умов і конфігурацій:
   - можливість динамічного налаштування параметрів візуалізації (наприклад, часових вікон або рівнів деталізації) відповідно до потреб користувача або змінних характеристик потоків даних;
   - гнучкість у зміні типів та обсягів відображуваної інформації залежно від ситуації та галузі застосування.
6. Підтримка мультимодальності:
   - забезпечення можливості інтегрованого представлення різнорідних типів даних, що надходять від різних типів пристроїв;

- можливість одночасно представляти взаємозв’язки між кількома параметрами.

7. Масштабованість:
   - можливість нарощування обсягів візуалізованих даних без суттєвого погіршення продуктивності;
   - можливість ефективно працювати як у невеликих, так і в масштабних розподілених середовищах IoT.
8. Інтерактивність:
   - підтримка інтерактивних елементів для можливості деталізації даних за вимогою користувача (наприклад, зміна масштабу, вибіркове фільтрування даних тощо).

Дотримання цих вимог уможливлює створення програмного забезпечення для аналізу й візуалізації потоків даних IoT у різних галузях застосування. Задовільнення зазначених критеріїв також дасть змогу підвищити точність ухвалення рішень, зменшити витрати на аналіз і, загалом, покращити якість роботи з даними, що є особливо актуальним у сучасному глобальному контексті.

3.2. Розроблений метод візуалізації

Враховуючи сформульовані вище вимоги, особливо актуальним завданням є створення методу візуалізації, який дає змогу ефективно працювати з великими обсягами темпоральних мультимодальних потокових даних IoT у режимі реального часу. Для досягнення цієї мети пропонується метод, що ґрунтується на технології даунсемплінгу (зменшення вибірки) з одночасним забезпеченням стабільності попередньо візуалізованих даних.

На рис. 3.1 зображено приклад візуалізації темпоральних даних у реальному часі, де по осі X відкладено часовий параметр (*Time parameter*), а по осі Y – значення вимірюваного параметра (*Value parameter*, у прикладі – *Param A*). Час представлений на осі та відображає інтервал від

поточного моменту (*Start – current time*) праворуч до кінцевої точки відображення, що визначається як поточний час мінус встановлене значення вікна відображення даних. Таким чином, відображається лише заданий часовий інтервал актуальних даних, що забезпечує релевантність представленої інформації. Напрямок руху даних на графіку спрямований з правого краю до лівого: нові значення надходять праворуч, а старі поступово зміщуються до лівого краю та врешті видаляються з області візуалізації. Значення параметра *Param A* відображено у вигляді серії точок (на даному графіку представлено червоними круговими маркерами), розташованих відповідно до часових міток на осі X та величин на осі Y. Візуалізація передбачає стабільність відображення вже зафіксованих значень, тоді як нові дані плавно додаються праворуч, зберігаючи контекст і сприйняття часової послідовності.

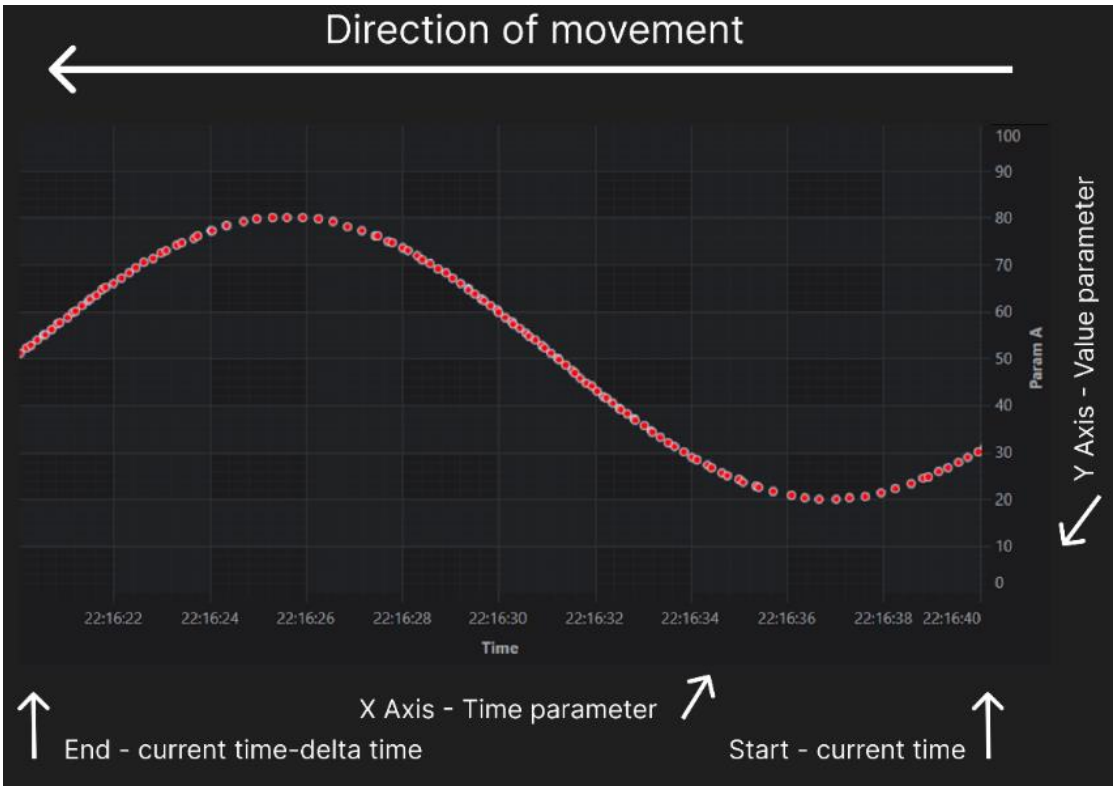


Рис. 3.1. Приклад графіку відображення темпоральних точкових даних на 2D площині

Основною ідеєю, що лежить в основі розробленого методу візуалізації, є вибіркове зменшення кількості точок даних, які відображаються у візуальному інтерфейсі, шляхом їхнього агрегування та ресемплінгу з метою зниження навантаження на систему при відображенні. Ключовою особливістю методу є забезпечення стабільності візуалізації, що передбачає збереження або мінімальні зміни вже представлених на графіку даних, навіть при оновленні інформації з джерел у реальному часі.

Запропонований метод складається з двох основних етапів.

Етап 1. Групування отриманих потокових даних.

На цьому етапі дані поділяються на три групи. Перша група – це застарілі дані, які виходять за межі визначеного часовим інтервалом вікна. Вони більше не мають аналітичної цінності та не беруть участі в ресемплінгу, а лише зберігаються для архіву або відразу видаляються для звільнення ресурсів. Серед усіх наявних даних $A = \langle a_i \rangle_{i=1}^{n}$ вони виділяються за формулою (3.1), де $t_{current}$ є часом початку обчислення, $t_{display}$ є максимальним часом відображення даних, а t_i є часом надходження даних a_i. Друга група – це стабільні дані, які вже були відображені раніше і не можуть бути змінені для збереження візуальної стабільності. Такі дані визначаються за формулою (3.2), у якій $t_{resample}$ є максимально дозволеним часом для ресемплінгу даних. Третя група складається з нових змінних даних, які ще не зафіксовані у візуалізації та можуть бути піддані ресемплінгу, визначається за формулою (3.3).

$$d_i = \begin{cases} a_i, & t_i < t_{current} - t_{display}, \\ \emptyset, & t_i \geq t_{current} - t_{display}. \end{cases} \quad (3.1)$$

$$c_i = \begin{cases} a_i, & t_{current} - t_{display} \leq t_i < t_{current} - t_{resample}, \\ \emptyset, & \text{інакше.} \end{cases} \quad (3.2)$$

$$r_i = \begin{cases} a_i, & t_{current} - t_{resample} \leq t_i < t_{current}, \\ \emptyset, & \text{інакше.} \end{cases} \quad (3.3)$$

Етап 2. Даунсемплінг нових даних (ресемплінг).

Нові дані підлягають даунсемплінгу, що передбачає вибіркове зменшення кількості даних. Використовується один із наступних підходів ресемплінгу: вибір репрезентативних точок (наприклад, середні, мінімальні, максимальні значення) та рівномірний відбір точок за заданими інтервалами часу. Важливою умовою є обмеження максимального періоду для даунсемплінгу, що гарантує збереження стабільності попередньо відображених даних.

Розглянемо алгоритмічну реалізацію даунсемплінгу.

Алгоритм реалізовано у вигляді послідовності кроків, що формують замкнений цикл оброблення та виведення даних.

Ініціалізація параметрів системи. Цей крок є початковим і важливим для коректної роботи всієї системи. Він виконується один раз перед запуском основного циклу оброблення потоків. Ініціалізація включає налаштування таких параметрів:

1. Часовий інтервал візуалізації ($t_{display}$): визначає вікно часу, в межах якого дані мають аналітичну та візуальну цінність. Усі точки даних, час надходження яких перевищує $t_{display}$ відносно поточного моменту, вважаються застарілими.
2. Параметри ресемплінгу ($t_{resample}$): визначають часовий діапазон, у межах якого дозволено виконувати даунсемплінг нових точкових даних.
3. Параметри групування: визначають критерії об’єднання даних за типами, джерелами, часовими відрізками, просторовими координатами тощо. Це дає змогу працювати з мультимодальними даними (температура, вологість, GPS координати тощо) у єдиному контексті.
4. Графічні налаштування: вибір типу графіка (2D, 3D), розміру маркерів, кольорових схем (наприклад, кольору маркерів), параметрів масштабування та стилів осей.

Ініціалізація виконується на рівні конфігурації системи в момент запуску або при перезапуску візуалізаційного модуля, та визначає поведінку протягом усього часу роботи.

Очікування надходження даних. Після ініціалізації система переходить у режим пасивного очікування даних від потокових джерел. Цей етап реалізований як нескінченний цикл опитування або підписки на події (event-driven model), в якому система:

- контролює стан з'єднання з брокером або потоком;
- виявляє появу нових подій у черзі повідомлень;
- фіксує факт надходження нових точкових даних (timestamp, id, payload).

Використання реактивної моделі (наприклад, Rx.NET, Kafka Streams) дає змогу зменшити затримки оброблення й уникнути перевантаження системи при великій кількості подій.

Перевірка та фільтрація даних. Після надходження нових повідомлень, система виконує вхідну перевірку (input validation), що включає:

- перевірку наявності даних у повідомленні (наявність payload, timestamp, id);
- фільтрацію за часом: перевірка, чи входять дані у поточне вікно $t_{display}$;
- перевірку на допустимість параметрів: чи відповідають дані типовим значенням (наприклад, температура – в межах фізично реалістичного діапазону для даних умов);
- форматування: приведення до єдиного формату для подальшої агрегації.

Дані, що не проходять перевірку, не потрапляють у наступні кроки оброблення й можуть бути або заархівовані, або проігноровані.

Розподіл даних на групи. Центральним кроком алгоритму є логічне розділення отриманих даних на три категорії:

1. Група D = $\langle d_i \rangle_{i=1}^{n}$ (застарілі дані). Визначаються за формулою 3.1. Ці дані більше не мають цінності для аналітики або візуалізації, і підлягають видаленню або архівації. Вони не потрапляють у візуалізаційний буфер. Дані потрапляють у цю групу, якщо їхня часова мітка вказує, що вони старіші за поточний момент часу з урахуванням часу відображення $t_{display}$.
2. Група C = $\langle c_i \rangle_{i=1}^{n}$ (стабільні дані). Дані, що вже були відображені на графіку, і які не повинні змінюватися для збереження цілісності відображення. Їхня часова мітка має бути між часом, що визначається як граничний для групи видалених даних, та часом, в який ще може відбуватися ресемплінг. Ці дані залишаються у візуалізації як «зафіксовані», не піддаються зміні або перерахунку.
3. Група R = $\langle r_i \rangle_{i=1}^{n}$ (нові змінні дані). Дані, що потрапили у вікно $t_{resample}$, ще не були візуалізовані, і можуть бути згруповані з вже наявними у цій групі даними або ж самі з собою. Їхня часова мітка має бути між поточним часом та часом, в який дані ще можуть пройти ресемплінг.

Таке розділення забезпечує логічну сегментацію даних і дає змогу керувати оновленням без зміни раніше відображених фрагментів.

Виконання ресемплінгу нових даних. На цьому кроці виконується даунсемплінг тільки для групи R. Мета – зменшити кількість даних, зберігаючи основні тренди. Ресемплінг може виконуватися наступними способами:

- середньозважене агрегування (mean): заміна групи значень середнім;
- метод екстремальних точок: збереження максимуму і мінімуму за інтервал;
- рівномірна вибірка: залишаються лише точки з певною частотою.

Результатом є новий піднабір точок, які будуть візуалізовані як змінна частина графіка.

Аналіз ефективності та адаптація параметрів. Після завершення ресемплінгу виконується аналіз ефективності оброблення:

- оцінюється ступінь зменшення даних (кількість до та після);
- аналізується зміна структури даних (флуктуації, сплески, спади);
- визначається, чи потрібно змінити парметри оброблення.

Адаптація параметрів виконується у разі перевищення допустимого навантаження на систему або при зміні динаміки потоку (наприклад, збільшенні частоти подій). Автоматична зміна є опціональною частиною методу, можливе також виконання виключно на параметрах, що були задані на початку роботи. У такому разі якість роботи залежить від оператора, що налаштовував кожну окрему систему, проте не відбувається витрата ресурсів на обчислення нових можливих параметрів.

Збереження результатів у буферах. На цьому кроці оброблені дані зберігаються у відповідні буфери пам’яті:

- C (стабільні), що залишаються незмінними у візуалізаційному шарі;
- R (нові), на основі яких оновлюється змінна частину графічного буфера;
- D (застарілі), що очищаються з оперативної пам’яті та прибираються з відображення.

Це дає змогу системі працювати стабільно, не перевантажуючи графічне ядро та не допускаючи повторного відображення старих фрагментів, що вже не мають цінності.

Відображення даних на графіку. Графік оновлюється з урахуванням:

- стабільних (C) точок, які залишаються візуально незмінними;
- змінних (R) точок, які додаються до існуючих в рамках відповідного часового вікна на графіку;
- очищення зображення від застарілих фрагментів (D) у випадку потреби за наявності таких точок.

Оновлення може відбуватися за своїм окремим таймером, що забезпечує плавність і стабільність візуалізації в реальному часі у разі, якщо

кількість оновлень відображуваних даних на секунду є більшою ніж практично необхідна для відображення в графічному інтерфейсі користувача.

Очищення застарілих даних. Особливістю алгоритму є фонове видалення застарілих даних, яке відбувається незалежно від наявності нових вхідних даних. Цей механізм:

- постійно перевіряє буфер C на наявність точок, що перевищують $t_{current} - t_{display}$;
- видаляє їх із пам'яті та візуалізаційного буфера;
- забезпечує збереження лише актуальної інформації.

Цей підхід є важливим для систем реального часу, у яких дані мають обмежений термін актуальності, а затримки в надходженні не повинні порушувати логіку відображення і при цьому кількість даних, що надходить, може займати значну частину наявної оперативної пам'яті. Схема цього алгоритму зображена на рис. 3.2.

Забезпечення стабільності відображення. Стабільність відображення досягається через розмежування даних на стабільні та нові змінні групи з обмеженням періоду даунсемплінгу [66]. Після закінчення періоду, що входить у часове вікно між поточним часом та часом ресемплінгу, дані переходять у стабільну групу, і надалі не змінюються. Це гарантує, що при постійному оновленні інформації вже візуалізовані елементи залишатимуться стабільними та не спричинятимуть зайвого когнітивного навантаження для користувачів.

Параметри даунсемплінгу та їх вибір. Параметри $t_{resampling}$ та частота ресемплінгу визначаються залежно від:

- швидкості надходження даних;
- необхідної деталізації візуалізації;
- доступних ресурсів (CPU, GPU);
- особливостей галузі застосування (IIoT, Smart City, eHealth).

Оптимальні значення параметрів встановлюються емпірично під час тестування системи і можуть бути адаптивно змінюваними під час роботи залежно від поточного навантаження та обсягу даних.

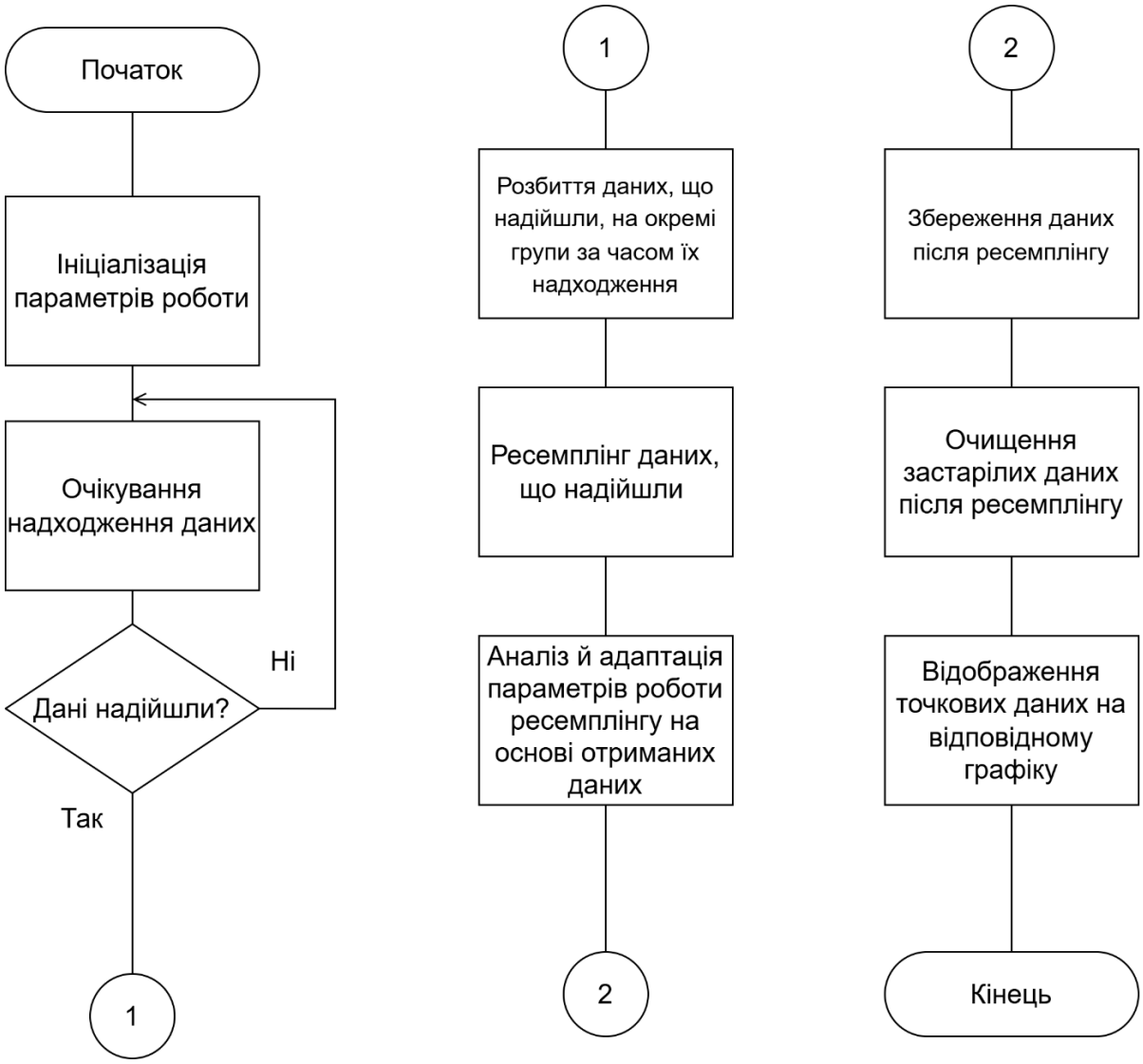


Рис. 3.2. Блок-схема алгоритму відображення потокових даних з їх ресемплінгом та зберіганням візуалізації попередньо відображених даних

Таким чином, запропонований метод забезпечує суттєве зниження обчислювального навантаження та когнітивного перевантаження завдяки зменшенню обсягів відображуваних даних при одночасному забезпеченні високої стабільності візуалізації. Варто зауважити, що сновна особливість роботи з мультимодальними даними полягає в тому, що кожен тип даних

має власну частоту дискретизації, рівень шуму та специфіку часових затримок. Відеопотоки характеризуються високою інтенсивністю даних, аудіосигнали вимагають аналізу в частотній області.

Основою трансформації безперервних мультимодальних потоків у дискретне точкове представлення є виділення значущих подій для їх подальшого оброблення за необхідності оброблення не дискретних даних. Замість розроблення нових методів виділення даних, у межах запропонованого підходу пропонується використовувати існуючі спеціалізовані функції детекції подій, оскільки сучасні програмні бібліотеки вже надають інструментарій для більшості типів даних. Кожна модальність (відео, аудіо тощо) за потреби обробляється відповідним детектором, який виділяє факти зміни стану системи та перетворює їх у дискретні точки з часовими мітками. Такий підхід дозволяє уніфікувати гетерогенні дані різної природи та відображати їх на спільній часовій осі в межах активної та стабільної зон візуалізаційного буфера. Використання дискретних подій замість «сирих» сигналів значно знижує обчислювальне навантаження на графічну підсистему, оскільки візуалізуються лише значущі моменти моніторингу.

Це робить його перспективним рішенням для широкого кола застосувань IoT, де необхідна оперативна і точна робота з потоковими темпоральними мультимодальними даними.

3.3. Програмна реалізація розробленого методу

Згідно із сформульованими вимогами та концепцією розробленого методу візуалізації темпоральних мультимодальних потокових даних IoT із використанням даунсемплінгу, було створено програмну реалізацію, яка демонструє практичну можливість та ефективність застосування методу. Основними цілями реалізації є забезпечення продуктивності оброблення та відображення великих потоків даних, мінімізація навантаження на ресурси комп’ютерної системи та підтримка стабільності візуального представлення

даних. Скріншот з реалізації для графіку на 3D площині представлений на рис. 3.3.

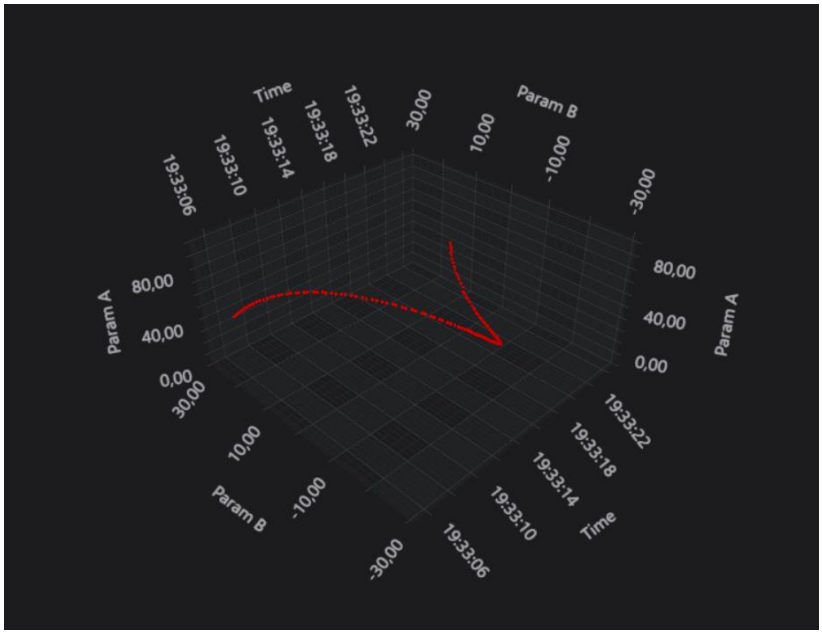


Рис. 3.3. Приклад реалізації графіку відображення темпоральних точкових даних на 3D площині

Для розроблення програмної реалізації запропонованого методу використано такі технології:

– .NET (C#) – вибір платформи зумовлений широкими можливостями для реалізації методів візуалізації, асинхронного оброблення даних та підтримки сучасних бібліотек для роботи з потоковими даними IoT;
– SciChart – потужна бібліотека для створення високопродуктивних 2D та 3D графіків на платформі .NET. Вибір SciChart забезпечує ефективне відображення великих обсягів даних із високою швидкодією та низьким навантаженням на ресурси CPU та GPU та можливість реалізації на багатьох інших платформах, для яких наявні реалізації SciChart або інших бібліотек для роботи з графіками;
– Reactive Extensions (Rx.NET) – для реалізації реактивної моделі оброблення потокових даних і забезпечення високої продуктивності при

великій кількості подій, що надходять від пристроїв IoT. Також доступні реалізації на інших платформах.

Загалом, завдяки використанню в реалізації бібліотек, що мають реалізації для інших платформ, можливо стверджувати про можливість створення реалізації на багатьох платформах, а не лише для .NET. Для реалізації функцій трансформації безперервних потоків у дискретні точкові дані сучасний стек технологій надає широкий спектр готових інструментів. Наприклад, в екосистемі .NET існують бібліотеки OpenCvSharp та Emgu CV для аналізу відео контенту, NAudio та CSCore для детекції аудіо подій, а для Python розроблено окремі спеціалізовані рішення, такі як PySceneDetect для детекції змін сцен відео даних, бібліотеки audio-slicer та rhasspy-silence для виділення активності в аудіосигналах, а також SciPy та Ruptures для точного пошуку піків і точок зламу в часових рядах. Проте також варто зауважити і те, що вищезначені інструменти не є єдиними можливими – для відображення даних існують також й інші бібліотеки, а сам .NET за останні роки адаптований не лише для виконання на win-x86, а й для linux.

Розглянемо компоненти програмного забезпечення для реалізації розробленого методу.

Модуль отримання потокових даних (Data Receiver) асинхронно отримує дані з різних джерел IoT-пристроїв та забезпечує попередню уніфікацію отриманих даних у загальний формат.

Модуль групування та фільтрації даних (Data Grouper) виконує первинне розподілення даних на групи (застарілі, стабільні, нові змінні дані) та здійснює попередню фільтрацію з урахуванням встановлених часових меж.

Модуль даунсемплінгу (Data Resampler) реалізує алгоритм вибіркового зменшення кількості даних [67-72] та використовує налаштовувані параметри для ресемплінгу (наприклад, середнє значення, рівномірний інтервал).

Модуль зберігання та передачі даних (Data Buffer) зберігає стабільні та ресемплінговані дані у проміжному сховищі (буфері) та взаємодіє з модулем візуалізації для передачі готових до відображення даних.

Модуль візуалізації (Visualization Module) використовує бібліотеку SciChart для візуалізації даних у вигляді інтерактивних графіків [73, 74] та забезпечує динамічне оновлення графіків без значних візуальних змін стабільних ділянок.

Для ефективного опрацювання великих потоків даних у режимі реального часу застосовано реактивний підхід, реалізований за допомогою бібліотеки Rx.NET. Такий підхід дає змогу асинхронно обробляти потоки подій, надаючи широкий інтерфейс доступних методів для реалізації складних логічних ланцюжків оброблення даних.

Приклад використання Rx.NET для ресемплінгу даних наведений у лістингу 3.1.

Лістинг 3.1. Використання Rx.NET для буферизації даних

```
var resampledStream = dataStream
    .Buffer(TimeSpan.FromSeconds(resampleInterval))
    .Select(bufferedPoints => new DataPoint
    {
        Timestamp = bufferedPoints.Last().Timestamp,
        Value = bufferedPoints.Average(p => p.Value)
    });
```

Таким чином, створюється потік даних, який передає усереднені значення за заданий інтервал часу, суттєво зменшуючи навантаження на систему, не втрачаючи важливої інформації. Наведена реалізація є найпростішою можливою для усереднення усіх даних за певний інтервал. Для запропонованої і протестованої надалі реалізації застосовувалося об’єднання даних за їхньою відстанню одне від одного, що дає змогу відображати декілька точок даних за певний інтервал, а не лише одну усереднену. Додатково варто відзначити, що Rx.NET підтримує інтеграцію з LINQ, що надає широкі можливості для гнучкого оброблення потокових даних. Це дає змогу безпосередньо в реактивному конвеєрі виконувати

фільтрацію, групування, сортування, агрегування та інші операції, характерні для LINQ-запитів. Такий підхід дає змогу будувати складні сценарії оброблення без проміжного збереження даних, забезпечуючи високу продуктивність і зручність у реалізації логіки, необхідної для підготовки даних до візуалізації.

Основні особливості розробленого програмного модуля полягають у наступному:

- гнучке налаштування параметрів ресемплінгу – користувач може адаптивно змінювати часові межі та частоту ресемплінгу відповідно до умов роботи та характеристик потоків даних;
- інтерактивність та стабільність – завдяки використанню SciChart та реалізованому методу досягнуто високої швидкості оновлення та стабільності візуалізації, що сприяє комфортному та ефективному аналізу інформації;
- оптимальне використання ресурсів – завдяки використанню асинхронних реактивних потоків, що проводять операції групування даних навантаження на процесор і графічний адаптер суттєво знижено за рахунок зменшення кількості відображуваних даних.

Розроблена реалізація може бути інтегрована у різноманітні галузі IoT. Наприклад, в системах індустріального Інтернету речей (IIoT) реалізація дає змогу ефективно моніторити роботу обладнання, виявляючи аномалії у реальному часі з мінімальними ресурсними затратами. Також може бути доцільно застосувати розроблене програмне забезпечення для моніторингу транспортних потоків та екологічних показників із високою деталізацією, але без інформаційного перевантаження.

3.4. Оцінювання ефективності розробленого методу

Для перевірки та оцінювання ефективності розробленого методу візуалізації потокових темпоральних мультимодальних даних IoT було проведено експериментальні дослідження з використанням спеціально

створеного тестового проєкту. Основними цілями тестування були визначення рівня навантаження на обчислювальні ресурси та оцінювання стабільності візуального представлення даних. Також проведено визначення оптимальних параметрів даунсемплінгу для різних умов роботи.

Тестування проводилось на комп'ютерній конфігурації з такими характеристиками:

- операційна система: Windows 10 x64;
- процесор: Intel Core i7-7700HQ;
- оперативна пам'ять: 16 GB DDR4;
- відеокарта: NVIDIA GeForce GTX 1050 4GB;
- програмне забезпечення: .NET 8, SciChart for WPF, Rx.NET, BenchmarkDotNet (для вимірювання продуктивності) [75].

Експериментальні дослідження передбачали порівняння роботи двох варіантів візуалізації. Без застосування даунсемплінгу – усі отримані дані повністю відображаються на графіках без фільтрації. Та із застосуванням запропонованого методу даунсемплінгу – дані попередньо агрегуються та вибірково об'єднуються перед візуалізацією.

Вимірювання проводились при різних значеннях кількості даних, що надходять за одиницю часу (від декількох тисяч до понад 800 тисяч надходжень даних за хвилину). Параметри даунсемплінгу налаштовувалися емпірично відповідно до роботи на конкретному наявному стенді.

Результати порівняння навантаження на CPU представлені на рис 3.4.

Отримані результати, представлені в табл. 3.1 та табл. 3.2, демонструють значне зниження навантаження на процесор при застосуванні розробленого методу візуалізації з даунсемплінгом.

З отриманих результатів можна зробити висновок, що при застосуванні ресемплінгу навантаження було меншим завдяки меншій кількості даних, що необхідно було відобразити. Також варто зауважити, що в навантаження на CPU також входить і обчислення власне ресемплінгу

даних, а відповідно навантаження від його використання значно менше, ніж звільнені ресурси від зменшення кількості відображуваних даних [76].

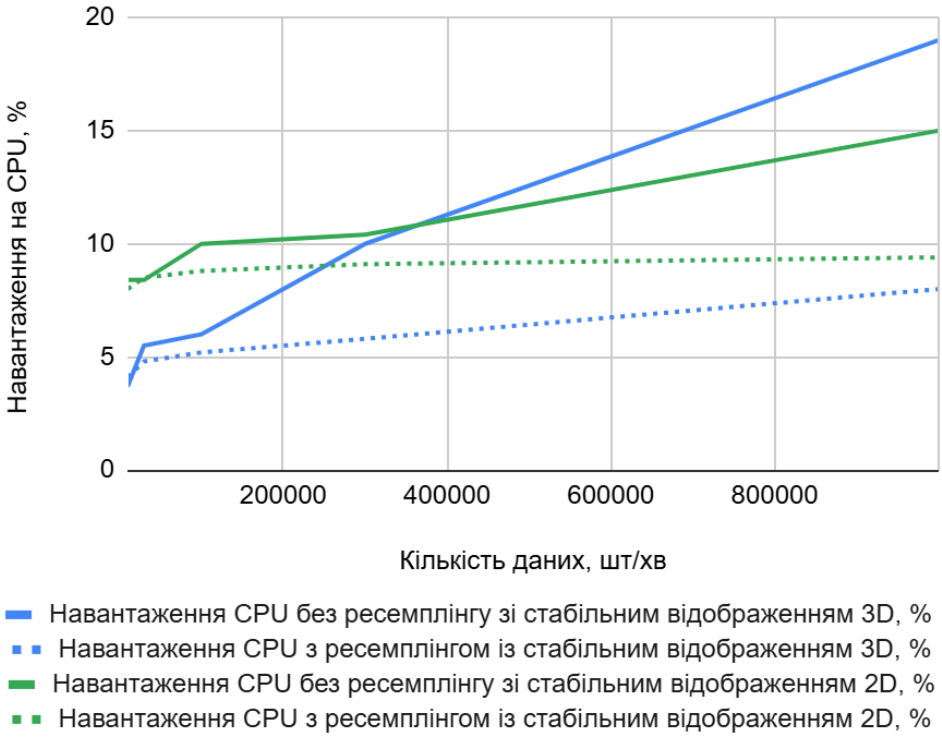


Рис. 3.4. Результати перевірки реалізації для CPU

Таблиця 3.1. – Навантаження CPU при 3D-відображенні даних

Кількість даних, шт/хв	**Навантаження CPU без ресемплінгу, %**	**Навантаження CPU з ресемплінгом, %**
10 000	3,7	4,1
30 000	5,5	4,8
100 000	6	5,2
300 000	10	5,8
1 000 000	19	8

Таблиця 3.2. – Навантаження CPU при 2D-відображенні даних

Кількість даних, шт/хв	**Навантаження CPU без ресемплінгу, %**	**Навантаження CPU з ресемплінгом, %**
10 000	8,4	8
30 000	8,4	8,5
100 000	10	8,8
300 000	10,4	9,1
1 000 000	15	9,4

Результати порівняння навантаження на GPU представлені на рис. 3.5. Застосування даунсемплінгу дало змогу значно знизити навантаження на GPU. Особливо помітна перевага спостерігається при високій частоті отримання даних, що робить запропонований метод особливо актуальним для використання у великих розподілених IoT-системах.

Проведені вимірювання, представлені в табл. 3.3 та табл. 3.4 показали, що відображення даних у 2D-площині створює більше навантаження на систему, ніж у 3D-режимі. Це, ймовірно, пов’язано з особливостями реалізації бібліотеки. У 3D-візуалізації використовуються інструменти DirectX, які краще оптимізовані для відображення великої кількості графічних елементів [77–79]. У наведеному прикладі сцена з певною кількістю точок є відносно простою порівняно з типовими сценаріями застосування 3D-відображення, що призвело до меншого навантаження у 3D-режимі, ніж при рендерингу у 2D за допомогою елементів WPF. Це дає змогу зробити висновок, що в окремих випадках доцільно розглядати можливість використання інструментів 3D-графіки навіть для відображення двовимірних даних. Водночас слід враховувати, що такий підхід може ускладнити реалізацію інших елементів інтерфейсу та вимагати додаткових оптимізацій.

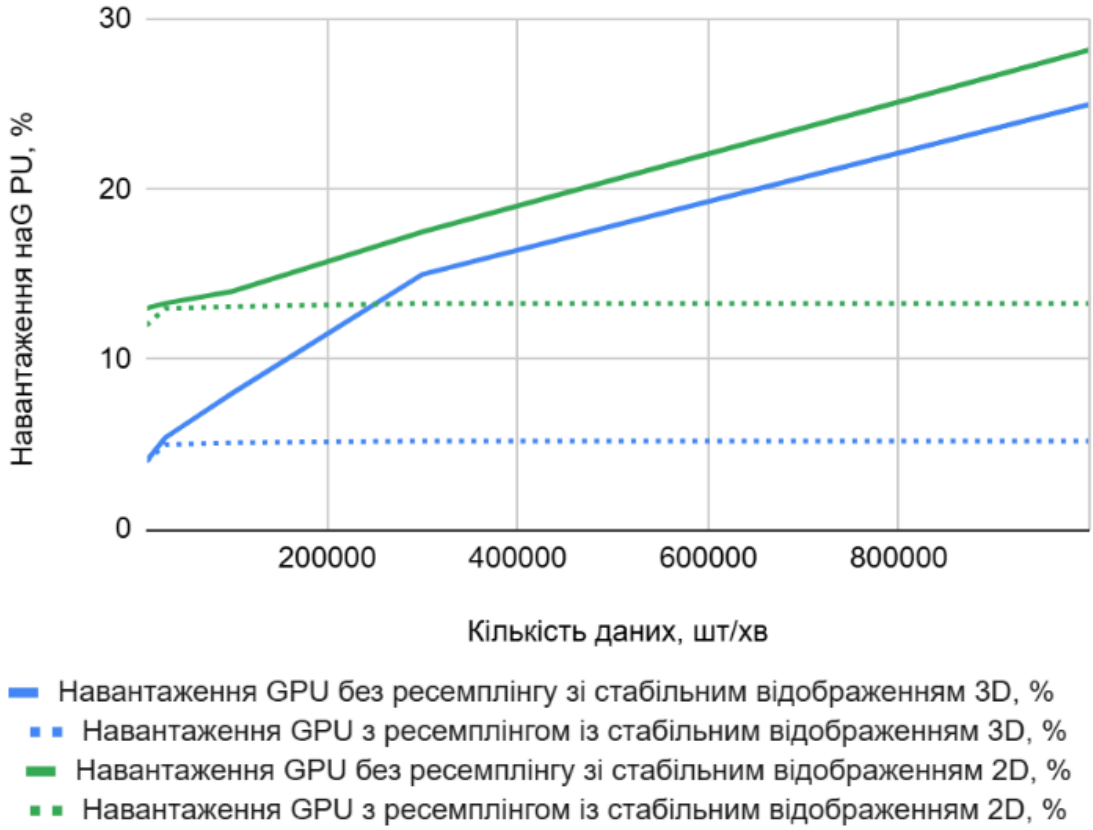


Рис. 3.5. Результати перевірки реалізації для GPU

Таблиця 3.3. – Навантаження GPU при 3D-відображенні даних

Кількість даних, шт/хв	**Навантаження GPU без ресемплінгу, %**	**Навантаження GPU з ресемплінгом, %**
10 000	4	4
30 000	5,4	5
100 000	8	5,1
300 000	15	5,2
1 000 000	25	5,2

Таблиця 3.3. – Навантаження GPU при 3D-відображенні даних

Кількість даних, шт/хв	**Навантаження GPU без ресемплінгу, %**	**Навантаження GPU з ресемплінгом, %**
10 000	13	12
30 000	13,3	13
100 000	14	13,1
300 000	17,5	13,3
1 000 000	28,2	13,3

Окремо проаналізовано стабільність візуалізації. Було зафіксовано, що завдяки чіткому розмежуванню стабільних і змінних даних, візуальні елементи на графіках демонстрували стабільне відображення, а зміни даних з часом не спостерігалося. Візуальна стабільність забезпечувалась за рахунок обмеження інтервалу ресемплінгу. Збереження точності візуалізації оцінювалося за допомогою порівняння схожості отриманого відображення з та без застосування методу через певний час роботи відображення, а саме виконувалося порівняння зображень екрану, внаслідок чого отримані результати показали схожість, що більше або дорівнює 90%.

Проаналізуємо вибір оптимальних параметрів ресемплінгу.

Загалом вбачається раціональним використовувати наступні параметри методу для практичного застосування:

- період ресемплінгу ($t_{resample}$): 1–5 секунд (залежно від швидкості надходження даних);
- максимальний період відображення даних ($t_{display}$): 60–120 секунд (визначається конкретними умовами застосування та потребами користувача, в окремих випадках можливо має сенс відображення впродовж десятків хвилин).

Для встановлення параметрів спершу необхідно оцінити навантаження на систему без використання ресемплінгу, а саме максимальну розрахункову кількість даних, що надходить, для встановлення обмеження по ОЗУ, тобто кількість даних, що теоретично має мати змогу обробити розроблювана система. Надалі необхідно розрахувати максимальний період відображення даних залежно від максимальної кількості даних на секунду. У разі, якщо розрахований час більший, ніж практично необхідний, розрахований час слід вважати максимальним обмеженням для встановлення користувачем, а за стандартний час обрати практично необхідний. Аналогічно необхідно встановити і параметр періоду ресемплінгу, проте у цьому випадку необхідно також провести практичні заміри, а саме, врахувати час на один прохід ресемплінгу та встановити час, який буде у балансі між навантаженням від ресемплінгу та зменшенням навантаження від меншої кількості даних для відображення.

Отримані результати свідчать про ефективність запропонованого методу з точки зору значного зниження навантаження на CPU та GPU, забезпечення стабільності представлення даних, що дає змогу комфортно сприймати і аналізувати інформацію без значного когнітивного перевантаження, а також можливості гнучкого налаштування параметрів для адаптації до конкретних умов роботи.

У процесі проведення експериментальних досліджень та практичного застосування розробленого методу візуалізації темпоральних мультимодальних потокових даних пристроїв IoT було визначено низку переваг і недоліків цього методу. Оцінка сильних та слабких сторін методу дає змогу зробити висновки щодо його доцільності для застосування в різних сценаріях, а також визначити перспективні напрями подальшого вдосконалення.

Основні переваги запропонованого методу даунсемплінгу для візуалізації потокових темпоральних мультимодальних даних IoT є наступними.

1. Зменшення навантаження на обчислювальні ресурси: метод дає змогу значно знизити навантаження на процесор (CPU) та графічний адаптер (GPU), що було підтверджено проведеними замірами. Це особливо важливо для використання в системах із обмеженими апаратними ресурсами або великою кількістю одночасно оброблюваних потоків даних.
2. Забезпечення стабільності візуалізації: запропонований механізм чіткого розмежування стабільних та змінних даних дає змогу уникнути зайвих змін уже візуалізованих елементів, зменшуючи когнітивне навантаження на користувача. Така стабільність є критично важливою в системах, які вимагають швидкого прийняття рішень, зокрема в промислових і медичних застосунках.
3. Гнучкість і адаптивність до різних сценаріїв застосування:
   – метод підтримує гнучке налаштування параметрів даунсемплінгу (частота, інтервали), що дає змогу адаптувати його до специфічних вимог різних галузей (IIoT, Smart City, eHealth тощо);
   – підтримка адаптивного налаштування параметрів забезпечує швидке реагування на зміну умов роботи і характеристик потокових даних.
4. Підтримка інтерактивності: використання високопродуктивної бібліотеки SciChart дає змогу створювати інтерактивні графіки, що дає змогу користувачам деталізувати інформацію та оперативно керувати візуалізацією.
5. Масштабованість: метод забезпечує можливість легкого масштабування систем візуалізації для підтримки великої кількості пристроїв IoT та потоків даних завдяки ефективному зменшенню обсягів оброблюваних даних.

Однак, разом із зазначеними перевагами, виявлено кілька обмежень та недоліків методу, які важливо враховувати під час практичного застосування.

1. Втрата деяких деталей у даних: внаслідок даунсемплінгу при певних параметрах методу деякі деталі в даних можуть бути втрачені або приховані через агрегацію та вибірковий відбір точок. Це може бути критично в галузях, де кожна точка даних є важливою для аналізу, наприклад, при високоточному моніторингу критичних медичних показників.
2. Необхідність ретельного вибору параметрів ресемплінгу:
   - оптимальність методу значною мірою залежить від правильного підбору параметрів, таких як часові межі для ресемплінгу;
   - некоректно підібрані параметри можуть або призвести до втрати важливої інформації, або знизити ефективність методу.
3. Додаткові витрати на обчислення даунсемплінгу:
   - незважаючи на загальне зниження навантаження, процес даунсемплінгу сам по собі потребує певних ресурсів на обчислення, що може створювати додаткове навантаження в разі некоректного налаштування;
   - важливо балансувати витрати на додаткові обчислення та ефект від зменшення обсягів візуалізованих даних.

Можливими шляхами вдосконалення методу є наступні.

Розроблення адаптивних алгоритмів автоматичного налаштування параметрів, а саме, впровадження механізмів автоматичного вибору оптимальних параметрів ресемплінгу залежно від поточних характеристик даних і умов роботи системи.

Інтеграція методів інтелектуального аналізу даних (machine learning), зокрема, застосування алгоритмів машинного навчання для покращення вибору репрезентативних точок у процесі ресемплінгу з метою зменшення втрат інформації або для автоматичного підбору параметрів.

Подальша оптимізація ресурсів, у тому числі, удосконалення алгоритмів і використання більш ефективних підходів паралельного обчислення з метою додаткового зниження навантаження на ресурси.

Таким чином, розроблений метод демонструє значні переваги, що підтверджуються експериментально, і є перспективним рішенням для широкого кола застосувань в IoT. Разом із тим, враховуючи виявлені недоліки та обмеження, важливо застосовувати цей метод з урахуванням конкретних умов і задач, а також розглядати можливість впровадження додаткових механізмів автоматичного налаштування та адаптації.

3.5. Висновки до третього розділу

У третьому розділі дисертаційної роботи було розроблено, обґрунтовано, реалізовано і досліджено метод візуалізації темпоральних мультимодальних потокових даних IoT із використанням технології даунсемплінгу для забезпечення стабільності відображення даних і оптимізації навантаження на обчислювальні ресурси.

На початку розділу було сформульовано ключові вимоги до методу візуалізації потокових IoT-даних, серед яких особливу увагу було приділено стабільності відображення, зниженню навантаження на комп'ютерні ресурси, адаптивності методу та зменшенню когнітивного навантаження на користувачів.

Розроблений метод включає розділення даних на застарілі, стабільні та нові змінні; реалізацію процесу даунсемплінгу (ресемплінгу), що забезпечує значне зменшення обсягів візуалізованих даних при збереженні ключової інформації; забезпечення стабільності візуального представлення шляхом чіткого розмежування між стабільними даними (які не змінюються після первинного відображення) та новими змінними даними.

Для практичної демонстрації методу було створено програмну реалізацію, яка ґрунтується на технологіях .NET, Rx.NET та SciChart. Розроблений програмний модуль підтвердив можливість використання методу в реальних системах IoT, демонструючи високу продуктивність та стабільність візуального представлення інформації.

Проведене експериментальне тестування дало змогу встановити такі ключові результати:

- використання даунсемплінгу суттєво знижує навантаження на CPU та GPU;
- запропонований підхід забезпечує стабільність візуального представлення даних, що зменшує інформаційне та когнітивне перевантаження для користувачів;
- визначено можливі параметри даунсемплінгу, які забезпечують баланс між зменшенням навантаження на ресурси та збереженням важливих деталей даних.

Було також показано, що запропонований метод має широку сферу практичного застосування, зокрема в галузях IIoT, Smart City та eHealth.

Аналіз переваг і недоліків показав, що основними сильними сторонами методу є ефективне використання обчислювальних ресурсів, стабільність візуалізації, адаптивність та інтерактивність. Серед недоліків було визначено потенційну втрату деяких деталей у даних і необхідність ретельного налаштування параметрів. Це визначає перспективи подальшого вдосконалення, зокрема розроблення адаптивних алгоритмів автоматичного налаштування параметрів та інтеграцію методів машинного навчання для покращення точності та якості ресемплінгу.

Отже, розроблений метод візуалізації мультимодальних потокових темпоральних даних із даунсемплінгом є можливим рішенням для практичного застосування у різних сценаріях IoT. Подальші дослідження та розроблення можуть бути спрямовані на розширення його функціональних можливостей, підвищення автоматизації налаштування та оптимізації з метою максимально повного задоволення потреб користувачів і розв’язання задач у різних прикладних сферах Інтернету речей.

РОЗДІЛ 4. РОЗРОБЛЕННЯ УЗАГАЛЬНЕНОЇ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІДОБРАЖЕННЯ ПОТОКОВИХ ДАНИХ ПРИСТРОЇВ IoT

4.1. Загальні вимоги до архітектури програмного забезпечення для роботи з потоковими даними пристроїв IoT

Системи IoT генерують значні обсяги гетерогенних даних у режимі реального часу, що зумовлює необхідність проєктування спеціалізованих архітектур програмного забезпечення для їх оброблення, агрегування, зберігання та візуалізації. Потокова природа цих даних, їхня темпоральність, мультимодальність та різнорідність форматів вимагають врахування специфічних вимог при побудові ефективних архітектурних рішень.

Особливістю даних, що генеруються у пристроях IoT, є поєднання таких характеристик [80 – 83]:

- висока швидкість надходження (у деяких сценаріях – десятки або сотні тисяч подій за секунду);
- великий обсяг – щоденний масив даних може сягати терабайт і більше навіть у середніх за масштабом системах;
- гетерогенність форматів і протоколів – різні пристрої використовують власні схеми структурування даних, протоколи зв’язку та кодування повідомлень;
- темпоральна прив’язка – майже всі дані містять часові мітки, але точність і синхронізація годинників пристроїв можуть суттєво відрізнятися;
- мультимодальність – комбінування різних типів даних (наприклад, поєднання відеопотоку з телеметрією та аудіоінформацією).

У таких умовах класичні підходи до оброблення та візуалізації даних часто виявляються неефективними. Без попередньої стандартизації, групування та оптимізації потоки даних швидко перевантажують канали

передавання та обчислювальні ресурси, що призводить до збільшення затримок і втрати релевантності інформації у режимах, де потрібна реакція у реальному часі.

До основних проблем, що виникають під час роботи з потоковими темпоральними мультимодальними даними IoT, належать наступні.

1. Перевантаження каналів та обчислювальних систем через високий обсяг вхідної інформації, яка часто містить надлишкові або малозначущі дані.
2. Відсутність уніфікованих форматів подання, що ускладнює інтеграцію пристроїв різних виробників у єдину систему.
3. Складність масштабування – зі збільшенням кількості пристроїв традиційні централізовані рішення втрачають ефективність.
4. Високі вимоги до затримки – у промислових, медичних або транспортних сценаріях навіть кількасекундна затримка може призвести до критичних наслідків.
5. Потреба в інтерактивній візуалізації – користувач повинен мати змогу змінювати параметри відображення під час виконання програми, швидко фільтрувати дані, збільшувати масштаб окремих ділянок і повертатися до історичних даних без порушення роботи системи.

Враховуючи вищезазначене, постає потреба у створенні узагальненої архітектури, яка би була основою для побудови програмного забезпечення відображення IoT-даних та забезпечувала наступні можливості:

– гнучке підключення та взаємодію з різнорідними пристроями, незалежно від їх виробника, типу та протоколу зв’язку;
– стандартизацію та попереднє оброблення даних на ранніх етапах надходження, щоб зменшити навантаження на подальші блоки;
– ефективне групування та агрегування за часовими інтервалами, географічними ознаками або іншими критеріями, що підвищує інформативність та знижує надлишковість даних;
– масштабованість і стійкість до збоїв;

– можливість багаторівневої та мультимодальної візуалізації з інтерактивними інструментами аналізу;
– підтримку різних варіантів розгортання (централізованого або децентралізованого), що дає змогу адаптувати програмне забезпечення до конкретних умов експлуатації.

Таким чином, при проєктуванні узагальненої архітектури програмного забезпечення відображення потокових темпоральних мультимодальних даних, виділено наступні вимоги, які розроблювана архітектура має забезпечити.

1. Надійний та стандартизований збір даних з різних IoT-пристроїв.
2. Оброблення та групування у режимі реального часу.
3. Інтерактивна візуалізація з можливістю масштабування та гнучкого налаштування.
4. Сумісність із сучасними протоколами обміну повідомленнями та інструментами оброблення потоків даних.

Архітектура програмного забезпечення повинна мати модульну організацію з чітко визначеними функціональними блоками, кожен з яких відповідає за окремий етап оброблення даних (збір, нормалізація, групування, візуалізація тощо). Такий підхід підвищує гнучкість, спрощує тестування та масштабування та забезпечує можливість незалежного оновлення компонентів. Модульність є основою для побудови мікросервісних систем, які останнім часом набули широкого поширення у глобальних реалізаціях IoT- рішень (наприклад, AWS IoT, Azure IoT Hub, Bosch IoT Suite).

Потокове оброблення даних у реальному часі передбачає використання механізмів оброблення подій (event-driven systems) або реактивного програмування. До основних характеристик потокового оброблення входять низька затримка між отриманням і обробленням події, підтримка асинхронного та паралельного оброблення. Оскільки кількість пристроїв та обсяг даних у ПЗ IoT може стрімко зростати, архітектура

повинна легко масштабуватися як вертикально (збільшення ресурсів серверів), так і горизонтально (додавання нових вузлів або мікросервісів). Програмне забезпечення повинне підтримувати як централізовані, так і децентралізовані конфігурації, залежно від умов експлуатації (географічної розподіленості, кількості вузлів тощо).

Архітектура повинна передбачати можливість швидкої адаптації до зміни умов, середовищ і вимог, зокрема, підтримки різних типів пристроїв (давачі різних виробників), можливість роботи з різними протоколами передавання даних (HTTP, CoAP, MQTT, WebSockets тощо), налаштовувані параметри оброблення та візуалізації, а також можливість конфігурації компонентів без перезавантаження системи.

Програмне забезпечення повинне функціонувати без збоїв навіть у разі втрати окремих вузлів або каналів зв’язку. Досягнення цієї мети може бути забезпечено завдяки використанню буферів повідомлень і брокерів (Kafka, RabbitMQ), дублюванню ключових сервісів, автоматичному перезапуску компонентів при збої та балансуванню навантаження і реплікації даних.

Оброблення темпоральних даних передбачає точне та узгоджене опрацювання міток часу. Для забезпечення цього архітектура повинна забезпечувати упорядкування подій та коректне оброблення даних, що надійшли з запізненням або раніше поточного часу. Цей принцип важливий при візуалізації у вигляді часових графіків, heatmap-аналізу або 3D-інтерфейсів, які враховують динаміку параметрів у часі.

Архітектура має передбачати інтерфейс (API або подієво-орієнтований канал), за допомогою якого блок візуалізації отримує актуальні, підготовлені для графічного представлення дані. Цей інтерфейс повинен бути мінімально залежним від формату «сирих» даних, підтримувати різні варіанти відображення отриманих даних, а також бути узгодженим із механізмом користувацьких фільтрів та запитів.

Таким чином, побудова ефективної архітектури для роботи з потоковими темпоральними мультимодальними даними IoT повинна базуватися на модульності, потоковості, масштабованості, адаптивності, відмовостійкості, часовій узгодженості та готовності до інтеграції з системами візуалізації.

4.2. Функціональні та нефункціональні вимоги

Проєктування узагальненої архітектури програмного забезпечення відображення потокових темпоральних мультимодальних даних IoT вимагає чіткого визначення вимог, що забезпечать її ефективність, масштабованість, гнучкість та сумісність із різноманітними сценаріями використання [84 – 90]. У контексті роботи з великими обсягами даних у реальному часі ключові вимоги можна поділити на функціональні та нефункціональні.

Функціональні вимоги наведені у табл. 4.1.

Таблиця 4.1. – Функціональні вимоги до програмного забезпечення IoT

Код вимоги	Назва вимоги	Опис вимоги
1	2	3
ФВ1	Підтримка різнорідних джерел даних	Архітектура має забезпечувати інтеграцію з широким спектром IoT- пристроїв – від простих давачів до комплексних систем збору мультимодальної інформації. Вона повинна підтримувати різні формати даних (JSON, XML, двійкові потоки, специфічні протоколи виробників) та здійснювати їх уніфікацію на рівні блоку роботи з пристроями.

Продовження табл. 4.1

1	2	3
ФВ2	Стандартизація та попереднє оброблення	У процесі надходження даних має відбуватися перетворення їх до узгодженого формату, приведення до одного формату часових міток, фільтрація шумових даних, та виявлення помилкових значень.
ФВ3	Темпоральне групування та агрегування	ПЗ повинне підтримувати формування часових вікон (fixed windows, sliding windows, session windows), групування даних за часовими та додатковими ознаками (географічними координатами, типами пристроїв, каналами вимірювання), а також обчислення агрегатних показників (наприклад середнє, максимум, мінімум, дисперсія).
ФВ4	Інтерактивна візуалізація у реальному часі	Архітектура повинна передбачати можливість динамічного вибору параметрів для відображення, фільтрації та масштабування графіків, синхронного перегляду 2D та 3D візуалізацій, а також перемикання між історичними та поточними даними без порушення роботи основного потоку.
ФВ5	Гнучка інтеграція з аналітичними модулями	ПЗ має підтримувати обмін даними з аналітичними сервісами (машинне навчання, прогнозування, виявлення аномалій), щоб доповнювати візуалізацію результатами автоматичного аналізу.

Нефункціональні вимоги наведені у табл. 4.2.

Аналіз функціональних та нефункціональних вимог дає змогу сформулювати цілісний набір вимог, які повинна задовольняти узагальнена архітектура програмного забезпечення відображення потокових темпоральних мультимодальних даних IoT.

Таблиця 4.2. – Нефункціональні вимоги до програмного забезпечення IoT

Код вимоги	Назва вимоги	Опис вимоги
НФВ1	Масштабованість	Архітектура повинна ефективно працювати як у малих, так і у великих розподілених інфраструктурах. Масштабування повинно бути можливим як вертикальним (збільшення ресурсів окремого вузла), так і горизонтальним (додавання нових вузлів або сервісів).
НФВ2	Низька затримка оброблення	Для сценаріїв моніторингу в реальному часі допустима затримка від моменту надходження даних до їх візуалізації не повинна перевищувати кількох сотень мілісекунд для критичних систем і 1–2 секунд для загальних.
НФВ3	Відмовостійкість та надійність	Архітектура має забезпечувати стабільну роботу навіть у разі збою окремих компонентів. Необхідна підтримка механізмів реплікації, автоматичного перемикання на резервні вузли, збереження проміжних результатів.
НФВ4	Сумісність з комунікаційними протоколами IoT	Підтримка стандартів MQTT, Apache Kafka, CoAP, HTTP/HTTPS є важливою для інтеграції у різноманітні мережеві середовища та сумісності з комерційними і відкритими рішеннями.
НФВ5	Модульність і гнучкість	Архітектура повинна давати змогу додавання, оновлення або заміну окремих модулів без зупинки роботи всього ПЗ. Це особливо важливо для динамічних середовищ, де вимоги до оброблення та візуалізації можуть змінюватися.
НФВ6	Безпека та контроль доступу	ПЗ повинне підтримувати шифрування даних при передаванні, автентифікацію та авторизацію користувачів, контроль доступу на рівні даних і сервісів.

4.3. Розроблена узагальнена архітектура програмного забезпечення

Враховуючи загальні принципи побудови архітектур для роботи з потоковими темпоральними мультимодальними даними IoT, розроблена узагальнена архітектура є орієнтованою на інтерактивну та стабільну візуалізацію великої кількості темпоральних гетерогенних потокових даних у реальному часі. Основною метою побудови архітектури є забезпечення масштабованості та ефективного розділення відповідальності між компонентами програмного забезпечення. Це дає змогу ефективно обробляти дані незалежно від кількості джерел, формату повідомлень або кінцевих сценаріїв візуалізації або іншого використання.

Узагальнена архітектура програмного забезпечення відображення потокових темпоральних мультимодальних даних IoT являє собою багаторівневу гнучку структуру. Вона забезпечує повний цикл роботи з даними – від їхнього отримання від пристроїв до інтерактивної візуалізації для кінцевого користувача [91]. Архітектура розроблена з урахуванням вимог, сформульованих у п. 4.1, і складається з трьох основних функціональних блоків: блока роботи з пристроями IoT, блока групування даних та блока відображення точкових темпоральних даних.

Між зазначеними блоками функціонує система уніфікованих каналів взаємодії, яка може бути реалізована як через внутрішньосистемні API, так і через стандартні засоби передавання даних (MQTT, Apache Kafka, HTTP/HTTPS, CoAP).

Розглянемо кожен з блоків розробленої архітектури (рис. 4.1).

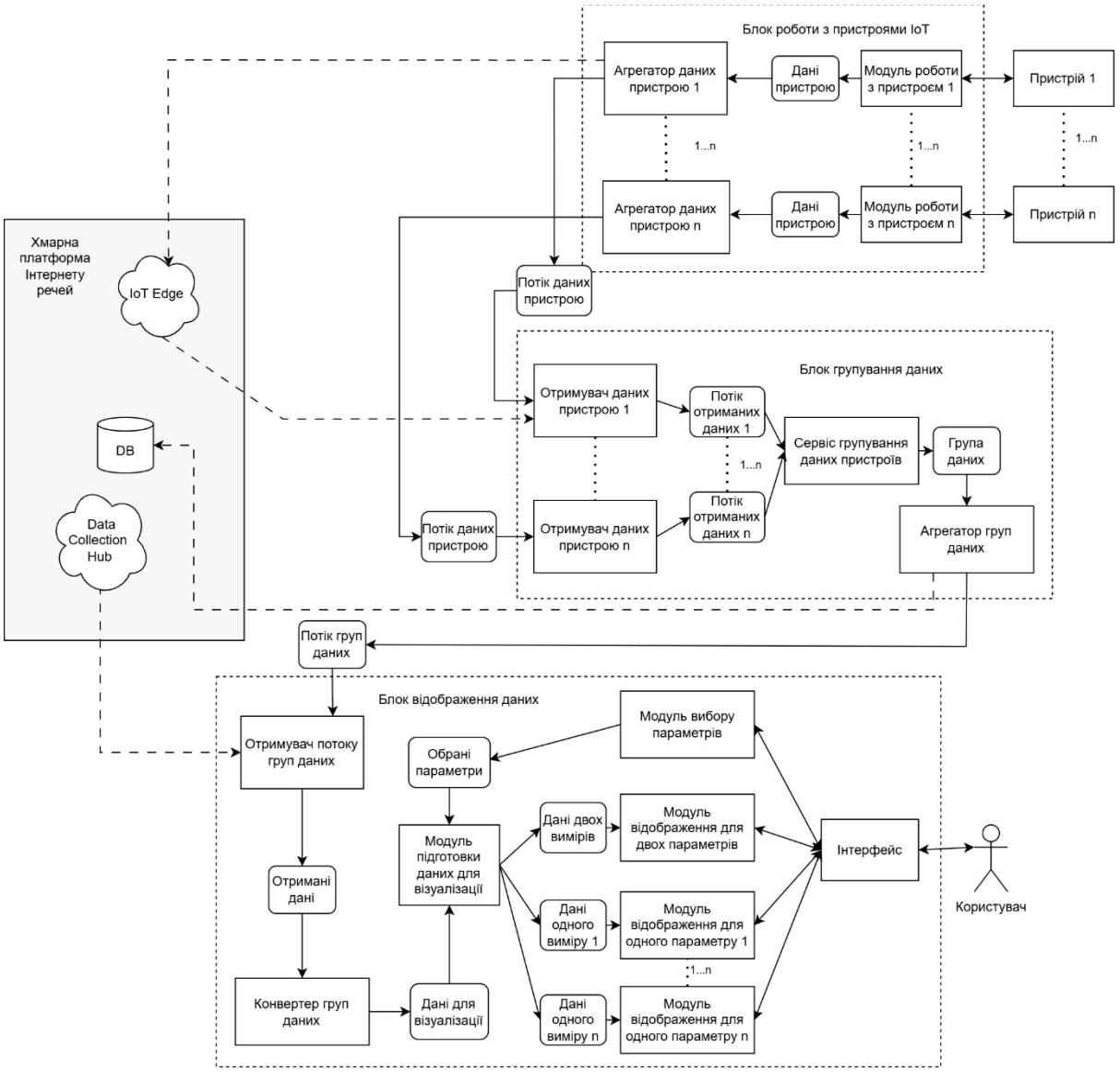


Рис. 4.1. Узагальнена архітектура програмного забезпечення відображення потокових темпоральних даних пристроїв IoT

Розглянемо *блок роботи з пристроями IoT*. Його призначення полягає у забезпеченні безпосереднього з’єднання з IoT-пристроями, зборі первинних даних (ФВ1), їх можливому попередньому обробленні та стандартизації для подальшої передачі іншим компонентам програмного забезпечення. У сучасних високонавантажених системах реалізація цього блоку базується на принципах граничних обчислень (Edge Computing). Це передбачає перенесення логіки первинного аналізу та фільтрації ближче до джерела даних, що надає змогу значно зменшити затримку та розвантажити

канали зв’язку. До основних функцій та особливостей цього блоку належать підключення до пристроїв через підтримувані комунікаційні протоколи (дротові та бездротові інтерфейси), уніфікація формату даних для усунення несумісностей (ФВ2), пов’язаних із різними виробниками обладнання та різними форматами повідомлень, валідація даних із перевіркою коректності значень, відсівом некоректних або пошкоджених даних і фільтрацією шумових даних, а також синхронізація часових міток шляхом перетворення їх у єдиний формат з урахуванням можливих відмінностей, пов’язаних з використанням різних форматів, часових зон та передачею часових міток. Окремо виконується оптимізація обсягу переданих даних, що включає ущільнення, видалення надлишкових параметрів і, за необхідності, попереднє агрегування. Після попереднього оброблення та стандартизації дані передаються в блок групування даних у вигляді уніфікованих повідомлень з часовими мітками та іншими можливими атрибутами метаданих, таких як назва пристрою тощо. Реалізація цього блоку може базуватися на використанні платформи Azure IoT Edge, що надає можливість розгортати логіку оброблення у вигляді автономних контейнеризованих модулів безпосередньо на граничних шлюзах. Такий підхід забезпечує виконання критично важливих операцій з валідації та фільтрації поблизу джерел даних, після чого нормалізовані потоки спрямовуються у відповідні хмарні сервіси, такі як Azure IoT Hub або AWS IoT Core. Це перетворює блок роботи з пристроями на ланку інтеграції, яка не лише збирає дані, а й уможливлює їхню сумісність із глобальними промисловими екосистемами та хмарними інфраструктурами потокової аналітики. За такої реалізації підключення до зовнішніх хмарних середовищ реалізується безпосередньо в межах блоку групування даних, де спеціалізований модуль отримання виступає як активний інтерфейс взаємодії з хмарними хабами. Це дозволяє за необхідності розгортати блок групування як автономний хмарний мікросервіс, оброблення потоків даних безпосередньо в масштабованому обчислювальному середовищі.

Блок групування даних призначений для агрегування та структурування потокових даних за часовими чи, альтернативно, також й іншими критеріями (ФВ3), а також для підготовки їх до подальшої візуалізації або іншого використання. У межах цього блоку реалізуються часові групування з використанням часових вікон, що дає змогу об'єднувати дані, отримані в один момент часу або протягом заданого інтервалу. Передбачено виконання агрегації даних із обчисленням середніх, мінімальних, максимальних значень, дисперсій та інших статистичних характеристик у межах груп за необхідності. Важливою функцією є буферизація даних для синхронізації з темпами візуалізації та уникнення перевантажень інтерфейсу користувача. Підготовлені групи даних надалі маршрутизуються у блок відображення через відповідні канали обміну. У взаємодії з іншими блоками цей компонент отримує дані з блоку роботи з пристроями, обробляє їх і надсилає результати в уніфікованому форматі в блок відображення.

Блок відображення точкових темпоральних даних має призначення візуалізувати агреговані дані з наданням користувачеві можливості інтерактивного налаштування параметрів відображення (ФВ4). Можливе отримання даних як від блоку групування даних, так і теоретично можливе використання підключення до хмарного хабу збору даних для використання цього блоку у якості розширення можливостей відображення даних існучих систем. На вході реалізується конвертація вхідних даних у формат, сумісний із графічними компонентами візуалізації. На основі цих даних здійснюється побудова множини 2D-графіків для індивідуального відображення окремих параметрів, а також формування інтегрованого 3D-графіка з фіксованою часовою віссю та можливістю відображення двох параметрів на інших осях. Також в рамках цього блоку реалізується відображення згрупованих подієвих даних за використанням нового типу користувацьких інтерфейсів – подієво-орієнтованих інтерфейсів, з використанням яких можливо працювати власне з подіями як новим типом даних для візуалізації.

Канали обміну та сценарії взаємодії між блоками можуть реалізовуватися по-різному. В одному випадку використовується внутрішньопроцесний обмін у разі централізованої реалізації на одному сервері, наприклад через внутрішні API. В іншому випадку може застосовуватися мережевий обмін повідомленнями, наприклад через MQTT, Apache Kafka та іншими [92–94] у разі децентралізованого розгортання (НФВ4). Цей підхід дає змогу динамічно масштабувати оброблення при зростанні навантаження, забезпечує можливість незалежного оновлення або повторного використання окремих компонентів, а також спрощує інтеграцію нових функцій чи форматів даних, підключення до існуючих аналітичних модулів (ФВ5), а також використання існуючих протоколів дає змогу використовувати їхні вбудовані можливості для шифрованої передачі даних або ж забезпечити безпеку даних на рівні окремої мережі (НФВ6).

Також важливою особливістю є реалізація *збору та агрегації даних* у запропонованій архітектурі. Ці механізми відіграють ключову роль у забезпеченні цілісності, точності та ефективності роботи програмного забезпечення візуалізації IoT-даних. З огляду на багатопотокову та мультимодальну природу інформації, що надходить від пристроїв IoT, запропонована архітектура передбачає чітку багатоступеневу модель оброблення, яка включає модулі для приймання, буферизації, фільтрації, уніфікації та агрегування потоків.

Логіка оброблення потокових темпоральних мультимодальних даних у межах архітектури побудована таким чином, щоб забезпечити мінімальні затримки, збереження релевантності та стабільності візуалізації, а також можливість масштабування системи залежно від кількості підключених пристроїв і обсягів даних. Процес оброблення поділяється на кілька послідовних етапів, які реалізуються взаємопов’язаними блоками архітектури. Діаграма активності даного процесу зображена на рис. 4.2.

Спочатку відбувається отримання та первинне оброблення даних. На початковому етапі блок роботи з пристроями IoT отримує дані від підключених пристроїв у режимі реального часу. Передача здійснюється як у безперервному потоці. Вхідні дані проходять валідацію, що включає перевірку коректності значень, формату та наявності часових міток, очищення від шумів, приведення часових міток до необхідного формату з урахуванням різниці в можливому надходженні міток з різних часових зон та у різному форматі зберігання, а також стандартизацію до уніфікованої структури повідомлення, яка спрощує подальше оброблення.

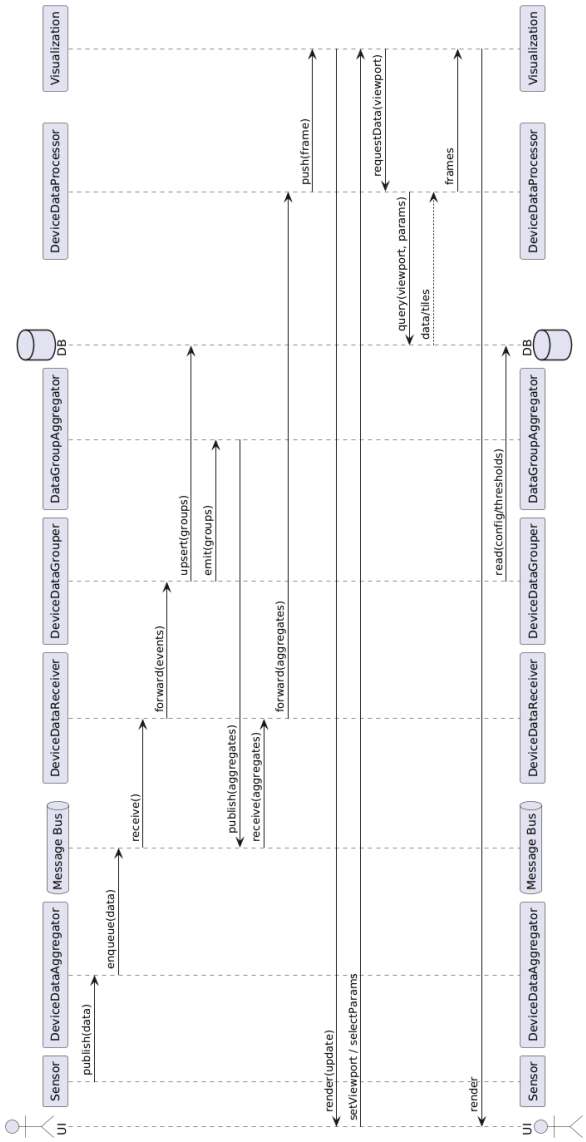


Рис. 4.2. Діаграма активності розробленого програмного забезпечення

Надалі відбувається групування та агрегування потоків. Після первинного оброблення дані надходять у блок групування, де здійснюється їх агрегація за часовою ознакою [95–99]. На цьому етапі виконуються основні процедури, до яких належать формування часових вікон, групування за темпоральним атрибутом, а також буферизація для відправлення окремих пакетів до наступного блоку. Механізм агрегації істотно зменшує обсяг даних, що передаються до візуалізаційного блоку, і завдяки цьому дає змогу уникнути перевантаження інтерфейсу при великій кількості подій, забезпечити стабільність відображення із застосуванням розробленого методу відображення, а також підвищити якість сприйняття графіків користувачем.

На останньому етапі відбувається підготовка даних до візуалізації в блоці візуалізації та взаємодії з користувачем. Цей компонент відповідає за кінцеве представлення оброблених темпоральних мультимодальних потокових даних IoT у формі, зручній для сприйняття користувачем у реальному часі. Блок побудований таким чином, щоб забезпечити максимальну стабільність, низькі затримки та взаємодію користувача з відображеними даними. Блок візуалізації в запропонованій архітектурі виконує отримання даних із блоку групування даних, забезпечує стабільність відображення потокових даних шляхом чіткого розмежування на стабільні та змінні дані, інтерактивну візуалізацію з можливістю зміни відображення, а також забезпечення мінімальних затримок між надходженням даних і їх відображенням на інтерфейсі користувача.

Розглянемо складові блоку візуалізації. У розробленій архітектурі блок візуалізації реалізовано як окремий компонент із внутрішньою структурою, що включає кілька модулів. Модуль отримання даних (Device Data Receiver) підписується на канали передачі агрегованих даних і приймає їх у режимі реального часу. Альтернативно, у разі інтеграції блоку візуалізації до вже існуючої системи (що зображена на схемі як Data Collection Hub), якщо необхідно додати саме можливості візуалізації, в

даному модулі дані можуть отримуватися з хмарних сервісів, таких як, наприклад, Azure IoT Hub [100] та AWS IoT [101] використовуючи стандартні варіанти передачі повідомлень, що дає змогу модулю виступати в ролі незалежного споживача у розподілених хмарних середовищах. Взаємодія з хмарною інфраструктурою може здійснюватися не лише отримуючи «сирі» дані пристроїв, а й після проходження через сервіси потокової аналітики, які виконують роль проміжної ланки для попереднього оброблення даних. Завдяки цьому програмне забезпечення візуалізації здатне ефективно опрацьовувати дані, що вже пройшли етап первинної фільтрації ще на рівні хмарного брокера, знижуючи навантаження на мережу для передачі даних й на сам блок візуалізації. Далі працює модуль трансформації даних (Data Group Converter), який здійснює оброблення останніх отриманих точкових даних, забезпечуючи їх стабільне відображення. Модуль підготовки даних до візуалізації (Data Visualization View Model) відповідає за як 2D, так і 3D-відображення. Інтерактивний інтерфейс користувача (View) дає змогу змінювати параметри візуалізації в режимі реального часу, наприклад, користуватися функціями навігації, такими як масштабування та інші. Така організація блоку візуалізації дає змогу створити компонент для роботи з потоковими темпоральними мультимодальними даними IoT.

Архітектура забезпечує високу продуктивність візуалізації навіть при роботі з великими обсягами мультимодальних потоків. Завдяки попередньому групуванню та застосуванню методу відображення досягається стабільна частота оновлення графіків і низька затримка, що дає змогу працювати з великою кількістю даних. Можливість налаштування параметрів відображення, наприклад вибір кольорових схем, методів агрегації та фільтрації робить архітектуру придатною для промислових, медичних, транспортних та міських аналітичних систем, деякі з яких мають специфічні вимоги до подання інформації, які власне і потребують можливості налаштування та високої продуктивності.

Розглянемо процедуру візуалізації в режимі реального часу. Дані відображаються у вигляді окремих 2D-графіків та 3D-графіку із часовою віссю. Користувач може змінювати параметри відображення, масштабувати й панорамувати зображення, перемикатися між режимами реального часу та, за потреби, переглядом архівної інформації. Архітектура передбачає використання паралельних потоків для роботи з даними при отриманні та групуванні даних, що підвищує пропускну здатність програмного забезпечення й зменшує затримки, особливо за великої кількості даних з багатьох пристроїв. При розподіленому розгортанні можливе багатократне використання окремих блоків, наприклад блоків отримання даних від пристроїв) і маршрутизація даних за географічним принципом (розташування блоків отримання даних безпосередньо поряд з пристроями і подальше надсилання на проміжні сервери), що мінімізує затримки для різних категорій користувачів.

Ефективність роботи узагальненої архітектури програмного забезпечення відображення потокових темпоральних мультимодальних даних IoT значною мірою визначається вибором технологій, що забезпечують стабільне, масштабоване та безпечне оброблення і передавання інформації. Розглянемо технології, що можуть використовуватися для різних імплементацій розробленої архітектури. При розробленні архітектури враховувалися вимоги до низької затримки, сумісності з різними пристроями та протоколами, відмовостійкості, а також можливості інтеграції з аналітичними сервісами.

Для передавання даних між блоками архітектури та інтеграції з пристроями запропоновано для реалізації наступні протоколи.

– MQTT (Message Queuing Telemetry Transport). Використовується для збирання даних від пристроїв із низьким енергоспоживанням або обмеженими обчислювальними ресурсами, що дає змогу за необхідності встановлювати відповідні за енергоспоживанням сервери збору даних від пристроїв. MQTT працює за принципом *publish–subscribe*, що дає змогу

масштабувати програмне забезпечення та підключати нові вузли без зміни конфігурації всієї системи. Перевагами є мала вага протоколу, підтримка нестабільних каналів зв’язку та можливість передавання повідомлень навіть при тимчасових розривах з’єднання;

– Apache Kafka. Використовується для високошвидкісного обміну повідомленнями та оброблення великих обсягів потокових даних у режимі реального часу. Kafka забезпечує горизонтальне масштабування, високу пропускну здатність та надійність зберігання повідомлень завдяки механізму реплікації. Її інтеграція до архітектури дає змогу створювати розподілені конвеєри оброблення даних та підключати аналітичні модулі без втручання в основні робочі процеси.

Загалом, розроблена архітектура не є зафіксованою на одному конкретному протоколі, оскільки також розглядається можливість використання цієї архітектури для систем, що вже певний час розроблювалися і відповідно вже мають обраний протокол обміну даних. Це означає, що має бути змога використання архітектури з різноманітними протоколами.

Для реалізації низьколатентного, масштабованого та відмовостійкого оброблення подій у запропонованій архітектурі використовуються взаємодоповнювальні технології різних рівнів: реактивні бібліотеки для локальних конвеєрів даних та високопродуктивні RPC-протоколи для міжсервісної взаємодії.

Rx.NET доцільно використовувати в межах .NET-компонентів, зокрема у клієнтських застосунках візуалізації та мікросервісах, коли потрібне подієво-орієнтоване оброблення у режимі реального часу з мінімальною затримкою. UI отримує рівномірний, контрольований потік даних, тоді як не агреговані події можуть надходити в значно більшій кількості, що може підвищити навантаження на систему. Rx.NET може інтегруватися з джерелами типу WebSocket або MQTT-клієнтів та з

бібліотеками графіків, даючи змогу будувати локальні конвеєри попереднього оброблення без звернення до зовнішніх даних.

У розробленій архітектурі візуалізація потокових темпоральних мультимодальних IoT-даних ґрунтується на скоординованому використанні високопродуктивних графічних рушіїв, спеціалізованих бібліотек для побудови діаграм і сцен. Вибір нижнього графічного шару фактично задає верхню межу пропускної здатності програмного забезпечення та визначає спектр доступних візуальних ефектів. Для GPU-прискореного 2D та 3D рендерингу використовуються DirectX (Windows), OpenGL, Vulkan (кросплатформене середовище), а також WebGL і WebGPU у браузерних клієнтах. Для задач, пов'язаних із відображенням мільйонів маркерів даних, доцільно надавати перевагу саме GPU-орієнтованому рендерингу зі спеціалізованими шейдерами. У випадках, коли вимоги до інтенсивності обчислень є помірними або апаратне прискорення обмежене, може застосовуватися Canvas2D та Skia. У .NET поширеним рішенням є SkiaSharp для кросплатформеного 2D-рендерингу, а у мультиплатформених UI фреймворках використовується Avalonia з GPU-бекендом.

Щодо бібліотек і фреймворків для побудови 2D та 3D діаграм, у настільних .NET-клієнтах застосовується SciChart (WPF, Avalonia), який забезпечує високопродуктивне відображення 2D та 3D графіків із продуктивністю, співставною з WebGL у десктопному середовищі, а також підтримує великі часові ряди, heatmap, contour-діаграми, 3D-точкові хмари та об'ємні дані. Для задач із помірною інтенсивністю потоків і статичними наборами даних можливим є використання ScottPlot. У веб-клієнтах Three.js виступає як універсальний 3D-рушій (WebGL) для побудови інтегрованих сцен, зокрема з часовими осями, представленими як геометричні об'єкти. Бібліотека D3.js забезпечує гнучку декларативну побудову 2D-візуалізацій, зручне зв'язування даних із DOM і SVG та створення інтерактивних часових осей.

Інтеграція вибраних технологій організована за принципами модульності та незалежності компонентів, що дає змогу без втручання в загальну роботу ПЗ додавати нові інструменти або замінювати існуючі, адаптувати архітектуру до появи нових протоколів і стандартів у сфері IoT, а також поступово масштабувати ПЗ без суттєвих змін у логіці оброблення даних.

Серед особливостей розробленої архітектури можливо виділити високу масштабованість, оскільки вона підтримує як централізовану, так і децентралізовану реалізацію, що робить її придатною для локальних і розподілених IoT пристроїв. Стабільність візуалізації забезпечується завдяки застосуванню розробленого методу відображення. Також можна виділити гнучкість конфігурації, яка дає змогу адаптувати програмне забезпечення до різних джерел даних, типів пристроїв і різноманітних сценаріїв візуалізації. Крім того, вона може інтегруватися в існуючі рішення, оскільки базується на стандартних протоколах і сучасних технологіях.

Розглянемо використання блоків архітектури як самостійних компонентів у різних сценаріях. Запропонована архітектура цілеспрямовано побудована за принципами слабкого зв’язування та чіткої контрактної взаємодії між компонентами. Завдяки уніфікованим інтерфейсам кожен із трьох базових блоків, тобто блок роботи з пристроями, блок групування даних і блок відображення, може застосовуватися ізольовано як самостійний сервіс у складі інших систем. Можливо виділити деякі варіанти такого роз’єднаного використання з урахуванням вимог до обчислювальних ресурсів, затримок, надійності та доменних обмежень.

Ізольоване використання блоку роботи з пристроями IoT. У ролі периферійного конектора блок забезпечує під’єднання до гетерогенних пристроїв, нормалізацію повідомлень, а також початкове фільтрування або ущільнення даних без потреби підключення інших блоків програмного забезпечення. Такий режим є доцільним у низці практичних сценаріїв. У

сфері IIoT промислові контролери та давачі, зокрема вимірювачі вібрацій і температури, передають телеметрію до локального шлюза, де виконується її валідація та знешумлення, після чого події публікуються в корпоративну шину на кшталт Kafka і стають придатними для зовнішніх систем або рішень предиктивного технічного обслуговування. У контексті Smart City, зокрема транспортних систем, придорожні блоки з обмеженими обчислювальними ресурсами можуть попередньо агрегувати дані з лічильників трафіку й детекторів черг та передавати лише узагальнені показники, наприклад, з інтервалами в декілька секунд, до центрального міського дата-центру. У сфері медичного програмного забезпечення домашні збирачі даних пацієнтів виконують уніфікацію телеметрії з конкретних медичних приладів і передають її у зашифрованому вигляді до хмарного середовища, де подальша візуалізація та глибинна аналітика можуть здійснюватися зовнішніми системами моніторингу стану пацієнта. Серед переваг такого застосування можливо виділити зменшення трафіку, мінімізацію затримок за рахунок локального передоброблення, ізоляцію специфіки пристроїв з можливістю використання певного запропонованого інтерфейсу. Можливі недоліки в такому сценарії включають необхідність керування версіями прошивок та адаптерів, витрати на моніторинг і віддалене оновлення.

Ізольоване використання блоку групування даних. Блок може працювати як автономний агрегатор потокових даних, що приймає уніфіковані події з різноманітних джерел (тобто не лише з запропонованого блоку роботи з пристроями, а й з існуючих систем, таких як Azure IoT Hub та при цьому бути розгорнутим у відповідній хмарі для проведення необхідних обчислень), виконує групування, можливий підрахунок статистичних даних, корекції часових зміщень і формує вихідні потоки даних або зберігає їх у сховищі даних для подальшого використання сторонніми застосунками. У задачах аналітики руху та картографії поєднуються дані про пересування транспорту, виконується групування

подій в одному потоці та формуються шари для геосервісів, які використовуються для відображення отриманої інформації. У контексті енергетичних систем можливе об’єднання телеметрії з різних підстанцій у синхронізовані зрізи часу з подальшим розрахунком флуктуацій або пікових навантажень і передаванням цих результатів у системи диспетчеризації.

Ізольоване використання блоку відображення (візуалізаційний клієнт). Блок відображення може бути під’єднаний до сторонніх джерел агрегованих даних (Azure IoT Hub, Azure Event Hub, тощо), виконуючи лише власне візуалізацію й інтерфейсну взаємодію для місця роботи операторів. Підключення до наявних корпоративних потоків даних і графічне відображення 2D або 3D графік у диспетчерських центрах без втручання в підсистеми збору. Також можлива реалізація клінічних інтерфейсів. Відображення часових трендів і сповіщень, що надходять з платформи дистанційного моніторингу, з можливістю інтерактивної фільтрації та змін масштабу без власних конвеєрів оброблення. Застосування в R&D може дати змогу швидке під’єднання фронтенду до тестових потоків даних для перевірки UX рішень або тестування нових методів візуалізації. Серед переваг можливо виділити мінімальні витрати часу порівняно з розробленням візуалізації з нуля, використання наявних потоків даних, однак також можливі такі обмеження, як залежність від якості та затримок зовнішніх потоків даних, потреба в узгоджених API (можлива необхідність створення окремого адаптера).

Гібридні конфігурації. Можливе також поступове комбінування блоків із нарощуванням функцій, коли не використовуються лише окремі блоки. У такому підході, наприклад, можуть використовуватися лише блоки візуалізації та групування – сторонні периферійні модулі виконують отримання, нормалізацію даних, які надалі надходять до блоку групування даних, а надалі саме відображення здійснюється у блоці візуалізації даних. Інший варіант передбачає підключення до пристроїв власним блоком, централізоване групування також власним блоком, а відображення може

відбуватися за допомогою інтеграції зі сторонніми сервісами. Можливе також використання проміжного компонента для оброблення потоків, такого як цифровий двійник. У цьому випадку блок групування подає потоки даних у модель двійника, який виконує також властиві цифровому двійнику додаткові функції, а блок відображення показує стан або прогнози без прямого доступу до початкових потоків даних, отримуючи їх від цифрового двійника.

Подібне роз'єднане використання уможливлює оптимізацію витрат за рахунок шляхом вибіркового масштабування компонентів із обмеженою пропускною здатністю, наприклад, через збільшення кількості блоків групування. Воно також дає змогу пришвидшити впровадження, починаючи роботу з одного блоку й поступово доповнюючи програмне забезпечення іншими без значних змін між версіями продукту, і підвищує стійкість, оскільки відмова одного сервісу не призводить до зупинки всього ПЗ. Водночас важливо врахувати ризики фрагментації, зокрема узгодженість часу між серверами, сумісність API, уніфікацію ідентифікаторів пристроїв, політики безпеки та потребу в централізованому спостереженні, що охоплює логування, метрики й трасування передачі даних при значному навантаженні.

Загалом, модульність і контрактна взаємодія блоків дають змогу використовувати їх як самостійні, повторно придатні елементи. Це розширює спектр практичних впроваджень (від локальних сценаріїв до глобальних розподілених рішень) і зменшує сукупну вартість розроблення завдяки поетапному масштабуванню та еволюційному розвитку ПЗ.

4.4. Аналіз розробленої архітектури

Розроблена узагальненої архітектура програмного забезпечення відображення потокових темпоральних мультимодальних даних IoT поєднує низку технічних та експлуатаційних переваг, що роблять її придатною для широкого спектра прикладних галузей у сфері IoT – від

промислової автоматизації до eHealth та Smart City. Представлена у цьому розділі архітектура програмного забезпечення візуалізації потокових темпоральних мультимодальних даних IoT демонструє як суттєві сильні сторони, так і певні обмеження, які необхідно враховувати при її практичному застосуванні.

До основних переваг такої архітектури належить модульність: вона побудована за модульним принципом, що дає змогу незалежно оновлювати або замінювати окремі компоненти без зупинки роботи всього ПЗ, спрощуючи адаптацію під нові формати даних, протоколи або вимоги користувачів. Важливою є масштабованість, оскільки підтримка як вертикального, так і горизонтального масштабування дає змогу працювати з різними обсягами даних – від невеликих локальних наборів IoT пристроїв до глобальних розподілених інфраструктур. Суттєвою перевагою є сумісність із широким спектром протоколів: підтримка різних стандартів комунікації у загальному вигляді дає змогу реалізовувати частини програмного забезпечення відповідно до запропонованої архітектури навіть у вже частково розроблених рішеннях з обраним протоколом обміну. Стабільність візуалізації даних забезпечується завдяки застосуванню спеціальних алгоритмів ресемплінгу та поділу даних на стабільні й змінні компоненти у запропонованому в попередньому розділі методі відображення, що сприяє мінімізації когнітивного навантаження на користувачів при постійному оновленні потоків. Використання асинхронних механізмів передавання та оброблення даних дає змогу досягти мінімального часу від отримання даних до їх візуалізації, що є критично важливим для сценаріїв реального часу. Додатковою перевагою є інтерактивність, оскільки можлива реалізація розширених інструментів взаємодії за потреби підвищує зручність роботи користувача. Можливість розподіленої реалізації підвищує відмовостійкість: окремі частини програмного забезпечення можуть продовжувати роботу навіть у разі відмови інших компонентів.

Водночас архітектура має і певні обмеження. Її ефективність значною мірою залежить від якості мережевої інфраструктури: для забезпечення низької затримки та безперервної роботи потрібні стабільні канали зв’язку з достатньою пропускною здатністю, особливо при децентралізованому розгортанні. На етапі впровадження можуть виникати суттєві початкові витрати, оскільки повнофункціональне розгортання вимагатиме інвестицій у серверні ресурси, налаштування інфраструктури та забезпечення безпеки передавання даних. Використання багаторівневої модульної структури, різних брокерів повідомлень та механізмів потокового оброблення зумовлює підвищену складність початкового налаштування й може вимагати значних часових та ресурсних витрат. Оптимальна робота ресемплінгу потребує ретельного налаштування параметрів, що ускладнює інтеграцію або адаптацію системи у разі змін типу навантаження чи структури відображуваних даних. Крім того, при некоректно обраних параметрах ресемплінгу можлива часткова втрата дрібних деталей у даних, що може виявитися критичним для систем, де важливі точні та високодеталізовані показники у режимі реального часу.

У подальшому розвитку програмного забезпечення доцільно розглядати інтеграцію алгоритмів штучного інтелекту та машинного навчання, що дасть змогу автоматизувати виявлення аномалій у потоках, здійснювати прогнозування майбутніх станів об’єктів та адаптивно оптимізувати параметри відображення залежно від поточного навантаження та аналітичних цілей. Окремого опрацювання потребує покращення мобільного доступу: оптимізація інтерфейсу для смартфонів і планшетів із підтримкою жестів, спрощеною навігацією та push-сповіщеннями про критичні події підвищить швидкість реагування в умовах, коли у користувача немає фіксованого робочого місця. Перспективним напрямом є також поглиблення хмарно-периферійної інтеграції: активніше використання edge computing для попереднього оброблення та фільтрування даних безпосередньо на периферії мережі дасть змогу знизити

навантаження на центральні вузли й скоротити затримки. Можливе розширення механізмів персоналізації інтерфейсу: профілі користувачів можуть автоматично визначати вигляд панелей, типи графіків і способи взаємодії, спираючись на історію роботи та роль у процесах аналізу, що підвищить зручність і аналітичну цінність програмного забезпечення.

Узагальнюючи наведене, можна зазначити, що розроблена архітектура має потенціал як для впровадження у поточних IoT-системах, так і для подальшого вдосконалення з урахуванням розвитку технологій потокового оброблення та візуалізації даних. Її модульність і масштабованість забезпечують можливість адаптації до нових реалізацій технологій та розширення програмного забезпечення без істотних змін у структурі. Враховуючи переваги й недоліки, доцільно окреслити особливості практичного застосування запропонованої архітектури для різних класів IoT-рішень. Для великих систем із високим навантаженням, таких як Smart City чи IIoT, доцільним є використання гібридної моделі розгортання, яка поєднує переваги часткового оброблення ближче до пристроїв (низька затримка, менший обсяг трафіку) з централізованим аналізом (зручність управління, консолідація даних). Для систем, чутливих до затримок, зокрема в eHealth чи критичних промислових комплексах, пріоритет може надаватися централізованому розгортанню з локалізованою агрегацією та ресемплінгом даних. При впровадженні програмного забезпечення важливо приділити увагу налаштуванню параметрів ресемплінгу та агрегації й провести експериментальне тестування для визначення оптимального балансу між точністю, продуктивністю та ресурсними витратами.

Запропонована узагальнена архітектура, завдяки своїй гнучкості, стабільності та високій продуктивності, має потенціал для застосування у різних прикладних галузях IoT, однак для її ефективного впровадження необхідно враховувати окреслені обмеження й особливості конфігурації. Оскільки розроблена архітектура передбачає варіативність реалізації

компонентів і різні сценарії розгортання, для її застосування доцільно розробити типові повторно використовувані рішення, що можуть бути застосовані при реалізації архітектури.

4.5. Впровадження програмного забезпечення візуалізації даних пристроїв IoT

Варіанти розгортання програмного забезпечення можна описати як гнучкий набір підходів, що дають змогу адаптувати узагальнену архітектуру відображення потокових темпоральних мультимодальних даних IoT до різних умов експлуатації, масштабів проєкту та вимог до продуктивності (НФВ2). Така гнучкість є однією з ключових переваг архітектури, оскільки забезпечує можливість балансувати між простотою впровадження, вартістю інфраструктури, швидкодією та відмовостійкістю залежно від потреб конкретного сценарію використання (НФВ3).

Централізоване розгортання. У межах нього всі основні блоки архітектури, тобто блок роботи з пристроями, блок групування даних та блок відображення, розміщуються на одному сервері або в межах одного обчислювального вузла, тобто ПК безпосереднього користувача. Цей варіант є найпростішим, передбачає мінімальні витрати на налаштування та підтримку, забезпечує простоту адміністрування та мінімальні внутрішньосистемні затримки, а також знижує вимоги до мережевої інфраструктури. Водночас він має обмежену масштабованість, є вразливим до відмов одного вузла і менш придатний для географічно розподілених IoT пристроїв. Такий підхід можливо використовувати для пілотних проєктів, тестових проєктів та невеликих локальних наборів пристроїв IoT. Діаграма такого розгортання зображена на рис. 4.3.

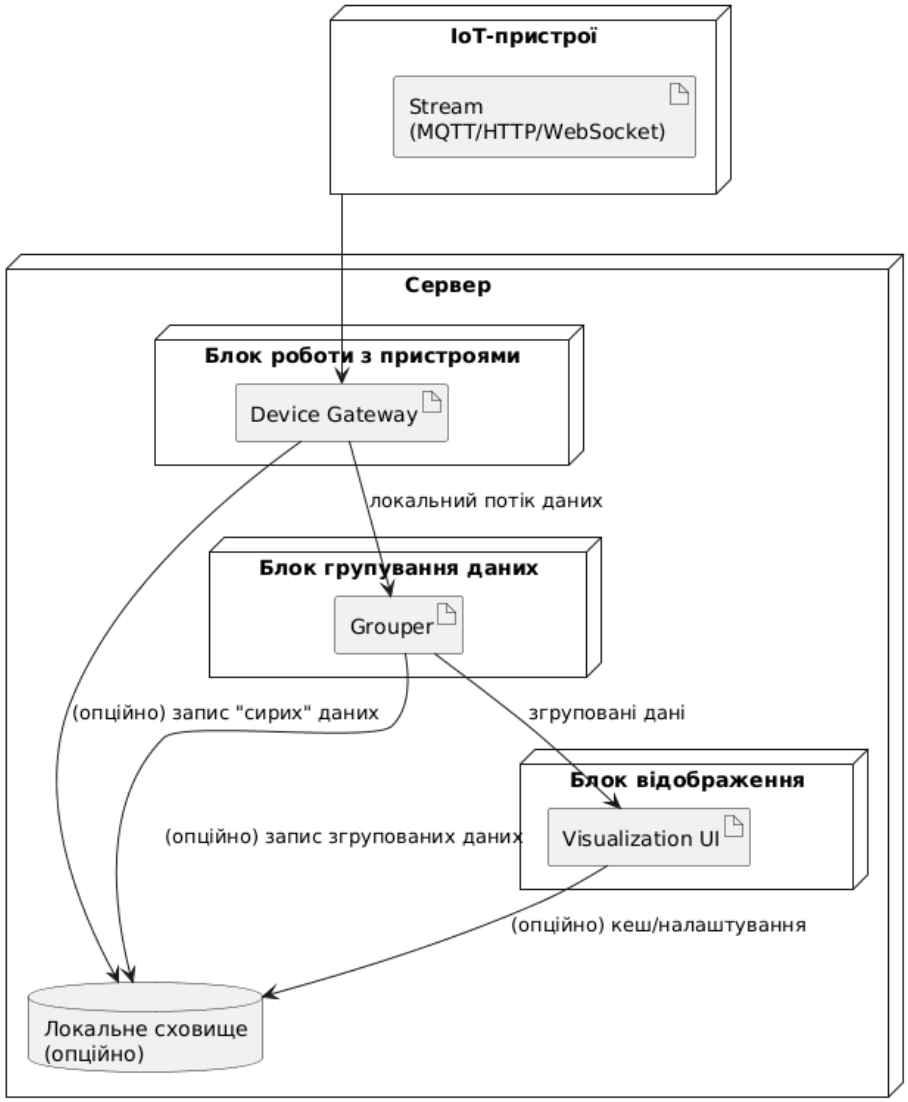


Рис. 4.3. Діаграма централізованого розгортання (один обчислювальний вузол)

Децентралізоване розгортання. У межах нього основні блоки архітектури розподіляються між кількома пристроями, потенційно розміщеними на значному віддаленні одне від одного (НФВ5). Наприклад, блок роботи з пристроями може бути розташований ближче до джерел даних, реалізовуючи технологію Edge Computing, що мінімізує затримки, тоді як відображення може виконуватись безпосередньо на робочих місцях користувачів (рис. 4.4). Такий підхід зменшує затримки, підвищує відмовостійкість і дає змогу масштабувати систему шляхом додавання нових пристроїв. Водночас він складніший в адмініструванні та вимагає

додаткових механізмів доставки даних. Подібний варіант доречний для розподілених IoT пристроїв.

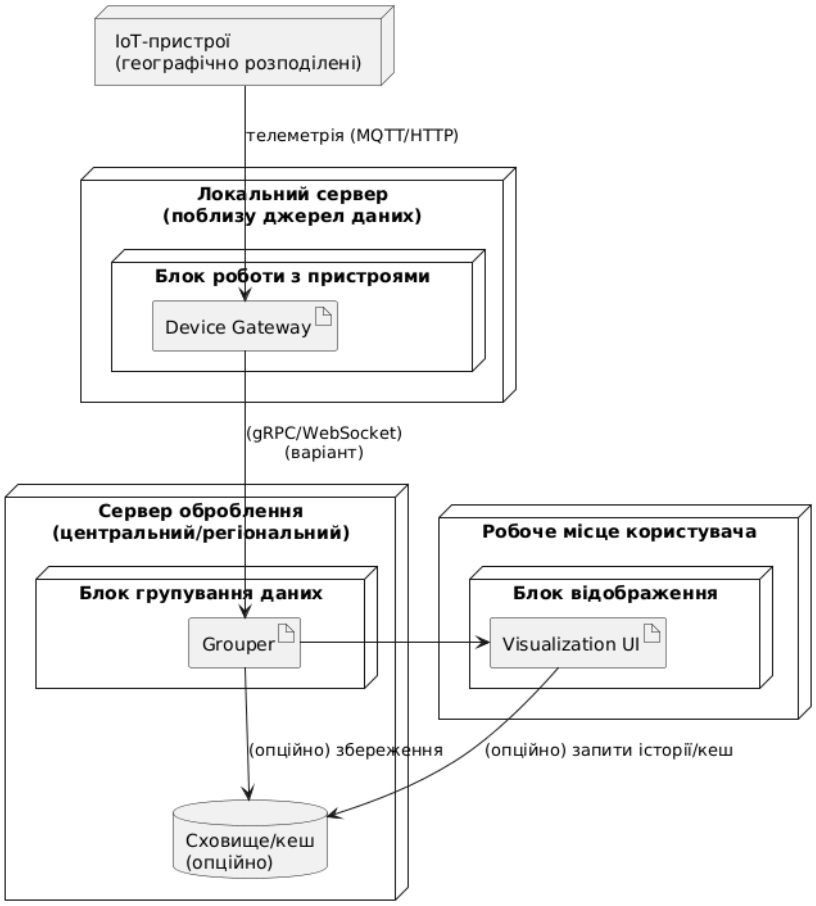


Рис. 4.4. Діаграма децентралізованого розгортання

Мікросервісне розгортання. Кожен функціональний блок архітектури реалізується як окремий мікросервіс з власним API та інфраструктурою (рис. 4.5).

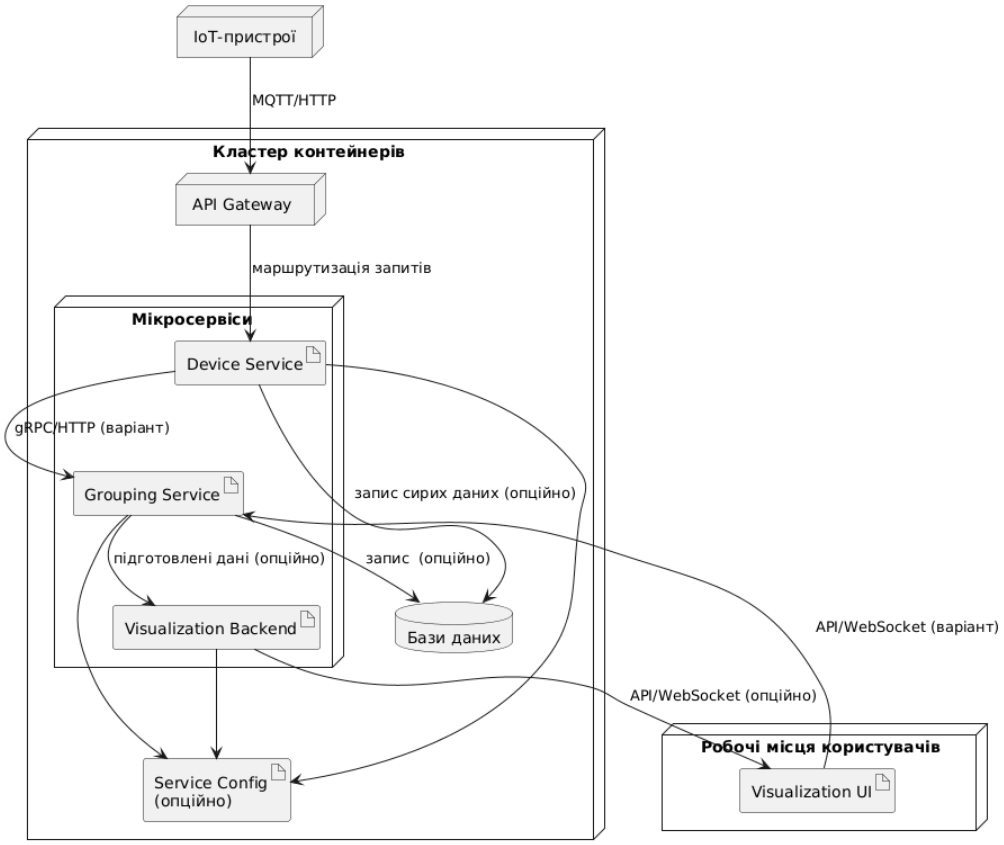


Рис. 4.5. Діаграма мікросервісного розгортання

Мікросервіси можуть розгортатися у контейнерах або хмарних середовищах, таких, що забезпечує гнучкість і незалежність компонентів. Такий підхід дає змогу масштабувати окремі блоки незалежно, виконувати оновлення та заміни модулів без зупинки програмного забезпечення. Також можливо й інтегруватися із зовнішніми сервісами. Однак він потребує розвинених DevOps-процесів і складнішого моніторингу взаємодій між сервісами. Цей варіант найбільше підходить для великих IoT-проєктів та хмарних сервісів із високою динамікою змін. Також варто зауважити, що існують такі технології як AWS Greengrass [102] та Azure IoT Edge [103], що дозволяють автоматизовано розгортати контейнери з блоками роботи з пристроями для отримання їх даних і подальшого відправлення до відповідні хаби AWS IoT Core та Azure IoT Hub. Автоматичне масштабування потужностей оброблення даних пристроїв IoT існує, наприклад, в Azure Event Hubs та AWS Kinesis. До того ж, з використанням

хмарних технологій, можливо збагачувати отримані від прострою дані також інформаціюєю про тип пристрою та іншою метаінформацією, що може бути використана при подальшому аналізі для врахування критичності пристрою або ж, наприклад, інформацією про його місцезнаходження.

Вибір конкретної моделі розгортання залежить від кількості IoT пристроїв, географічного розташування пристроїв, вимог до затримок і продуктивності, а також бюджету на інфраструктуру. Також можливий гібридний підхід, що поєднує централізовані й децентралізовані елементи, коли первинне оброблення даних виконується у віддалених вузлах, а групування, аналітика та візуалізація зосереджені у централізованому середовищі.

Від способу розгортання також залежить механізм масштабування архітектури. У разі повністю централізованого розгортання масштабування дасть змогу лише збільшення ресурсів конкретного сервера, однак у разі децентралізованого розгортання програмне забезпечення може масштабуватися як горизонтально, так і вертикально. Горизонтальний підхід передбачає додавання нових вузлів чи екземплярів компонентів та розподіл потоків даних між ними, що дає змогу обслуговувати велику кількість пристроїв. Вертикальний підхід ґрунтується на збільшенні обчислювальних ресурсів наявних серверів і є доцільним при помірному зростанні навантаження, хоча й обмежується фізичними можливостями обладнання та економічною доцільністю. Оптимальним рішенням може бути комбінування обох видів масштабування.

У процесі масштабування значну роль відіграють сучасні інструменти. Контейнеризація (Docker, Kubernetes) забезпечує горизонтальне масштабування компонентів завдяки швидкому запуску додаткових контейнерів, а також автоматизує управління ресурсами і балансування навантаження. Оркестрація інфраструктури дає змогу автоматизувати розгортання та налаштування у хмарних і гібридних середовищах, забезпечуючи стандартизацію конфігурацій. Системи

моніторингу та балансування навантаження забезпечують контроль ключових показників продуктивності, включно із затримками та навантаженням на CPU і GPU, що дає змогу виявляти необхідність у додаткових ресурсах у реальному часі.

4.6. Шаблон проєктування програмного забезпечення для відображення потокових даних Multimodal Stream Stable Visualization (MSSV)

Шаблон проєктування для відображення мультимодальних потокових даних Multimodal Stream Stable Visualization (MSSV) виконує роль стандартизованого архітектурного рішення для програмного забезпечення. Його задачею є взаємодія користувача з даними, що надходять у режимі реального часу від пристроїв IoT. Його мета полягає у спрощенні розроблення програмного забезпечення, що призначене для відображення потокових даних пристроїв IoT завдяки можливості використання вже розроблених структурних елементів для певної типової задачі, що також допомагає з майбутнім розширенням програмного забезпечення відображення таких даних завдяки модульності та гнучкості розробленого шаблону. Відповідно, також використання шаблону сприятиме підвищенні якості коду, зменшенню часу розроблення та підтримки цього коду у майбутньому.

Сформулюємо опис розробленого шаблону проєктування згідно зі загальноприйнятою схемою опису шаблонів [104].

Intent (Намір). Шаблон описує, як організувати презентаційний шар для відображення темпоральних мультимодальних даних пристроїв так, щоб значення могли бути переглянуті як окремо по кожному параметру, так і по поєднанню параметрів в реальному часі. Рішення розділяє надходження подій, видобування параметрів і побудову відображуваних рядів на чіткі ролі та контракти.

Motivation (Мотивація). У потокових IoT-сценаріях дані з різних джерел приходять із різною частотою і з можливими запізненнями. Якщо відображати дані безпосередньо з потоку, інтерфейс отримує нестабільні графіки та нерівномірну латентність. До того ж, у такому випадку дані будуть відображені окремо. У запропонованій побудові новий пакет подій надходить через сервіс постачання, фіксується моделлю, потрібне числове значення отримується, далі опрацьовується в параметр-моделі та лише потім потрапляє до кореневої моделі, яка керує часовими вікнами і оновлює видимі ряди частково.

Завдяки цьому пізні події не руйнують попередньо відображену картину, а користувацький вибір параметрів впливає тільки на відповідні проєкції. А можливість обрати параметри для узгодженого відображення надає змогу працювати з відображеними поєднаними за темпоральною ознакою даними, що надає нові можливості для аналізу відповідних даних.

Applicability (Застосовність). Шаблон доцільно застосовувати, коли інтерфейс у реальному часі має спільно відображати кілька параметрів одного чи багатьох пристроїв, коли можливі пізні або переставлені події та різні частоти надходження.

Structure (Структура). Структура розробленого шаблону представлена на рис. 4.6.

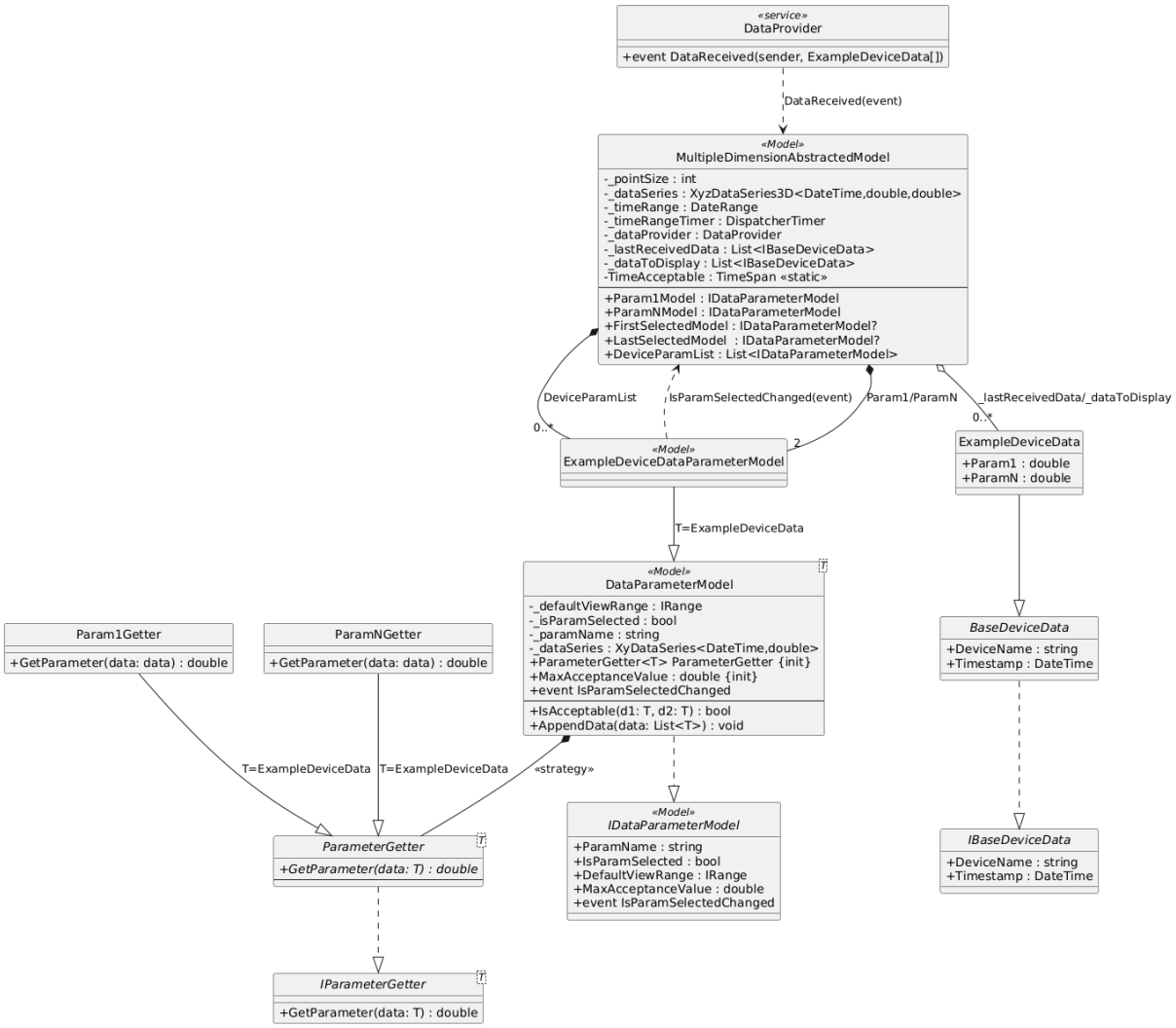


Рис. 4.6. Структурна діаграма шаблона проєктування для відображення мультимодальних потокових даних Multimodal Stream Stable Visualization

Опис окремих компонентів:

- IBaseDeviceData та BaseDeviceData. Інтерфейс і базова (абстрактна) реалізація для даних, що надходять: уніфіковані поля назви пристрою та часової мітки;
- ExampleDeviceData. Конкретна доменна сутність з потрібними полями даних (наприклад, Param1 … ParamN), яка наслідує базову модель і зберігає вимір із прив’язкою до пристрою та моменту часу, необхідна окрема реалізація для кожного конкретного випадку;
- IParameterGetter<T> та ParameterGetter<T>. Узагальнений інтерфейс і абстрактна основа для стратегій, що дістають значення з об’єкта типу

T (має наслідуватися від BaseDeviceData). У цій частині за необхідності реалізуються методи отримання дискретних подій з мультимодальних даних. Метод GetParameter є чистим: не змінює вхідних даних і не має побічних ефектів;

- Param1Getter та ParamNGetter. Конкретні реалізації стратегії для ExampleDeviceData, які інкапсулюють правило отримання потрібного параметра. Необхідно створення конкретної стратегії для кожного окремого параметра;
- IDataParameterModel. Інтерфейс моделі параметру з назвою параметра, ознакою вибраності, типовим діапазоном огляду, порогом прийнятності та подією зміни вибору;
- DataParameterModel<T>. Узагальнена реалізація моделі параметру, що тримає власну серію значень для відображення, приймає ін’єкцію стратегії ParameterGetter<T>, додає дані пакетами (AppendData) і перевіряє їх на прийнятність у часі та за значенням (IsAcceptable);
- ExampleDeviceDataParameterModel. Реалізація для ExampleDeviceData;
- DataProvider. Сервіс постачання даних, що маж подію DataReceived із списком нових ExampleDeviceData. Єдине джерело надходження даних у презентаційний шар;
- MultipleDimensionAbstractedModel. Коренева модель композиції: тримає часовий діапазон і таймер кадру, буфери отриманих даних та даних до відображення, набір моделей параметрів та посилання на перший та останній обрані.

Collaborations (Співпраці). Коренева модель володіє параметр-моделями та їхнім списком, приймає події DataReceived від постачальника, а також реагує на IsParamSelectedChanged від конкретних моделей параметру.

Consequences (Наслідки). Така декомпозиція спрощує розширення: щоб додати новий параметр, достатньо реалізувати стратегію отримання параметру та модель параметру і додати їх в базову модель відображення.

Implementation (Реалізація). На рівні контрактів варто зафіксувати, що Timestamp визначає порядок опрацювання та входження у поточне вікно. Події вибору параметра піднімаються з моделей параметрів угору, і коренева модель оновлює лише відповідні часові ряди. Оновлення бажано має керуватися окремим таймером кадру, а не кожною окремою подією. Для зменшення навантаження корисні ресемплінг і повторне використання буферів. Додавання нового параметра не має змінювати наявні частини: створюється чергова реалізація ParameterGetter<NDeviceData> і відповідна DataParameterModel<NDeviceData>, які включаються до DeviceParamList.

Related Patterns (Пов'язані шаблони). Шаблон поєднується з MVVM (логіка в моделях, відображення – окремо), використовує стратегію для видобування значень із об'єкта даних, та використовує спостерігача для подій. На відміну від локальних UI-патернів окремих віджетів, це рішення описує структуру та поведінку всієї презентаційної частини загального програмного забезпечення оброблення та візуалізації даних.

4.7. Узагальнена схема та принципи побудови користувацького інтерфейсу для розробленого шаблону проєктування

Для забезпечення програмної реалізації розробленого шаблону проєктування також було спроєктовано схему користувацького інтерфейсу, яка безпосередньо реалізує логіку відображення мультимодальних даних. Розроблений користувацький інтерфейс базується на модульному підході, що передбачає поділ робочої області на логічно ізольовані зони:

- панель вибору параметрів – призначена для вибору джерел даних, параметрів відображення та налаштування фільтрів. Забезпечує швидкий доступ до основних функцій без необхідності перемикання між вкладками;
- зона 2D-візуалізації – містить набір окремих графіків для відображення параметрів у двовимірному просторі. Кожен графік відповідає за

окремий вимір або показник, що дає змогу аналізувати деталі роботи окремих пристроїв;

– зона 3D-візуалізації – призначена для відображення взаємозв’язків між кількома параметрами одночасно. Вона дає змогу одночасно оцінювати як стан системи у певний момент часу, так і динаміку змін за поєднанням двох параметрів.

Інтерфейс, схема якого представлена на рис 4.7, побудований із дотриманням принципів узгодженості елементів, мінімізації кількості необхідних дій для досягнення результату та адаптивності до різних розмірів екранів і типів пристроїв.

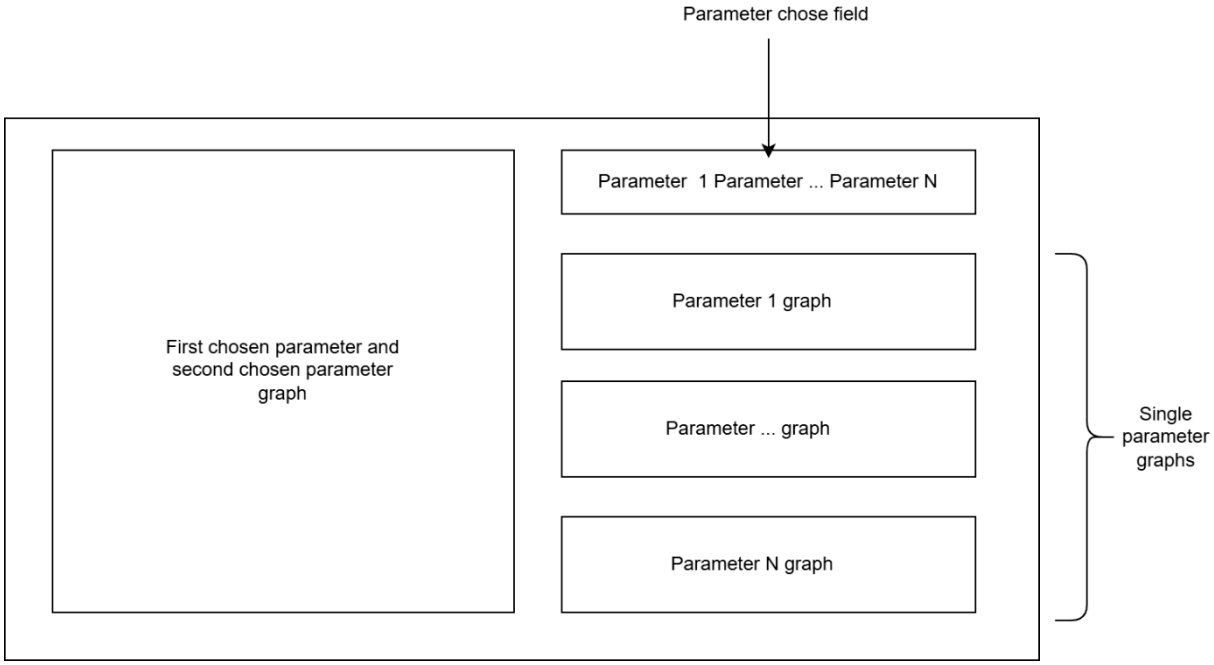


Рис. 4.7. Користувацький інтерфейс для відображення мультимодальних темпоральних потокових точкових даних пристроїв IoT

Інтерфейс, приклад реалізації якого представлений на рис. 4.8, підтримує набір інструментів для інтерактивної роботи з даними:

– зміна параметрів відображення без перезавантаження всієї візуалізації. Користувач може додавати або вилучати параметри з графіку у реальному часі;

– масштабування та панорамування для детального аналізу окремих ділянок даних або огляду загальної картини;
– часова фільтрація з можливістю швидкого вибору інтервалів (наприклад, останні 5 хвилин, весь доступний час відображення історичних даних тощо);
– перемикання режимів – відображення даних у режимі реального часу або робота з історичними даними.

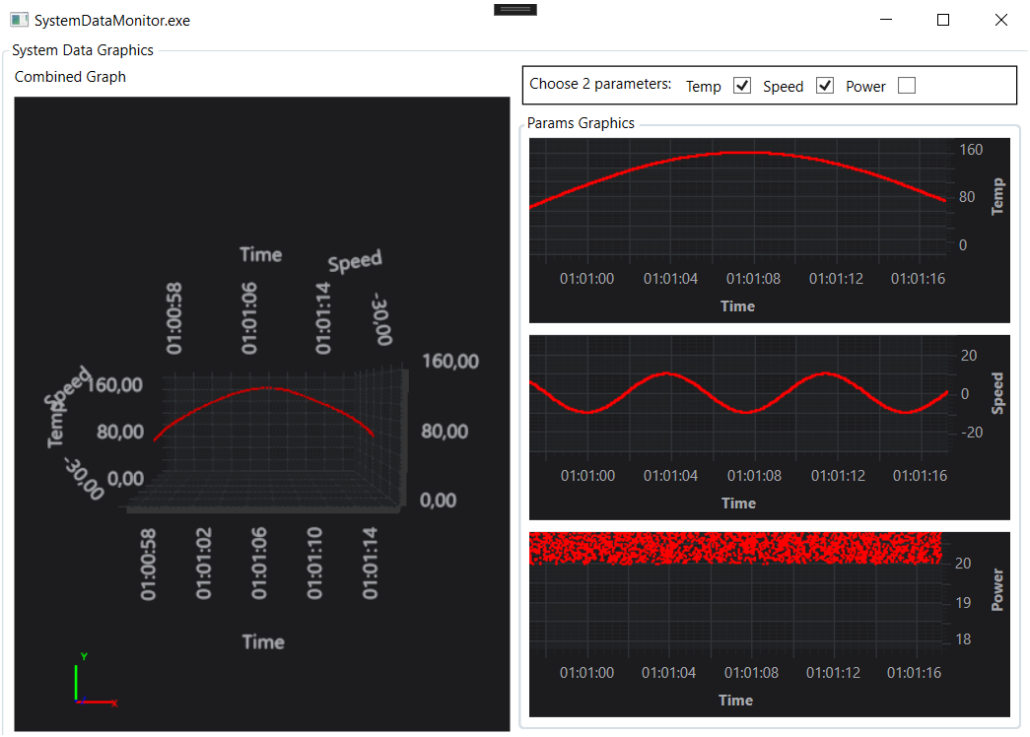


Рис. 4.8. Приклад реалізації користувацького інтерфейсу відображення мультимодальних потокових даних

Завдяки цим механізмам користувач може не лише спостерігати за поточними графіками, а й знаходити та аналізувати аномалії чи тренди використовуючи механізми керування відображенням та переглядати стан подій також й на окремому *подієво-орієнтованому* графіку. Приклад такого відображення наведений на рис. 4.9. Воно поєднує базові компоненти раніше описаної схеми для відображення мультимодальних даних про стан

виробництва з додатковими елементами візуалізації супутніх показників та згрупованих подієвих даних.

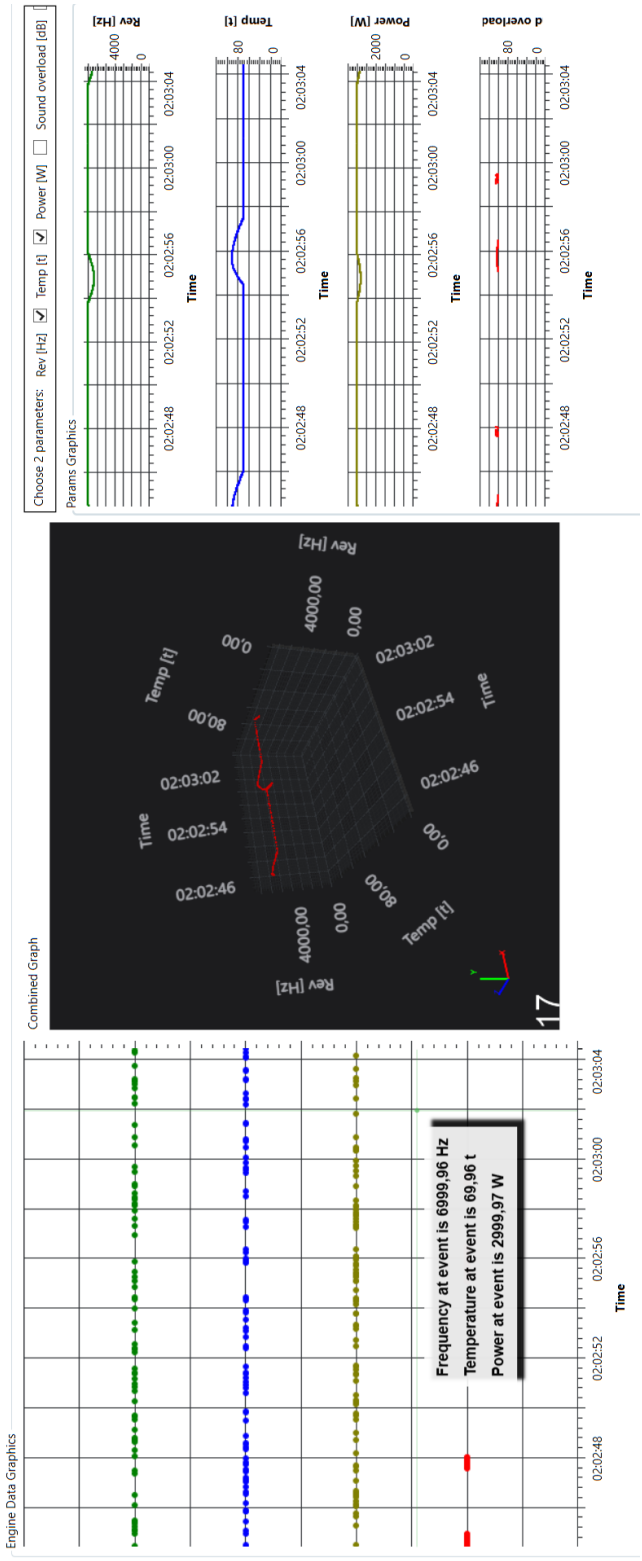


Рис. 4.9. Приклад реалізації користувацького інтерфейсу програмного забезпечення моніторингу стану виробництва

Можливість динамічного комбінування обраних даних надає аналітику розширені можливості для встановлення або спростування зв’язків між відповідними даними, що створює необхідні умови для оперативного прийняття обґрунтованих управлінських рішень відповідно до поточної ситуації в режимі реального часу. А завдяки можливості розроблення нового типу користувацьких інтерфейсів – *подієво-орієнтованих інтерфейсів*, користувач отримує можливість працювати з подіями як з новим типом даних на відображенні.

На відміну від класичних інтерфейсів, заснованих на Scatter Plot Matrix (SPLOM) [70] або окремих статичних панелях моніторингу, запропонована схема інтерфейсу має кілька ключових переваг:

- інтеграція 3D-відображення з часовою віссю дає змогу в одному візуальному просторі відслідковувати як поточні значення, так і їхню зміну в часі;
- динамічна адаптація вмісту – користувач сам формує вигляд інтерфейсу, виходячи з поточних завдань, що надає можливість користувачу обрати саме ті дані, які необхідно перевірити в конкретний момент часу;
- поєднання відображення всіх окремих параметрів на об’єднаного відображення по обраним параметрам в рамках одного вікна, або, за необхідності, розділення на декілька окремих вікон у разі реалізації з наданням можливості використовувати розділення на окремі вікна (докінг).

Крім того, реалізація подієво-орієнтованого підходу дає змогу перейти від спостереження за безперервними потоками значень до аналізу дискретних подій на користувацьких інтерфейсах нового типу – *подієво-орієнтованих інтерфейсах*. Використання інтерактивних маркерів подій дозволяє значно підвищити щільність інформації в інтерфейсі. Кожен такий маркер забезпечує миттєву навігацію та доступ до синхронізованих мультимодальних даних, що безпосередньо стосуються зафіксованої події.

Це створює умови для швидкого отримання контексту події та підвищує ефективність прийняття рішень у реальному часі.

Для підвищення ефективності роботи користувача з інтерфейсом у системі можливо реалізувати набір розширених інструментів взаємодії, що дають змогу керувати відображенням даних, виконувати навігацію у складних візуалізаціях та адаптувати робочий простір під конкретні завдання. До таких інструментів належать:

- гарячі клавіші та комбінації клавіш, що дають змогу перемикатися між наборами параметрів, змінювати масштаб, активувати або вимикати окремі графіки, а також викликати інструменти фільтрації чи експорту;
- підтримка жестів (для сенсорних екранів і мобільних пристроїв) з можливістю масштабування, обертання 3D-сцени, панорамування та переходу в часовому діапазоні;
- контекстні меню для швидкого доступу до налаштувань візуалізації без необхідності переходу до окремих панелей конфігурації за їх наявності;
- інтерактивний вибір елементів візуалізації – користувач може виділити конкретну точку або групу точок на графіку, щоб переглянути наявні атрибути, пов’язані події або історичні дані для цього елемента, у разі наявності такої інформації;
- експорт даних та візуалізацій у поширені формати (CSV, JSON, PNG, PDF) безпосередньо з інтерфейсу, що спрощує формування звітів і подальший аналіз у сторонніх інструментах або збереження певного стану відображення для його використання у майбутньому;
- можливість створення користувацьких шаблонів відображення, що зберігають налаштування розташування графіків, фільтрів та параметрів, аби їх можна було застосувати для повторних сесій роботи у разі перезавантаження системи відображення.

Завдяки цим інструментам користувач може отримати набір можливостей для інтерактивної роботи з даними, що може скоротити час на аналітичні операції та підвищити зручність користування програмним забезпеченням.

4.8. Висновки до четвертого розділу

У цьому розділі розроблено та обґрунтовано узагальнену архітектуру програмного забезпечення відображення потокових темпоральних мультимодальних даних пристроїв IoT, орієнтовану на роботу в режимі реального часу та підтримку інтерактивної взаємодії з користувачем. Проведений аналіз уможливив формування переліку функціональних та нефункціональних вимог.

Архітектура побудована за модульним принципом і включає три ключові блоки: блок роботи з пристроями IoT, що забезпечує отримання, стандартизацію та первинне оброблення даних; блок групування даних, який відповідає за часову та змістову агрегацію, створення потоків і підготовку їх до передачі на блок візуалізації; блок відображення даних, що реалізує побудову 2D та 3D візуалізацій, інтерактивну фільтрацію та керування параметрами відображення у режимі реального часу.

Детально описано логіку оброблення даних, яка охоплює етапи від отримання інформації з пристроїв до її інтерактивного представлення кінцевому користувачу. Запропоновано та проаналізовано різні варіанти розгортання програмного забезпечення (централізований, децентралізований та мікросервісний) з визначенням їхніх переваг, обмежень та сфер застосування.

Обґрунтовано вибір можливих технологій для реалізації архітектури, зокрема використання протоколів MQTT та Apache Kafka для передавання повідомлень, а також сучасних інструментів оброблення потоків і візуалізації.

Розроблено шаблон проєктування програмного забезпечення для відображення мультимодальних потокових даних, який стандартизує організацію презентаційного шару для програмного забезпечення відображення мультимодальних потокових даних та надає можливості відображення подієво-орієнтованого інтерфейсу. Застосування цього шаблону дає змогу розділити процеси надходження подій, видобування конкретних параметрів даних та побудови візуалізацій та уможливлює підтримку нових типів вхідних даних без порушення цілісності програмного забезпечення, що спрощує розроблення програмного забезпечення.

Запропоновано схему користувацького інтерфейсу, що поєднує модульну структуру, інтегровані 2D- та 3D-відображення, подієво-орієнтовані графіки, механізми інтерактивності, персоналізації та розширені інструменти взаємодії.

Проведений аналіз переваг та обмежень розробленої архітектури показав її гнучкість, здатність до масштабування, та придатність для використання у різних прикладних галузях. Визначено перспективи подальшого розвитку, що включають інтеграцію алгоритмів штучного інтелекту, розширення підтримки мультимодальних даних та активніше використання хмарно-периферійних обчислень.

Таким чином, розроблена узагальнена архітектура є можливою основою для створення вдосконаленого програмного забезпечення візуалізації потокових даних IoT, здатною забезпечити оброблення потоків даних від пристроїв IoT та представлення результатів у режимі реального часу.

ВИСНОВКИ

У дисертаційній роботі розв’язано актуальну науково-технічну задачу удосконалення процесів створення програмних засобів для подіє-орієнтованої візуалізації даних великих обсягів у програмних системах оброблення мультимодальних темпоральних даних пристроїв IoT.

Основні наукові та практичні результати полягають у наступному.

1. Проведено ґрунтовний аналіз сучасних підходів до роботи з потоковими мультимодальними даними та їх відображення, який показав, що наявні рішення здебільшого не забезпечують одночасно стабільності візуалізації та масштабованого оброблення гетерогенних потоків даних.
2. Проаналізовано та формалізовано вимоги до програмного забезпечення процесів аналізу та відображення мультимодальних потокових даних. Зокрема, обґрунтовано наступні основні вимоги до систем візуалізації потокових IoT-даних: стабільність раніше відображених сегментів, робота в реальному часі, інтерактивність, підтримка мультимодальності, масштабованість.
3. Розроблено метод подіє-орієнтованого групування темпоральних мультимодальних даних зі встановленням часових допусків. Метод забезпечує багаторівневу агрегацію, подіє-орієнтоване групування даних великих обсягів і толерантність до мережевих затримок. Виконано реалізацію розробленого методу з використанням Rx.NET. Експериментально підтверджено можливість оброблення потоків з великою інтенсивністю подій, а саме, до 40 000 подій за секунду від 128 віртуальних пристроїв на тестовому стенді.
4. Розроблено метод візуалізації темпоральних даних зі збереженням стабільності вже відображених даних шляхом введення обмеженого у часі інтервалу даунсемплінгу та трирівневої моделі даних (шари змінних, стабільних даних та даних, що підлягають видаленню). Це

усуває нестабільні зміни, зменшує кількість перемальовувань, зберігає стабільність зображення відносно попередньо відображених значень. Виконано програмну реалізацію розробленого методу візуалізації. Показано, що за рахунок фіксації відображених сегментів і контрольованого даунсемплінгу навантаження на CPU та GPU при відображенні великих потоків подій у реальному часі зменшується більш ніж у 2 рази залежно від типу відображення. При цьому забезпечується візуальна стабільність відображення – схожість з відображенням без застосування даунсемплінгу складає не менше 90%.

5. Розроблено узагальнену архітектуру програмного забезпечення відображення потокових даних пристроїв IoT, яка забезпечує подієво-орієнтоване групування темпоральних мультимодальних даних та стабільне відображення даних великих обсягів у реальному часі. Розроблена архітектура передбачає можливість створення програмного забезпечення візуалізації даних, яке уможливлюватиме: розширення функціональності з відображення даних у наявних програмних системах оброблення даних пристроїв IoT; створення нових програмних засобів з подіє-орієнтованою візуалізацією мультимодальних темпоральних даних, зокрема, для систем підтримки прийняття рішень; масштабування та розподілене розгортання програмних компонентів.
6. Розроблено архітектурний шаблон проєктування програмного забезпечення для візуалізації потокових даних Multimodal Stream Stable Visualization (MSSV), який декомпозує процеси надходження подій, видобування даних та побудови відповідних візуалізацій. Шаблон базується на розділенні відповідальності між блоками постачання даних, стратегіями видобування параметрів та моделями відображення, що дає змогу додавати нові типи даних без модифікації наявного коду.

7. Розроблено узагальнену схему користувацького інтерфейсу для інтерактивного аналізу даних великих обсягів, що складається з трьох взаємопов’язаних зон: панелі конфігурації відображення для вибору параметрів, вікон візуалізації окремих параметрів та зв’язаного простору для комбінованого відображення обраних параметрів у тривимірному просторі.
8. Обґрунтовано створення подієво-орієнтованих користувацьких інтерфейсів, що базуються на переході від візуалізації безперервних потоків значень до представлення дискретних подій. Це уможливлює розроблення користувацьких інтерфейсів нового типу – подієво-орієнтованих інтерфейсів, які дають змогу оперувати подіями як самостійними одиницями даних та надають можливість швидкого доступу до пов’язаних з подією мультимодальних темпоральних даних, що сприятиме розширенню функціональності програмних систем оброблення даних IoT та систем підтримки прийняття рішень.

Таким чином, сформовано комплексне теоретичне й прикладне підґрунтя для створення програмних засобів візуалізації мультимодальних темпоральних потокових даних пристроїв IoT із стабільністю відображення та масштабованістю. Запропоновані методи й архітектурні рішення забезпечують можливість реалізації інструментів для візуального аналізу в реальному часі та можуть бути безпосередньо інтегровані як в наявні, так і в майбутні промислові, міські та медичні платформи моніторингу.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Keim D. Solving problems with visual analytics: Challenges and applications. / D. Keim, Z. Leishi. // Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies. – 2011. – C. 1–4.
2. Islam S. M. S. Quality-Focused Internet of Things Data Management: A Survey. / S. M. S. Islam, S. Nooruddin, F. Karray. // IEEE Internet of Things Journal. – 2024. – Vol. 11, № 2. – P. 1-18.
3. Ahmed S. F. et al. Toward a Secure 5G-Enabled Internet of Things: A Survey on Requirements, Privacy, and Challenges. / S. F. Ahmed et al. // IEEE Access. – 2024. – Vol. 12. – P. 13125-13145.
4. Muhammad Ali N., Novikov B. A. Big Data: Analytical Solutions, Research Challenges and Trends // Proceedings of ISP RAS. 2020. T. 32. № 1. C. 181–204.
5. McCulloch E. S. Harnessing the Power of Big Data in Biological Research // BioScience. 2013. T. 63. № 9. C. 715–716.
6. Cui W. Visual Analytics: A Comprehensive Overview // IEEE Access. 2019. T. 7. C. 81555–81573.
7. Scholtz J. User-Centered Evaluation of Visual Analytics. // Cham: Springer, 2018. – XI, 71 p.
8. Wong P. C. та ін. The Top 10 Challenges in Extreme-Scale Visual Analytics // IEEE Comput. Grap. Appl. 2012. T. 32. № 4. C. 63–67.
9. Sasaki Y. A Survey on IoT Big Data Analytic Systems: Current and Future. / Yuya Sasaki. // IEEE Internet of Things Journal. – 2022. – Vol. 9, № 2. – P. 1024-1036.
10. Ulmer A., Angelini M., Fekete J.-D. та ін. A Survey on Progressive Visualization // IEEE Trans. Vis. Comput. Graph. 2024. T. 30. № 9. C. 6447–6467.

11. Cohen L. Time-frequency distributions-a review / Leon Cohen. // Proceedings of the IEEE 77. – 1989. – №7. – C. 941–981.
12. Mykhailo L., Yevgeniya S. Real-time data visualization for IoT network systems: challenges and strategies for performance optimization // System technologies. – 2023. – Vol. 5, № 148. – C. 52–61.
13. Miller H. J. A Measurement Theory for Time Geography // Geographical Analysis. 2004. T. 37. № 1. C. 17–45.
14. Y. Zhao, X. Luo, X. Lin et al. Visual analytics for electromagnetic situation awareness in radio monitoring and management. // IEEE transactions on visualization and computer graphics. – 2019. – №26. – C. 590–600.
15. M. Cotton, J. Wepman, J. Kub et al. An overview of the NTIA/NIST spectrum monitoring pilot program // 2015 IEEE Wireless Communications and Networking Conference Workshops (WCNCW). – 2015. – C. 217 - 222.
16. M. Brehmer, B. Lee, B. Bach et al. Timelines revisited: A design space and considerations for expressive storytelling. // IEEE transactions on visualization and computer graphics. – 2016. – №24. – C. 2151–2164.
17. Wang C. et al. Apache IoTDB: a Time series database for IoT applications. / C. Wang et al. // Proceedings of the ACM on Management of Data. – 2023. – Vol. 1, № 2. – P. 1–27.
18. Adnan M., Just M., Baillie L. Investigating Time Series Visualisations to Improve the User Experience // Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI ’16). – New York: ACM, 2016. – P. 5444–5455.
19. Aigner W. Visualization of Time-Oriented Data. 2nd ed. / Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, Christian Tominski. – London: Springer, 2023. – 441 p.

20. Andrienko N., Andrienko G. Visual analytics of movement: An overview of methods, tools and procedures // Information Visualization. 2012. T. 12. № 1. C. 3–24.
21. Saleem A. et al. A Comprehensive Systematic Survey of IoT Protocols: Implications for Data Quality and Performance. / A. Saleem et al. // IEEE Access. – 2024. – Vol. 12. – P. 196206-196225.
22. EmotionTFN: Hierarchical temporal fusion architecture for emotion recognition in IoT. // Applied Sciences (MDPI). – 2024. – Vol. 15, № 14. – 8089.
23. Heldner M. Detection thresholds for gaps, overlaps, and no-gap-no-overlaps // The Journal of the Acoustical Society of America. 2011. T. 130. № 1. C. 508–513.
24. Maceachren A. M., Kraak M.-J. Exploratory cartographic visualization: Advancing the agenda // Computers & Geosciences. 1997. T. 23. № 4. C. 335–343.
25. Damaševičius R. et al. Visual analytics for cyber security domain: State-of-the-art and challenges //Information and Software Technologies: 25th International Conference, ICIST 2019, Vilnius, Lithuania, October 10–12, 2019, Proceedings 25. – Springer International Publishing, 2019. – C. 256 - 270.
26. Zhao Y. et al. IDSRadar: a real-time visualization framework for IDS alerts //Science China Information Sciences. – 2013. – T. 56. – C. 1-12.
27. Sherkat E., Milios E. E., Minghim R. A Visual Analytics Approach for Interactive Document Clustering // ACM Trans. Interact. Intell. Syst. 2019. T. 10. № 1. C. 1–33.
28. Olszewski D. A clustering-based adaptive Neighborhood Retrieval Visualizer // Neural Networks. 2021. T. 140. C. 247–260.
29. Liu Y. та ін. Document Clustering Method Based on Visual Features // 2011 International Conference on Internet of Things and 4th International

Conference on Cyber, Physical and Social Computing. 2011. Т. 28. С. 458–462.

30. Tatu A. та ін. ClustNails: Visual analysis of subspace clusters // Tinshhua Sci. Technol. 2012. Т. 17. № 4. С. 419–428.
31. Palaniswami M. та ін. The Role of Visual Assessment of Clusters for Big Data Analysis: From Real-World Internet of Things // IEEE Syst. Man Cybern. Mag. 2020. Т. 6. № 4. С. 45–53.
32. Celebi M. E., Kingravi H. A., Vela P. A. A comparative study of efficient initialization methods for the k-means clustering algorithm // Expert Systems with Applications. 2013. Т. 40. № 1. С. 200–210.
33. Milosevic N. та ін. A framework for information extraction from tables in biomedical literature // IJDAR. 2019. Т. 22. № 1. С. 55–78.
34. Real-Time Analytics in Power BI | Analytics8. [Електронний ресурс]. URL: https://www.analytics8.com/blog/real-time-analytics-in-power-bi/ (дата звернення: 13.09.2025).
35. Fireteanu V.-V. Integrating Tableau with Internet of Things Acquiring Projects // 2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). 2020. С. 1–5.
36. Why power BI?. [Електронний ресурс]. URL: https://powerbi.microsoft.com/en-us/why-power-bi/ (дата звернення: 09.08.2024).
37. Chen L., Zhou H. Research and application of dynamic and interactive data visualization based on D3 // 2016 International Conference on Audio, Language and Image Processing (ICALIP). 2016. Т. 17. С. 150–155.
38. Kibana vs Grafana. [Електронний ресурс]. URL: https://uptrace.dev/comparisons/kibana-vs-grafana (дата звернення: 6.03.2026).
39. Thingsboard. [Електронний ресурс]. URL: https://thingsboard.io// (дата доступу: 09.08.2024).

40. Taivalsaari A., Mikkonen T. Visualizing Streaming Data in Industrial Scale IoT Systems // Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing (SAC ’25). – 2025. – P. 663–670. – DOI: 10.1145/3543712.3543750.
41. Khalid M. Y., Then P. H. H., Raman V. Exploratory Study for Data Visualization in Internet of Things // 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). 2018. Т. 8. С. 517 - 521.
42. Sun G.-D. та ін. A Survey of Visual Analytics Techniques and Applications: State-of-the-Art Research and Future Challenges // J. Comput. Sci. Technol. 2013. Т. 28. № 5. С. 852–867.
43. Bostock M., Ogievetsky V., Heer J. D³ Data-Driven Documents // IEEE Trans. Visual. Comput. Graphics. 2011. Т. 17. № 12. С. 2301–2309.
44. Attar-Khorasani S., Chalmeta R. Internet of Things Data Visualization for Business Intelligence // Big Data. – 2024. – Vol. 12, No. 6. – P. 478–503.
45. Akçit N., Tomur E., Karslıoğlu M. O. Importance of the spatial data and the sensor web in the ubiquitous computing area // SPIE Proceedings. 2014. Т. 9229. С. 922909.
46. Few S., Edge P. Data visualization effectiveness profile //Perceptual Edge. – 2017. – С. 12.
47. Huang W., Eades P., Hong S. H. Measuring effectiveness of graph visualizations: A cognitive load perspective //Information Visualization. – 2009. – Т. 8. – №. 3. – С. 139-152.
48. Boronat F., Lloret J., Garcia M. Multimedia group and inter-stream synchronization techniques: A comparative study //Information Systems. – 2009. – Т. 34. – №. 1. – С. 108-131.
49. Sivrikaya F., Yener B. Time synchronization in sensor networks: a survey //IEEE network. – 2004. – Т. 18. – №. 4. – С. 45-50.

50. Ravindran K., Bansal V. Delay compensation protocols for synchronization of multimedia data streams //IEEE Transactions on Knowledge and Data Engineering. – 1993. – Т. 5. – №. 4. – С. 574-589.
51. Akbar A. et al. Real-time probabilistic data fusion for large-scale IoT applications //Ieee Access. – 2018. – Т. 6. – С. 10015-10027.
52. Bannach D., Amft O., Lukowicz P. Automatic event-based synchronization of multimodal data streams from wearable and ambient sensors //Smart Sensing and Context: 4th European Conference, EuroSSC 2009, Guildford, UK, September 16-18, 2009. Proceedings 4. – Springer Berlin Heidelberg, 2009. – С. 135-148.
53. Henning S., Hasselbring W. Scalable and reliable multi-dimensional sensor data aggregation in data streaming architectures //Data-Enabled Discovery and Applications. – 2020. – Т. 4. – №. 1. – С. 5.
54. Schueller G., Behrend A. Stream fusion using reactive programming, LINQ and magic updates //Proceedings of the 16th International Conference on Information Fusion. – IEEE, 2013. – С. 1265-1272.
55. Griffiths I., Campbell L. Introduction to Rx.NET. – 2nd ed. – .NET Foundation, 2024. – 256 с.
56. Rx Part 9–Join, Window, Buffer and Group Join. Lee Campbell. [Електронний ресурс]. URL:: https://leecampbell.com/2011/03/14/rx-part-9-join-window-buffer-and-group-join/ (дата звернення: 16.08.2024).
57. Languages. ReactiveX. [Електронний ресурс]. URL:: https://reactivex.io/languages.html (дата звернення: 15.3.2024).
58. ReactiveX - Join operator [Електронний ресурс]. URL: https://reactivex.io/documentation/operators/join.html (дата звернення: 08.08.2024).
59. Лук’янець М.О., Сулема Є.С. Метод угрупування темпоральних мультимодальних потокових даних пристроїв IoT на основі

реактивного програмування // Системні технології. – 2024. – Т. 4, № 153. – С. 123–132.

60. Лук’янець М. Угрупування темпореальних мультимодальних потокових даних пристроїв IoT: особливості часової синхронізації // Матеріали конференцій МЦНД. – Харків, 10.05.2024. – С. 117–118.
61. 101 Rx Samples - a work in progress - Reactive Framework (Rx) Wiki [Електронний ресурс]. URL:http://rxwiki.wikidot.com/101samples (дата звернення: 08.08.2024).
62. Il-Agure Z., Dempere J. Review of data visualization techniques in IoT data // Proc. 8th Int. Conf. on Information Technology Trends (ITT). – IEEE, 2022. – С. 167–171.
63. Protopsaltis A., et al. Data visualization in Internet of Things: tools, methodologies, and challenges // Proc. 15th Int. Conf. on Availability, Reliability and Security. – 2020. – С. 1–11.
64. Hamza-Lup F. G., Iacob I. E., Khan S. Web-enabled intelligent system for continuous sensor data processing and visualization // Proc. 24th Int. Conf. on 3D Web Technology. – 2019. – С. 1–7.
65. Лук’янець М. Проблеми та можливі рішення оптимізації відображення даних мереж IoT у реальному часі // Collection of Scientific Papers «ΛΌГОΣ». – Zurich, 24.05.2024. – С. 190–191.
66. Лук’янець М., Сулема Є. Метод відображення точкових темпоральних мультимодальних потокових даних пристроїв IoT зі збереженням стабільності відображення попередньо візуалізованих даних // Herald of Khmelnytskyi National University. Technical Sciences. – 2025. – Т. 347, № 1. – С. 245–250.
67. James G., Witten D., Hastie T., Tibshirani R. An Introduction to Statistical Learning: with Applications in Python. – Cham: Springer, 2023. – Розд. «Resampling methods», С. 201–228.
68. SciChart. ResamplingMode (WPF Documentation). – [Електронний ресурс]. URL:

https://www.scichart.com/documentation/win/current/SciChart.Data~SciChart.Data.Numerics.ResamplingMode.html (дата звернення: 4.1.2025).
69. Dong J. et al. DFPS: An Efficient Downsampling Algorithm Designed for the Global Feature Preservation // Sensors. – 2025. – Vol. 25, № 17. – 5549.
70. Succetti M. Adaptive Embedding approach for time series forecasting. / Michele Succetti. – PhD Thesis, Sapienza University of Rome, 2024. – 150 p.
71. Lin L. Feature-Preserving Point Cloud Simplification Based on Adaptive Graph Filtering. // Journal of Computing in Civil Engineering. – 2025. – Vol. 40, № 1. – P. 34-45.
72. Li H., Yang B., Chua Y. FPCS: Feature Preserving Compensated Sampling of Streaming Time Series Data // IEEE Transactions on Visualization and Computer Graphics. – 2025. – Т. 31, № 1. – С. 1333–1342.
73. SciChart. WPF 2D Chart Library Features. – [Електронний ресурс]. URL:: https://www.scichart.com/wpf-chart-features/ (дата звернення: 4.1.2025).
74. SciChart. WPF 3D Chart Library Features. – [Електронний ресурс]. URL:: https://www.scichart.com/wpf-3d-chart-features (дата звернення: 4.1.2025).
75. BenchmarkDotNet: Powerful .NET library for benchmarking. – URL: https://github.com/dotnet/BenchmarkDotNet (дата звернення: 4.1.2025).
76. Лук’янець М.О., Сулема Є.С. Алгоритм оптимізації процесу відображення темпоральних мультимодальних потокових даних пристроїв IoT // ПМК’2024: зб. тез XVII конф. магістрантів та аспірантів (20–22.11.2024, Київ). – Київ, 2024. – С. 159–163.
77. W3C. WebGPU. – [Електронний ресурс]. URL:: https://www.w3.org/TR/webgpu/ (дата звернення: 4.1.2025).

78. W3C. WebGPU Shading Language (WGSL). – [Електронний ресурс]. URL:: https://www.w3.org/TR/WGSL/ (дата звернення: 4.1.2025).
79. Khronos Group. Vulkan 1.3 and Roadmap 2022. – [Електронний ресурс]. URL: https://www.khronos.org/blog/vulkan-1.3-and-roadmap-2022 (дата звернення: 4.1.2025).
80. Khattach D., Hassan M. M., Moubayed A., et al. End-to-End Architecture for Real-Time IoT Analytics and Predictive Maintenance: A Comprehensive Survey // Sensors. – 2025. – Vol. 25, № 9. – Art. 2945.
81. Tusa F. Privacy-Preserving FaaS: A Marketplace for the Serverless Edge-Cloud Continuum. // IEEE ICCCN 2025. – 2025. – P. 1-8.
82. Laska M., et al. A scalable architecture for real-time stream processing of spatiotemporal IoT data // ISPRS Int. J. Geo-Information. – 2018. – С. 238.
83. Dang-Ha T. H., Roverso D., Olsson R. Graph of virtual actors (GOVA): A big data analytics architecture for IoT // 2017 IEEE BigComp. – 2017. – С. 162–169.
84. OPC Foundation. OPC UA Part 1: Overview and Concepts, v1.05.04. – [Електронний ресурс]. URL: https://reference.opcfoundation.org/Core/Part1/v105/docs/ (дата звернення: 16.7.25).
85. OGC SensorThings API Part 1: Sensing. Version 1.1. – [Електронний ресурс]. URL: https://docs.ogc.org/is/18-088/18-088.html (дата звернення: 16.7.25).
86. OGC SensorThings API Standard (overview). – [Електронний ресурс]. URL: https://www.ogc.org/standards/sensorthings/ (дата звернення: 16.7.25).
87. HL7. FHIR Release 5 – History. – [Електронний ресурс]. URL:: https://build.fhir.org/history.html#v5.0.0 (дата звернення: 16.7.25).
88. Confluent Platform. Supported Versions and Interoperability. – [Електронний ресурс]. URL:

https://docs.confluent.io/platform/current/installation/versions-interoperability.html (дата звернення: 16.7.25).

89. Obaidat M. S. et al. Internet of Things: Device Capabilities, Architectures, Protocols, and Smart Applications. // IEEE Internet of Things Journal. – 2023. – Vol. 10, № 4. – P. 3611-3641.
90. Kiss T. et al. Decentralised Orchestration of Microservices in the Cloud-to-Edge Continuum. // 16th International Workshop on Science Gateways (IWSG2024). – 2024. – P. 1-10.
91. Лук'янець М., Сулема Є. Високорівнева архітектура системи відображення потокових темпоральних мультимодальних даних мереж пристроїв IoT // Herald of Khmelnytskyi National University. Technical Sciences. – 2025. – Т. 355. – №. 4. – С. 303-308.
92. MQTT. Message Queuing Telemetry Transport. – [Електронний ресурс]. URL: https://mqtt.org/ (дата звернення: 16.7.25).
93. Apache Kafka. – URL: https://kafka.apache.org/ (дата звернення: 16.7.25).
94. Apache Kafka – Kafka Streams Documentation. – [Електронний ресурс]. URL: https://kafka.apache.org/documentation/streams/ (дата звернення: 16.7.25).
95. Reactive Streams Specification. – [Електронний ресурс]. URL: https://www.reactive-streams.org/ (дата звернення: 16.7.25).
96. Akidau T., et al. The Dataflow Model: Balancing Correctness, Latency, and Cost // Proc. VLDB Endowment. – 2015. – 8(12). – P. 1792–1803.
97. Apache Flink Documentation (LTS). – [Електронний ресурс]. URL: https://nightlies.apache.org/flink/flink-docslts/(датазвернення: 16.7.25).
98. Apache Flink 2.0.0: A New Era of Real-Time Data Processing. – 24.03.2025. – [Електронний ресурс]. URL: https://flink.apache.org/2025/03/24/apache-flink-2.0.0-a-new-era-of-real-time-data-processing/ (дата звернення: 16.7.25).

99. Kubernetes Documentation. Production-grade Container Orchestration. – [Електронний ресурс]. URL: https://kubernetes.io/docs/home/ (дата звернення: 16.7.25).
100. IoT Hub | Microsoft Azure. – [Електронний ресурс]. URL: https://azure.microsoft.com/en-us/products/iot-hub/ (дата звернення: 16.7.25).
101. AWS IoT. – [Електронний ресурс]. URL: https://aws.amazon.com/iot/ (дата звернення: 16.7.25).
102. AWS Greengrass. – [Електронний ресурс]. URL:https://aws.amazon.com/greengrass/ (дата звернення: 16.7.25).
103. IoT Edge . – [Електронний ресурс]. URL:https://azure.microsoft.com/en-us/products/iot-edge (дата звернення: 16.7.25).
104. Gamma E. Design patterns: Elements of reusable object-oriented software / Gamma E., Helm R., Johnson R., Vlissides J. // Addison-Wesley. – 1994.

ДОДАТКИ

Додаток А. Реалізація замірювання продуктивності для групування 64 потоків даних

```
[MemoryDiagnoser]
public class Rx64GeneratorsLoadBenchmark
{
    private int _anyGroupCount;
    private int _pairGroupCount;
    private int _noPairGroupCount;

    private int _totalGenerated;
    [Params(200)] public int MasterIntervalMs { get; set; }
    [Params(250)] public int Count { get; set; }

    private List<LoadTestBaseDataGenerator<BaseDeviceData>>
_generators = new();
    private List<BaseStreamGrouper> _groupers = new();

    [IterationSetup]
    public void Setup() {
        _generators = new
List<LoadTestBaseDataGenerator<BaseDeviceData>>();
        _groupers = new List<BaseStreamGrouper>();

        var twoGroupsGrouper =
TreeBuildHelper.Get64GroupsGrouper(_generators, _groupers,
MasterIntervalMs, Count);

        twoGroupsGrouper.AnyGroupFormed +=
TwoSourcesGrouperOnAnyGroupFormed;
        twoGroupsGrouper.PairGroupFormed +=
TwoSourcesGrouperOnPairGroupFormed;
```

```
        twoGroupsGrouper.NoPairGroupFormed +=
TwoSourcesGrouperOnNoPairGroupFormed;

        foreach (var generator in _generators) {
            generator.DataReceived += OnDataGenerated;
        }
    }

    private void OnDataGenerated(IBaseDeviceData obj) {
        _totalGenerated++;
    }

    private void TwoSourcesGrouperOnNoPairGroupFormed(IDataGroup
obj) {
        _noPairGroupCount++;
    }

    private void TwoSourcesGrouperOnPairGroupFormed(IDataGroup obj){
        _pairGroupCount++;
    }

    private void TwoSourcesGrouperOnAnyGroupFormed(IDataGroup obj)
    {
        _anyGroupCount++;
    }

    [IterationCleanup]
    public void Cleanup()
    {
        foreach (var generator in _generators)
        {
            generator.StopGeneration();
        }
```

```
        foreach (var grouper in _groupers){
            grouper.StopGrouping();
        }

        _totalGenerated = 0;
        _noPairGroupCount = 0;
        _pairGroupCount = 0;
        _anyGroupCount = 0;
    }

    [Benchmark]
    public async Task Benchmark()
    {
        foreach (var grouper in _groupers){
            grouper.StartGrouping();
        }

        foreach (var generator in _generators)
        {
            generator.StartGeneration();
        }

        while (_generators.Any(x => x.IsRunning)) {   }

        await Task.Delay(150);

        Console.WriteLine("Total generated " + _totalGenerated);
        Console.WriteLine("No pair count " + _noPairGroupCount);
        Console.WriteLine("Pair count " + _pairGroupCount);
        Console.WriteLine("Any count " + _anyGroupCount);
    }
}
```

Додаток Б. Реалізація запропонованого шаблону інтерфейсу

```
<UserControl
x:Class="DataStreamCollectionAndVisaulization.MultipleDimensionTempl
ate"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-
namespace:DataStreamCollectionAndVisaulization"
             xmlns:s="http://schemas.abtsoftware.co.uk/scichart"
             xmlns:s3D="http://schemas.abtsoftware.co.uk/scichart3D"
             xmlns:viewModels="clr-
namespace:DataStreamCollectionAndVisaulization.ViewModels"
             mc:Ignorable="d"
             d:DataContext="{d:DesignInstance
Type=viewModels:MultipleDimensionTemplateViewModel,
IsDesignTimeCreatable=False}"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <s3D:SciChart3DSurface
            Grid.Column="0"
```

```
            Grid.RowSpan="4"
            x:Name="SciChart" BorderThickness="0"
WorldDimensions="200,100,200" >
            <s3D:SciChart3DSurface.Camera>
                <s3D:Camera3D ZoomToFitOnAttach="True"/>
            </s3D:SciChart3DSurface.Camera>
            <s3D:SciChart3DSurface.RenderableSeries>
                <s3D:ScatterRenderableSeries3D
x:Name="ScatterSeries3D"  DataSeries="{Binding DataSeries}">
                    <s3D:ScatterRenderableSeries3D.PointMarker>
                        <s3D:EllipsePointMarker3D Fill="Red"
Size="{Binding PointSize}" Opacity="1"/>
                    </s3D:ScatterRenderableSeries3D.PointMarker>
                </s3D:ScatterRenderableSeries3D>
            </s3D:SciChart3DSurface.RenderableSeries>
            <s3D:SciChart3DSurface.XAxis>
                <s3D:DateTimeAxis3D AxisTitle="Time"
VisibleRange="{Binding TimeRange, Mode=TwoWay}"
                    TextFormatting = "HH:mm:ss" GrowBy="0.1, 0.1"/>
            </s3D:SciChart3DSurface.XAxis>
            <s3D:SciChart3DSurface.YAxis>
                <s3D:NumericAxis3D VisibleRange="0, 100"
AxisTitle="{Binding AxisYName}" GrowBy="0.1, 0.1"/>
            </s3D:SciChart3DSurface.YAxis>
            <s3D:SciChart3DSurface.ZAxis>
                <s3D:NumericAxis3D VisibleRange="-30, 30"
AxisTitle="{Binding AxisZName}" GrowBy="0.1, 0.1"/>
            </s3D:SciChart3DSurface.ZAxis>
            <s3D:SciChart3DSurface.ChartModifier>
                <s3D:ModifierGroup3D>
                    <s3D:OrbitModifier3D  ></s3D:OrbitModifier3D>

<s3D:MouseWheelZoomModifier3D></s3D:MouseWheelZoomModifier3D>
```

```
                    <s3D:FreeLookModifier3D
ExecuteOn="MouseLeftButton" ExecuteWhen="Shift" IsEnabled="True"/>
                </s3D:ModifierGroup3D>

            </s3D:SciChart3DSurface.ChartModifier>

        </s3D:SciChart3DSurface>

        <StackPanel Orientation="Horizontal"
            Grid.Column="1"
            Grid.Row="0">
            <Label Content="Param A" Margin="2"
VerticalAlignment="Center"/>
            <CheckBox IsChecked="{Binding IsParamASelected}"
Margin="2"
                      VerticalAlignment="Center"/>
            <Label Content="Param B" Margin="2"
VerticalAlignment="Center"/>
            <CheckBox IsChecked="{Binding IsParamBSelected}"
Margin="2" VerticalAlignment="Center"/>
            <Label Content="Param C" Margin="2"
VerticalAlignment="Center"/>
            <CheckBox IsChecked="{Binding IsParamCSelected}"
Margin="2" VerticalAlignment="Center"/>
        </StackPanel>

        <s:SciChartSurface
            Grid.Column="1"
            Grid.Row="1"
            Name="ParamA" BorderThickness="0">
            <s:SciChartSurface.RenderableSeries>
                <s:XyScatterRenderableSeries
x:Name="ScatterSeries2D"  DataSeries="{Binding DataSeriesParamA}"
                           ResamplingMode="None">
```

```
                    <s:XyScatterRenderableSeries.PointMarker>
                        <s:EllipsePointMarker Fill="Red"
BorderBrush="Red" BorderThickness="{Binding PointSize}"
Opacity="1"/>
                    </s:XyScatterRenderableSeries.PointMarker>
                </s:XyScatterRenderableSeries>
            </s:SciChartSurface.RenderableSeries>

            <s:SciChartSurface.XAxis>
                <s:DateTimeAxis AxisTitle="Time"
VisibleRange="{Binding TimeRange, Mode=TwoWay}"
                    TextFormatting = "HH:mm:ss" GrowBy="0.1, 0.1"/>
            </s:SciChartSurface.XAxis>

            <s:SciChartSurface.YAxes>
                <s:NumericAxis VisibleRange="0, 100"
AxisTitle="Param A" GrowBy="0.1, 0.1"/>
            </s:SciChartSurface.YAxes>

            <s:SciChartSurface.ChartModifier>
                <s:ModifierGroup>
                    <s:MouseWheelZoomModifier ActionType="Zoom"/>
                    <s:YAxisDragModifier/>
                </s:ModifierGroup>
            </s:SciChartSurface.ChartModifier>

        </s:SciChartSurface>

        <s:SciChartSurface
            Grid.Column="1"
            Grid.Row="2"
            Name="ParamB"
            BorderThickness="0">
            <s:SciChartSurface.RenderableSeries>
```

```
                <s:XyScatterRenderableSeries DataSeries="{Binding
DataSeriesParamB}"
                                             ResamplingMode="None">
                    <s:XyScatterRenderableSeries.PointMarker>
                        <s:EllipsePointMarker Fill="Red"
BorderBrush="Red" BorderThickness="{Binding PointSize}"
Opacity="1"/>
                    </s:XyScatterRenderableSeries.PointMarker>
                </s:XyScatterRenderableSeries>
            </s:SciChartSurface.RenderableSeries>

            <s:SciChartSurface.XAxis>
                <s:DateTimeAxis AxisTitle="Time"
VisibleRange="{Binding TimeRange, Mode=TwoWay}"
                                TextFormatting = "HH:mm:ss"
GrowBy="0.1, 0.1"/>
            </s:SciChartSurface.XAxis>

            <s:SciChartSurface.YAxes>
                <s:NumericAxis VisibleRange="-30, 30"
AxisTitle="Param B" GrowBy="0.1, 0.1"/>
            </s:SciChartSurface.YAxes>

            <s:SciChartSurface.ChartModifier>
                <s:ModifierGroup>
                    <s:MouseWheelZoomModifier ActionType="Zoom"/>
                    <s:YAxisDragModifier/>
                </s:ModifierGroup>
            </s:SciChartSurface.ChartModifier>
        </s:SciChartSurface>
        <s:SciChartSurface
            Grid.Column="1"
            Grid.Row="3"
            Name="ParamC"
```

```
            BorderThickness="0">
            <s:SciChartSurface.RenderableSeries>
                <s:XyScatterRenderableSeries DataSeries="{Binding
DataSeriesParamC}"
                                             ResamplingMode="None">
                    <s:XyScatterRenderableSeries.PointMarker>
                        <s:EllipsePointMarker Fill="Red"
BorderBrush="Red" BorderThickness="{Binding PointSize}"
Opacity="1"/>
                    </s:XyScatterRenderableSeries.PointMarker>
                </s:XyScatterRenderableSeries>
            </s:SciChartSurface.RenderableSeries>

            <s:SciChartSurface.XAxis>
                <s:DateTimeAxis AxisTitle="Time"
VisibleRange="{Binding TimeRange, Mode=TwoWay}"
                                TextFormatting = "HH:mm:ss"
GrowBy="0.1, 0.1"/>
            </s:SciChartSurface.XAxis>

            <s:SciChartSurface.YAxes>
                <s:NumericAxis VisibleRange="0, 50" AxisTitle="Param
C" GrowBy="0.1, 0.1"/>
            </s:SciChartSurface.YAxes>

            <s:SciChartSurface.ChartModifier>
                <s:ModifierGroup>
                    <s:MouseWheelZoomModifier ActionType="Zoom"/>
                    <s:YAxisDragModifier/>
                </s:ModifierGroup>
            </s:SciChartSurface.ChartModifier>
        </s:SciChartSurface>
    </Grid>
</UserControl>
```

Додаток В. Реалізація запропонованого методу стабільного відображення

```
public partial class DataParameterViewModel<T> : ObservableObject
where T : BaseDeviceData
{
    [ObservableProperty] private IRange? _defaultViewRange;
    [ObservableProperty] private bool _isParamSelected;
    [ObservableProperty] private string _paramName = "";
    [ObservableProperty] private XyDataSeries<DateTime, double>
_dataSeries = new();
    public ParameterGetter<T> ParameterGetter { get; init; }

    public double MaxAcceptanceValue { get; init; }

    public event Action? IsParamSelectedChanged;

    public DataParameterViewModel(string paramName, double
maxAcceptanceValue, ParameterGetter<T> parameterGetter)
    {
        ParamName = paramName;
        MaxAcceptanceValue = maxAcceptanceValue;
        ParameterGetter = parameterGetter;
    }

    public bool IsAcceptable(T data, T dataToCompare)
    {
        return IsParamSelected &&
               Math.Abs(ParameterGetter.GetParameter(data) -
ParameterGetter.GetParameter(dataToCompare)) <
               MaxAcceptanceValue;
    }

    public void AppendData(List<T> data)
    {
```

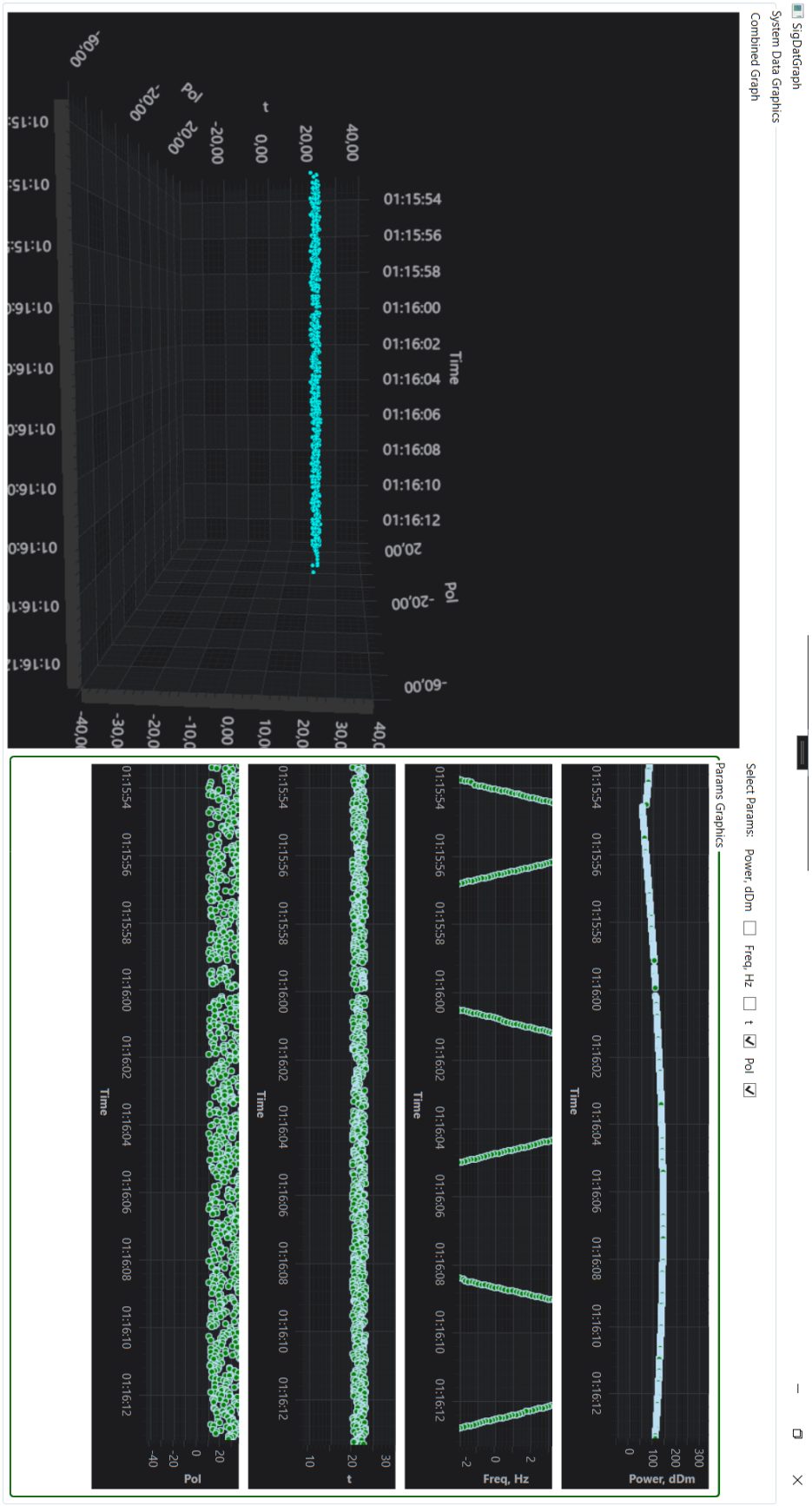
```
            DataSeries.Clear();
            DataSeries.Append(data.Select(x => x.Timestamp),
                data.Select(x => ParameterGetter.GetParameter(x)));
        }

        partial void OnIsParamSelectedChanged(bool value)
        {
           IsParamSelectedChanged?.Invoke();
        }
    }

    public class ReclusterClassPlaceholderExample(){

        private void ReclusterLastData()
        {
            if (DeviceParamList.Count(x => x.IsParamSelected) != 2)
            {
                return;
            }

            var now = DateTime.Now;
            var min = now - VariableDisplayTime;
            var totalMin = now - DeltaTime;
            _dataToDisplay.RemoveWhere(x => x.Timestamp > min ||
    x.Timestamp < totalMin);

            var dataToRecluster = _lastReceivedData.Where(x =>
    x.Timestamp > min).ToList();

            var reformatedClusters = new List<TestDeviceData>();

            foreach (var data in dataToRecluster)
            {
```

```
            var compatibleCluster =
reformatedClusters.FirstOrDefault(x => IsDataCanBeClusteredWith(x,
data));
            if (compatibleCluster == null)
            {
                reformatedClusters.Add(data);
            }
            else
            {
                // add metadata
            }
        }

        _dataToDisplay.AddRange(reformatedClusters);
    }

    private bool IsDataCanBeClusteredWith(TestDeviceData data,
TestDeviceData dataToCompare)
    {
        var selected = DeviceParamList.Where(x =>
x.IsParamSelected).ToList();
        if (selected.Count != 2)
        {
            return false;
        }

        return selected.All(x => x.IsAcceptable(data,
dataToCompare))
               && Math.Abs((data.Timestamp -
dataToCompare.Timestamp).TotalMilliseconds) <
               TimeAcceptable.TotalMilliseconds;
    }

    private void TimeRangeTimerOnTick(object? sender, EventArgs e)
```

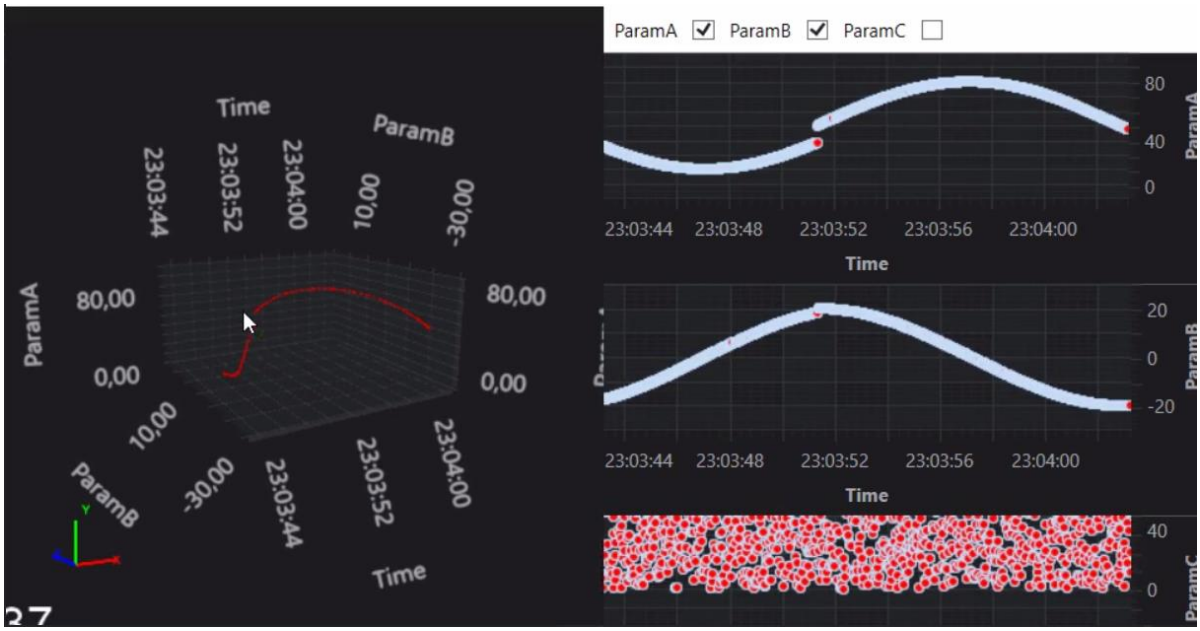
```
    {
        var now = DateTime.Now;
        var min = now - DeltaTime;
        TimeRange = new DateRange(min, now);
        _lastReceivedData.RemoveWhere(x => x.Timestamp < min);
    }
}

public interface IParameterGetter<in T> where T : BaseDeviceData
{
    public double GetParameter(T data);
}

public abstract class ParameterGetter<T> : IParameterGetter<T> where
T : BaseDeviceData
{
    public abstract double GetParameter(T data);
}
```

Додаток Б. Приклади інтерфейсу на основі розробленої схеми

Приклад інтерфейсу з 4 окремими параметрами, окрім часового



Приклад інтерфейсу з 3 окремими параметрами, окрім часового



Додаток В. Список публікацій здобувача за темою дисертації

1. Mykhailo Lukianets, Yevgeniya Sulema. Real-time data visualization for IoT network systems: challenges and strategies for performance optimization // System technologies. – 2023. – Vol. 5. № 148. – С. 52–61. Внесок М. О. Лук'янця: аналіз існуючих методів та систем візуалізації та виявлення можливих напрямків оптимізації роботи систем оброблення даних в мережах IoT. Внесок Є. С. Сулеми: визначено теоретичні засади оптимізації в межах систем IoT реального часу.
2. М.О. Лук'янець, Є.С. Сулема. Метод угрупування темпоральних мультимодальних потокових даних пристроїв IoT на основі реактивного програмування // Системні технології. 2024. –Т. 4. – № 153. – С. 123–132. Внесок М. О. Лук'янця: розроблення методу групування даних, його алгоритмізація та перевірка ефективності. Внесок Є. С. Сулеми: обґрунтування підходу до застосування парадигми реактивного програмування для агрегації мультимодальних даних.
3. Лук'янець М., Сулема Є. Метод відображення точкових темпоральних мультимодальних потокових даних пристроїв IoT зі збереженням стабільності відображення попередньо візуалізованих даних // Herald of khmelnytskyi national university. Technical sciences. – 2025. – Т. 347. – №. 1. – С. 245-250. Внесок М. О. Лук'янця: розроблення алгоритму стабілізації відображення та проведення серії експериментів для підтвердження роботи методу. Внесок Є. С. Сулеми: розроблення концепції забезпечення візуальної стабільності при відображенні високоінтенсивних потоків даних.
4. Лук'янець М., Сулема Є. Високорівнева архітектура системи відображення потокових темпоральних мультимодальних даних мереж пристроїв IoT // Herald of khmelnytskyi national university. Technical sciences. – 2025. – Т. 355. – №. 4. – С. 303-308. Внесок М. О. Лук'янця: розроблення концепції побудови архітектури системи відображення, розроблення функціональних модулів та механізмів оброблення

потокових даних. Внесок Є. С. Сулеми: визначення принципів взаємодії компонентів.

5. Лук’янець, М. (2024). Угрупування темпореальних мультимодальних потокових даних пристроїв IoT: особливості часової синхронізації // *Матеріали конференцій МЦНД.* - Харків:10.05.2024. – С. 117–118.
6. Лук’янець , М. (2024). Проблеми та можливі рішення оптимізації відображення даних мереж IoT у реальному часі // *Collection of Scientific Papers «ΛΌГОΣ».* – Zurich: May 24, 2024. – С. 190–191.
7. М.О. Лук’янець, Є.С. Сулема. (2024) Алгоритм оптимізації процесу відображення темпоральних мультимодальних потокових даних пристроїв IoT // Прикладна математика та комп’ютинг ПМК' 2024 : збірник тез доповідей Сімнадцятої конференції магістрантів та аспірантів (20-22 листопада 2024 р. Київ, Україна). – Київ, 2024. – С. 159-163. Внесок М. О. Лук’янця: розроблення алгоритму оптимізації процесу відображення даних пристроїв IoT. Внесок Є. С. Сулеми: визначення теоретичних засад оптимізації продуктивності систем візуалізації в реальному часі.
8. Лук’янець, М. (2026). Шаблон проєктування Multimodal Stream Stable Visualization (MSSV) для відображення мультимодальних потокових даних у системах реального часу // 3 Міжнародна науково-практична конференція Innovative Research in Science and Economy. – Брюссель: Січень 2026.