Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: «Розробка системи уніфікації потоків даних на мові програмування C#, платформа .Net»

Виконав(ла): студент(ка) VI курсу, групи СПм-61
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Дударчук В.С.
	(підпис)	(прізвище та ініціали)
Керівник		Цуприк Г.Б.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Бойко І.В.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М.Р.
	(підпис)	(прізвище та ініціали)
Рецензент		Лупенко С.А.
	(підпис)	(прізвище та ініціали)

Тернопіль
2020

АНОТАЦІЯ

Дударчук В.С. Розробка системи уніфікації потоків даних на мові програмування C#, платформа .Net. – Рукопис.

Кваліфікаційна робота здобуття освітнього ступеня магістр за спеціальністю 121 – Інженерія проґрамного забезпечення. – Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і проґрамної інженерії, кафедра проґрамної інженерії, група СПм-61 // м.Тернопіль, 2020 // С. , рис. – 23, табл. – , додат. – , бібліогр. –

Метою кваліфікаційної роботи є розробка спеціалізованої комп’ютерної системи, робота якої передбачає певний визначений алгоритм виявлення тренду ринку і, залежно від його тенденції, прийняття відповідного рішення та здійснення відповідного втручання.

Практичне застосування – розроблено систему, яку рекомендовано до використання в усіх видає діяльності в якій використовується прогнозування, пошук шаблонів, прийняття рішень та аналіз в режимі реального часу.

Технічні вимоги – методи розробки базуються на технології C#, реалізація на платформі .Net.

Ключові слова: СИСТЕМА, ПРОГНОЗУВАННЯ, АЛГОРИТМ, ТЕХНІЧНІ ВИМОГИ, АРХІТЕКТУРА, МОДЕЛЬ, ЯКІСТЬ, РЕАЛІЗАЦІЯ, ОХОРОНА ПРАЦІ.

ABSTRACT

Dudarchuk V.S. Development of a system for unification of data flows in the C # programming language, .Net platform. – Manuscript.

Qualifying work for obtaining a master's degree in specialty 121 — Software Engineering. – Ternopil Ivan Pul'ui National Technical University, Faculty of Computer Information Systems and Software Engineering, Software Engineering Department, group SPm-61 // Ternopil, 2020 //

Pages. – , pictures. – 23, tables. – , supp. – , bibl.ref. –

The purpose of skilled work is to develop a specialized computer system. The operation of the system involves a constant defined algorithm for detecting market trends. It will give a possibility to make a decision and make correction.

Practical application - a system has been developed that is recommended for use in all activities use forecasting, template search, decision making and real-time analysis.

Technical requirements - development methods are based on C # technology, implementation on the .Net platform.

Keywords: SYSTEM, FORECASTING, ALGORITHM, TECHNICAL REQUIREMENTS, ARCHITECTURE, MODEL, QUALITY, IMPLEMENTATION, LABOR PROTECTION.

ЗМІСТ

ВСТУП

Актуальність теми замовлена тим, що обраний напрям дослідження, з фінансової точки зору, є одним із найнестабільніших в світі. В один момент можна стати як фінансово успішним так і, вже за короткий час, втратити все. Саме тому архіважливою задачею є вчасне реагування на будь-які, навіть на перший погляд несуттєві, зміни, які дуже швидко можуть набути лавиноподібного характеру і добре, якщо в позитивну сторону, оскільки якщо таке стається – вплинути на ситуацію достатньо важко і залишається лише очікувати хоча б її мінімальної стабілізації. В цьому питанні не можна покладатись на долю, а варта підійти з наукової точки зору, наприклад, використавши відомий математичний апарат – математичну модель.

Одним із важливих понять, без розуміння якого неможлива постановка задачі є поняття тренду - це направленість руху ціни активу, яка сформується в рамках певного часового проміжку. Досить часто за синонім даного поняття використовують термін «тенденція», який можна означити як найбільш ймовірний напрямок руху ціни в перспективі, який визначається її попереднім і поточним станом. В обидвох варіантах найважливішим (ключовим) аспектом прогнозування, як цінових рухів так і торгівлі в цілому, є поняття цілеспрямованості метою якої є, в кінцевому результаті, отримання прибутку, числове значення якого формується на корегуванні чи зміні вартості активу в наперед обраному напрямку.

Завданням будь-якої спеціалізованої комп’ютерної системи є робота з коректною математичною моделлю, в якій максимально враховано всі критерії, які можуть суттєво чи навіть критично впливати на тенденцію ринку в цілому (це і комерція безпосередньо, і робота фондового ринку, і біржева активність в цілому). Вдало складений алгоритм, в свою чергу, дасть змогу співставити реально існуючі параметри та коректно визначити ті, що можуть позитивно вплинути на проведення фінансових операцій та уникнути кроків які б, в свою чергу, негативно впливали на кінцевий результат.

Об’єктом дослідження обрано актуальний на сьогодні напрямок, а саме: комерційна діяльність, а якій передбачено формалізований процес здійснення торгових операцій на фінансових ринках за заданим алгоритмом з використанням спеціалізованих комп’ютерних систем.

Предметом дослідження є програмний продукт, яким є спеціалізована комп’ютерна система, робота якої передбачає певний визначений алгоритм виявлення тренду ринку і, залежно від його тенденції, прийняття відповідного рішення та здійснення відповідного втручання.

Наукова новизна роботи полягає в адаптації та доопрацюванні системи комплексної обробки подій при використанні комп’ютерних алгоритмів для досягнення визначеної трейдової мети шляхом розбивання інформації на частини та рознесення її в просторі та часі, та яку можна використати для проектування подібних систем для вирішення інших задач.

Прогнозується, що розроблений програмний продукт матиме важливе практичне значення для обраного напрямку дослідження. Його рекомендовано до використання в усіх видах комерційної діяльності в яких робота ведеться з великими об’ємами даних, що надходять з різних джерел та потребують приведення до єдиного формату, та які використовуються для прогнозування, пошуку шаблонів, прийняття рішень та аналіз в режимі реального часу.

Оформлення кваліфікаційної роботи роботи буде виконане відповідно до діючих стандартів: ДСТУ 2391-94. «Система технологічної документації. Терміни та визначення»; ДСТУ 3008-95. «Документація. Звіти. Структура і правила щодо оформлення», а також ЕСКД та іншим чинним стандартам.

1 АНАЛІТИЧНИЙ ОГЛЯД В ОБЛАСТІ ДОСЛІДЖЕНЬ

1.1 Аналіз архітектури системи

Подія можна визначити як «суттєва зміна стану» [1]. Наприклад, коли ви придбали якийсь товар, його статус зміниться з «продається» на «проданий». Системна архітектура продавця товарів будь-якого типу (аж до типу товарів, які підлягають від визначення рухоме та нерухоме майно) може розглядати таку зміну статусу як подію, що створюється, оприлюднюється, визначається і використовується різними додатками в складі архітектури.

Такий архітектурний шаблон використовується при розробці, а також реалізації програм та систем, через які передаються події посеред мало пов’язаних між собою програмних компонентів й служб. Система, керована подіями, зазвичай може містити джерела подій (чи агентів). Також її складовими є і споживачі подій (або стоки, які як правило є відповідальними за реакцію, як тільки подія настала. Така «реакція» може повністю чи ні (не повністю) створюватися стоком. Наприклад, сток може відповідати виключно за фільтрацію, перетворення і супровід («доставку», передачу) події до наступної (іншої) компоненти. Також ним може бути створена й інша, якась власна реакція на цю таку подію. До першої категорії можна віднести стоки для яких характерні традиційні компоненти. Зокрема, сюди можна віднести проміжне програмне забезпечення призначене для обміну повідомленнями. До другої категорії відносяться стоки, які в результаті формування власної реакції в процесі роботи, можуть потребувати наявності більш підходящої платформи для виконання відповідних транзакцій.

Створення додатків і систем в межах архітектури, керованої подіями, дає можливість конструювання їх таким способом, який сприятиме покращенню показників інтерактивності, оскільки системи, керовані

подіями, за структурою більш орієнтовані на непередбачувані і асинхронні оточення [2].

Архітектура, керована подіями відповідає критеріям сервіс-орієнтованої архітектури (SOA), оскільки такого типу сервіси як правило активуються тригерами, що спрацьовують від вхідних подій [2] [3].
Така парадигма є особливо корисною у випадках, коли сток не надає алгоритму власного виконання дій.

Сервіс-орієнтована архітектура, керована подіями розвиває архітектури SOA і EDA для надання більш глибокого і надійного рівня послуг за рахунок використання раніше невідомих причинно-наслідкових зв'язків, щоб сформувати нову модель подій. Цей достатньо новий шаблон бізнес-аналітики призведе до подальшої автоматизації обробки, що додає нових можливостей підприємству, які були неосяжними раніше для суттєвого підвищення продуктивності шляхом внесення цінної інформації в розпізнаний шаблон діяльності [4, 5].

Таким чином, виходячи з вище наведеного подію варта трактувати як суттєву зміну стану. Якщо формалізувати, то те, що виробляється чи створюється, публікується чи видається, поширюється чи розповсюджується, виявляється та може споживатись (і все це зазвичай асинхронно) є ні що інше ніж повідомлення, яке ще часто можуть називати нотифікацією (або сповіщенням про подію), а не власне самою подією, яка безпосередньо і є тим чинником, який викликає зміну статусу, викликану появою повідомлення.

Зазвичай так звана ПОС(подійно-орієнтована система) може складатись з генераторів (емітерів) таких подій та безпосередньо користувачів стоків (або подій). В свою чергу стоки «відповідають» за вчасну реакцію на настання такої події. Потрібно уточнити, що сама реакція не завжди забезпечується в повному об'ємі безпосередньо самим стоком. Для прикладу, сток може «відповідати» лише за окремі функції, наприклад фільтрація, трансформація і відправка (переадресація) до наступного

(іншого) компонента; крім цього ним може забезпечуватись повністю самостійна реакція на певну конкретну подію про яку отримано інформацію та на яку потребується реакція [1].

В даному питанні існує два типи архітектур EDA (англійською: event-driven architecture, українською: подійно-орієнтована архітектура) та SOA (англійською: service-oriented architecture, українською: сервісно-орієнтована архітектура). При чому перша може доповнювати другу, тому що безпосередньо служби-сервіси можуть активуватись тригерами, що активізуються-ініціюються при виникненні події [1,2]

Функції опрацювання подій можна розділити на прості і складні, грунтуючись в першу чергу на складності.

Прості події – це події, які не об'єднують і не представляють набір інших подій, фільтруються і маршрутизуються без зміни. Таким чином, коли відбувається подія, вона ініціює наступну дію (дії), і кожна поява події обробляється незалежно. Незважаючи на назву "прості", ці події можуть мати велике значення і надавати важливу бізнес-інформацію. Події можуть перетворюватися шляхом трансляції, розбиття чи злиття однієї або декількох подій. Проста обробка включає в себе такі види обробки як:

- перетворення схеми подій з однієї форми в інші;
- поповнення події корисним навантаженням з додатковими даними;
- перенаправлення події з одного каналу або потоку в інший;
- генерування кількох подій на основі корисного навантаження однієї події.

Складна обробка подій передбачає виявлення шаблонів, що охоплюють кілька незалежних подій, для породження нових "складних" подій. Складна обробка подій включає в себе обробку наборів подій з метою виявлення важливої бізнес-ситуації. Зазвичай така обробка означає застосування до набору подій ряду умов або обмежень. Події (значущі або звичайні) можуть бути різнотипними і відбуватися протягом певного періоду часу. Вони можуть корелюватися включаючи причинно-наслідковий зв'язок, тимчасове і

просторове співвідношення. Необхідно розуміти, що отримана при складній обробці подій інформація є складною і багатою за своєю природою, але не є, або не повинна бути, складною для користувача.

Обробка подій може бути об'єднана з аналітичними методами для передбачення подій, пошуку шаблонів і породження подій аналізу в режимі реального часу. Використання таких методів для аналізу подій в режимі реального часу дозволяє приймати більш обгрунтовані рішення.

1.2 Обґрунтування актуальності розробки

Базові принципи обробки подій вже давно широко використовуються в інтеграційному програмному забезпеченні проміжного рівня і в різних формах системного програмного забезпечення (наприклад, в операційних системах) [6].

Однак значущою така система стане лише за умов легкої маштабованості та наявності механізму який дозволив би визначати дійсно важливі події (шаблони подій) з поміж великої кількості інших подій. Це, у свою чергу, дозволить швидко реагувати на нові можливості і загрози, поширювати релевантну інформацію, організовувати активну діагностику проблем і брати участь у формуванні загального уявлення про стан речей в режимі реального часу.

Така система допоможе ідентифікувати тенденції і конкурентні загрози, надати можливості для зниження ризиків, сприяти більш швидкому отриманню прибутку, зменшити час виявлення і реагування.

Ще однією причиною актуальності такої системи обробки подій є необхідність переходу з пакетної обробки на обробку в режимі реального часу для більш швидкого прийняття рішень. Характеристики нових робочих навантажень теж вимагають близькою до реального часу обробки складних подій, що включають в себе не тільки події від звичайних джерел даних, а й від таких джерел, як звук і відео.

Галузеві аналітики стверджують, що різного роду події (від простих до складних) будуть все більш широко застосовуватися в бізнес-додатках. Реалізація керованих подіями бізнес-процесів дає величезні фінансові та стратегічні переваги, оскільки цей підхід відповідає природі багатьох аспектів реального світу.

Керовані подіями бізнес-процеси не тільки прискорюють традиційні процеси, але і мають специфічні властивості, що відрізняють їх від традиційних підходів. Керовані подіями програми дозволяють процесам швидко змінюватися і реагувати на помилки і виняткові ситуації, що порушують звичайні процеси. В умовах, коли підприємства прагнуть знизити витрати і поліпшити оперативність роботи з клієнтами, постачальниками і світом у цілому, концепція керованого подіями дизайну стає все більш популярною. Підприємства бачать переваги реалізації керованих подіями бізнес-процесів не тільки в тому, що вони дають конкурентні переваги з точки зору витрат і термінів окупності.

Для того щоб система була масштабованою вона не повинна залежати від конкретної предметної області чи бізнес-процесу. Спроектувати її потрібно таким чином, щоб вона легко інтегровувалася у вже існуючі системи з різноманітними джерелами даних та забезпечувала весь потрібний функціонал.

Система повинна реалізовувати архітектуру не просто обробки подій а саме складних подій. Ідея складних подій випливає із спостережень, що при деяких сценаріях, значення не завжди пов'язані з якою-небудь конкретною подією або записом про подію. Замість цього, ми берем до уваги значення результату кількох подій, найчастіше отриманих в різний час, як частина різних потоків. У цьому випадку важливі дані отримуються методом обробки різних подій.

В такому випадку подія повинна не лише сповіщати про факт свого настання, але й нести вичерпний опис для реалізації її комплексної обробки. Тому вона повинна містити щонайменше з дві частини: заголовк події і тіло

події. Заголовок події може містити інформацію про, для прикладу, назву події, часову мітку події і тип події. Тілом події називають частину, в якій описується факт, який настув в реальності. Тіло події не слід плутати з шаблоном або логікою, яка може бути застосована в якості реакції на саму події.

Система повинна базуватися на простих подіях для того, щоб її можна було інтегрувати у будь яку існуючу систему. Проста подія реєструється безпосередньо з датчика, що повідомляє про подію і надходить у систему. Складне подія отримується шляхом виявлення шаблонів з різних потоків примітивних подій. Для прийняття рішення, ми можемо спиратися на ланцюжка попередніх подій, інформацію з баз даних і довідкові дані. У системі це призводить до концепції ієрархії подій. Примітивні події агрегуються на більш високому рівні подій, які самі в подальшому можуть об'єднуватися в події ще вищого рівня. У міру просування вгору за рівнями абстракції подій ми вимушені використовувати все більш складні форми аналізу і робити висновки на все більш широкій агрегації різних типів подій.

Отже така система може використовуватися в різних сценаріях, для яких необхідні висока пропускна здатність і малі затримки: спостереження за виробничими процесами та управління процесами, аналітика, фінансові служби і торгівля, моніторинг IT-середовища, телекомунікації.

1.3 Постановка задачі

Щоб спроектувати систему і ідеєю комплексної обробки подій [7] перш за все необхідно скласти список ключових вимог до системи такого типу:

- відсутні прив’язки спроектованої системи до конкретної предметної області;
- легка масштабованість системи;
- система повинна опрацьовувати від 100 до 1000 подій в секунду, залежно від їх складності;

- система повинна бути багатопотоковою для пришвидшення швидкодії;
- параметри багатопочності системи повинні мати можливість конфігурування;
- система повинна надавати базові інтерфейси для реалізації вхідних та вихідних адаптерів, що будуть відповідати за вхідні та вихідні данні в системі;
- система повинна надавати інтерфейси базових фільтрів для агрегації подій та визначення шаблонів подій;
- система повинна надавати інтерфейс та можливість реалізації проекції – обчислення для утворення додаткових властивостей подій;
- система повинна підтримувати необмежену кількість рівнів абстракцій подій;
- вихідні адаптери повинні мати можливість перенаправляти потоки подій на вхід системи;
- система повинна реалізовувати основні сутності такі як подія, визначати її параметри та надавати можливість розширення;
- система повинна реалізовувати механізм реєстрації подій та часових міток;
- система повинна надавати доступ до реєстру подій за певний період часу;
- систем повинна підтримувати 3 типи подій залежно від часових характеристик:
    1) інтервальні події;
    2) точкові;
    3) граничні;
- архітектура системи повинна бути якомога краще розподіленою, така вимога є однією з ключових та обумовлюється тим, що за подію можна прийняти будь-що, за означенням, а присутньою

подія можу бути де завгодно, тобто її існування нічим не обмежується;

- архітектура системи повинна слабо зв’язуватись, оскільки події як такій «не відомо» нічого щодо наслідків свого настання.

Для досягнення поставленої мети потрібно вирішити наступні задачі:

- визначити логічний рівень в системі який буде відповідати за уніфікацію вхідних та вихідних даних;
- визначити протоколи даних які будуть достатніми з функціональної точки зору для передачі даних між рівнями системи;
- визначити які методи обробки даних будуть доступні в системі та як вони можуть бути розширеними при потребі;
- спроектувати оптимальну концептуальну модель системи;
- спроектувати оптимальну концептуальну архітектуру системи;
- визначити та реалізувати ті особливості системи які будуть необхідні саме для сфери алгоритмічної торгівлі.

1.4 Глосарій

Глосарій містить описи термінів, що використовуються при проектуванні та реалізації системи, визначає основні поняття, безпосередньо пов'язані з проектуванням та проходженням тестувань.

Глосарій створений в рамках даного проекту та містить такі основні терміни:

- подія;
- адаптер;
- генератор подій;
- канал подій;
- інтервальна подія;

- точкова подія;
- гранична подія;
- реєстр подій;
- шаблон (або патерн) події;
- проста подія;
- складна подія;
- уніфікація даних.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Моделювання архітектури системи

Якщо говорити про архітектуру [8] як про поняття, то варта зазначити ,що загального чи навіть узагальненого, єдиного терміну «архітектура програмного забезпечення» не існує. Однак якщо говорити про практику, то для великої кількості розробників відповідь на запитання добрим є код чи поганим є очевидною і однозначною. Хорошою архітектурою перш за все варта назвати таку, яка є «вигідною» з точки зору спрощення та підвищення ефективності процесу розробки і супроводу програми . Не секрет що процес розширення та зміна програми з «хорошою» архітектурою є значно простішим. З добро пропрацюваною архітектурою легше розуміються та проходять і процеси тестування та наладки. Таким чином варта підсумувати та окреслити перелік зрозумілих та універсальних критеріїв, за яких архітектуру програмного продукту можна назвати адекватною поставленій задачі та оптимальною.

1.Насамперед система повинна вміти справлятись з поставленими задачами і ефективно виконувати свої функції незважаючи на умови. Надійність, безпека, продуктивність, здатність ефективно виконувати функції при збільшенні навантаження (масштабованість) – це все характеристики ефективності системи в цілому.

2. Не секрет що колись приходить час змін будь-якого додатку, оскільки вимоги можуть не лише мінятись але й додаватись нові. І власне швидкість процесу та зручність внесення змін до функціоналу і є основними критеріями гнучкості та конкурентоздатності системи. Також дуже важливим є і те, що такі зміни повинні викликати мінімальну кількість помилок, або ж не викликати їх взагалі. Якщо так стається, то ми говоримо про достатню гнучкість системи.

Таким чином в процесі розробки я буду оцінювати те, що отримав з погляду того, що мені може і прийдеться вносити зміни. І лише цим будуть аргументуватись ті кроки, які я приймаю. Тут також важливим є і те, що потрібно чітке розуміння того, що зміна жодного фрагмента системи не повинно впливати на інші її фрагменти. В цілому ж концепція така, що якщо можливо, то, архітектурні рішення не повинні «вирубуватись в камені», а наслідки архітектурних помилок повинні бути зведені до мінімуму, якщо їх не вдалось все таки уникнути повністю.

3.Можливість розширення системи. Можливість додавати в систему нові сутності та функції, не порушуючи її основної структури. На початковому етапі в системі варта враховувати закладання тільки основного та найнеобхіднішого функціоналу. Однак поруч із цим архітектурно варта передбачити можливість легкого нарощування додаткового функціоналу в міру необхідності. При цьому це потрібно зробити таким чином, що внесення найймовірніших змін повинне потребувати мінімальних зусиль.

4. Вимога до гнучкості та розширюваності архітектури системи. Тут йдеться про можливість вносити зміни та еволюціонувати. Це є вкрай важливо і сформульовано в окремий принцип - принципу відкритості закритості (Open-Closed Principle – він є другим із 5-ти принципів SOLID). Тут йдеться про те, що програмні суті, як наприклад, клас, модулі, функції та інші, варта робити доступними для розширення, однак вони точно повинні бути не доступними (закритими) для модифікування. Якщо говорити іншими словами ,то потрібно мати можливість змінювати чи розширювати поведінку системи без можливості змінювати / переписувати вже існуючі частини системи.

Для мене це значить, що систему потрібно спроектувати таким чином, щоб зміну її поведінки і додавання нового функціоналу можна було б

досягнути після створення/написання нового коду - розширення). Однак, це не призвело б до необхідності вносити зміни в вже написаний (існуючий) код. Таким чином необхідність врахування нових вимог не викличе потребу модифікувати вже існуючу логіку. Цього можна досягнути за рахунок можливості розширення. Власне цей принцип і лежить в основі т.з. «пагін-архітектури» (Plugin Architecture).

5.Масштабованість процесу розробки. Можливість скоротити термін розробки за рахунок додавання до проекту нових людей. Архітектура повинна дозволяти розпаралелити процес розробки таким чином, щоб над програмою одночасно могли працювати необхідна кількість людей.

6.Тестованість. Код, який легше тестувати, буде містити менше помилок і буде надійніше працювати. Але тести не тільки покращують якість коду. Багато розробників приходять до висновку, що вимога «хорошої тестованості» є також і тією силою, яка дасть поштовх автоматично отримати в результаті хороший дизайн. Це в той самий час є і одним із важливих критеріїв, який дасть змогу оцінити якість коду: І навіть у випадку якщо не писати жодного рядка тестового коду, відповідь на це питання в 90% випадках дасть змогу зрозуміти, яка ситуація з «дизайном коду». Існує ціла методологія розробки програм на основі тестів, яка так і називаєть ся - Розробка через тестування (Test-Driven Development, TDD).

7.Можливість повторного використання. Систему бажано проектувати так, щоб її фрагменти можна було повторно використовувати в інших системах. При цьому код повинен бути добре структурований та зрозумілий

8. Супровідність. Над повноцінною програмою, як правило, працює безліч людей: одні йдуть, приходять нові. Часто,після написання супроводжувати програму теж, як правило, доводиться людям, які не приймали участь в її розробці. Тому хороша архітектура повинна давати

можливість відносно легко і швидко розібратися в системі новим людям. Сам е тому проект повинен бути добре структурованим.

В представленій роботі розроблено систему яка керується подіями. Іншими словами тут використано специфічний шаблон проектування, який дає змогу розширити загальний шаблон проектування більш пристосованими функціями. Канал подій створює централізований канал для подій. Цей централізований канал дозволяє об’єктам публікувати події або підписуватися на події. Механізм каналу подій значно розширює базовий шаблон в тому, що передплатник може отримувати опубліковані події від більш, ніж одного об'єкта, навіть якщо він зареєстрований тільки на одному каналі.

Шаблон «канал подій» (Event Channel) використовує об’єкти-представники для підписки на канал подій і об’єкт-представник для публікації подій в каналі. Використання представників дозволяє представникам існувати поза межами справжнього видавця або передплатника. Концептуальне уявлення каналу подій показано нижче:

Далі потрібно проаналізувати основні вимоги, а також вимоги до сутностей системи, яка керуватиметься подіями. Архітектура подібного типу розбивається на чотири логічних рівні (це оптимальні кількість шарів). Початок – це власне встановлення чи виявлення самого факту. Далі слідує технічне представлення (подання) його вже безпосередньо у формі події. На завершення потрібно представити не порожню множину реакцій, які будуть отримані на цю таку подію.

В загальному, керована подіями архітектура включає постачальників подій, які створюють потоки подій, і споживачів подій, які прослуховують ці події.

Концептуальне представлення архітектури - уявлення про канал подій показано графічно на рисунку 2.1.у вигляді рівнів.

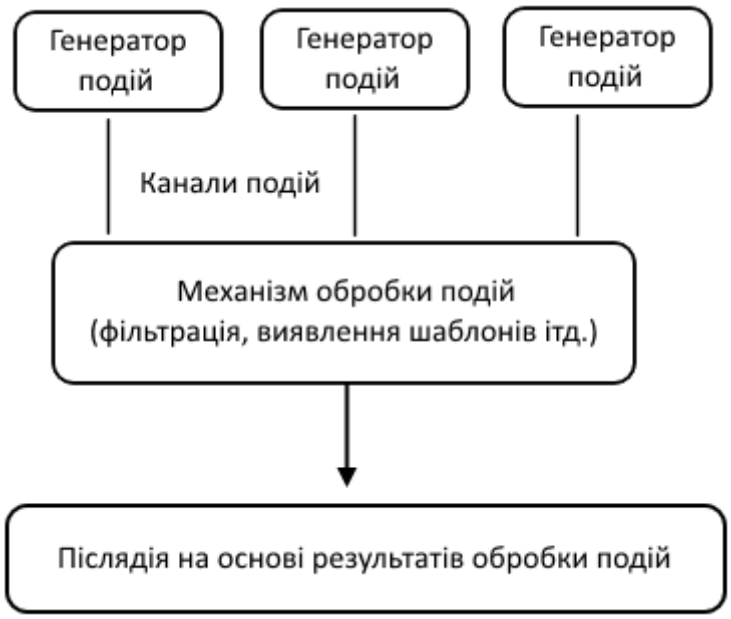

Рисунок 2.1. Концептуальне представлення архітектури

Події доставляються практично миттєво, що дозволяє споживачам негайно реагувати на події, що відбуваються. Постачальники не пов’язані зі споживачами – жоден постачальник не знає, хто прослуховує його події. Споживачі також не залежать один від одного, і кожен з них отримує всі події. Це важлива відмінність від шаблону конкуруючих клієнтів, в якому користувачі можуть отримувати повідомлення з черги, і кожне повідомлення обробляється тільки один раз (якщо не виникає помилок). У деяких системах, події обробляються в величезних обсягах.

В основі керованої подіями архітектури може лежати модель публікації і підписки або модель потоку подій.

Публікація і підписка – це інфраструктура обміну повідомленнями в якій підтримується список підписок. Кожна публікована подія надсилається кожному передплатнику. Отримане повідомлення можна відтворити повторно. Також воно не доставляється тим передплатникам, які додаються пізніше.

Передача потокового подій передбачає запис в журнал. Події в межах кожної секції строго впорядковані і зберігаються протягом тривалого часу. Клієнти не підписуються на потік, а просто зчитують події з будь-якої його

частини. Кожен клієнт самостійно управляє своїм положенням в потоці. Це означає, що клієнт може підключитися в будь-який час та (або) прослухати події повторно.

На стороні одержувача є кілька найпоширеніших варіантів реалізації. Сюди можна віднести обробку простих подій. Кожна подія негайно запускається дію в об'єкті-одержувачі. Також важливо згадати і про обробку складних подій, коли об'єкт-одержувач обробляє послідовність подій і відстежує в них певні закономірності за допомогою деякого технологічного рішення, наприклад Azure Stream Analytics або Apache Storm. Наприклад, можна виконувати статистичну обробку показань вбудованого пристрою за деякий період часу, щоб створювати повідомлення, коли ковзне середнє значення виходить за певні порогові значення. При обробці потоку подій платформу потокової передачі даних можна використовувати як конвеєр для прийому подій і передачі їх в обробники потоків. Обробники потоків певним чином реагують на ці процеси або перетворюють потік. Може існувати кілька обробників потоку для різних підсистем додатки.

Джерело подій може перебувати за межами системи. У такій ситуації система повинна підтримувати прийом даних в таких обсягах і на такій швидкості, які відповідають характеристикам джерела даних.

На представленій вище логічної схемою кожен тип споживача позначений окремим блоком. На практиці зазвичай використовується кілька примірників кожного постачальника, щоб вони не ставали єдиною точкою відмови. Наявність декількох екземплярів також може вимагатися для обробки подій в потрібних обсягах і (або) з потрібною швидкістю. Крім того, кожен об'єкт-одержувач може обробляти події в декількох потоках. Це може привести до проблем, якщо події повинні оброблятися по порядку або вимагати рівно одну семантику.

2.1.1 Представлення концептуальної моделі системи

Концептуальною моделлю зазвичай називають абстрактну модель, яка як правило визначатиме структуру системи, яка моделюється, з визначенням властивостей її елементів, причинно-наслідковими зв'язками, властивостями системи, тобто все те, що може суттєво впривати на досягнення поставленої цілі, якою є безпоседеньо моделювання [10].

Концептуальна модель представляє два різних погляду на системи обробки подій, призначені для опису важливих концепцій та їх взаємовідносин на абстрактному рівні з відокремленням від всіх технічних подробиць. Мережа обробки подій (Event Processing Network - EPN) абстрагує важливі функціональні можливості компонентів вводу, обробки і виведення даних з системи обробки подій. Керуючись концепціями EPN, наша концептуальна архітектура ідентифікує абстрактні архітектурні елементи, або компоненти, які можуть бути залучені в реалізацію системи обробки подій і відношення між ними для забезпечення цінності. Цей абстрактний рівень не залежить від технологій, протоколів і продуктів, які могли б використовуватися для реалізації компонентів.

Мета концептуальної архітектури – сформувати основу для реалізації систем обробки подій і керованих подіями додатків, а також надати загальну інтегровану середу для визначення, порівняння та протиставлення архітектур і реалізацій рішень для обробки подій.

Ми прагнемо того, щоб концептуальна архітектура надавала на концептуальному рівні достатній набір компонентів, з яких можна побудувати систему обробки подій, проте не встановлювала жодних вимог щодо реалізації компонентів, які не потрібні в конкретній системі, а також не накладала жодних вимог по відповідності.

У нашій високорівневій абстракції системи обробки подій застосуємо концепції структури мережі обробки поді, що зображена на рисунку 2.2. Як вже зазначалося, мережа обробки подій є концептуальним формулюванням,

яка описує структуру систем обробки подій і загальні функціональні можливості, які повинні підтримувати всі такі системи.

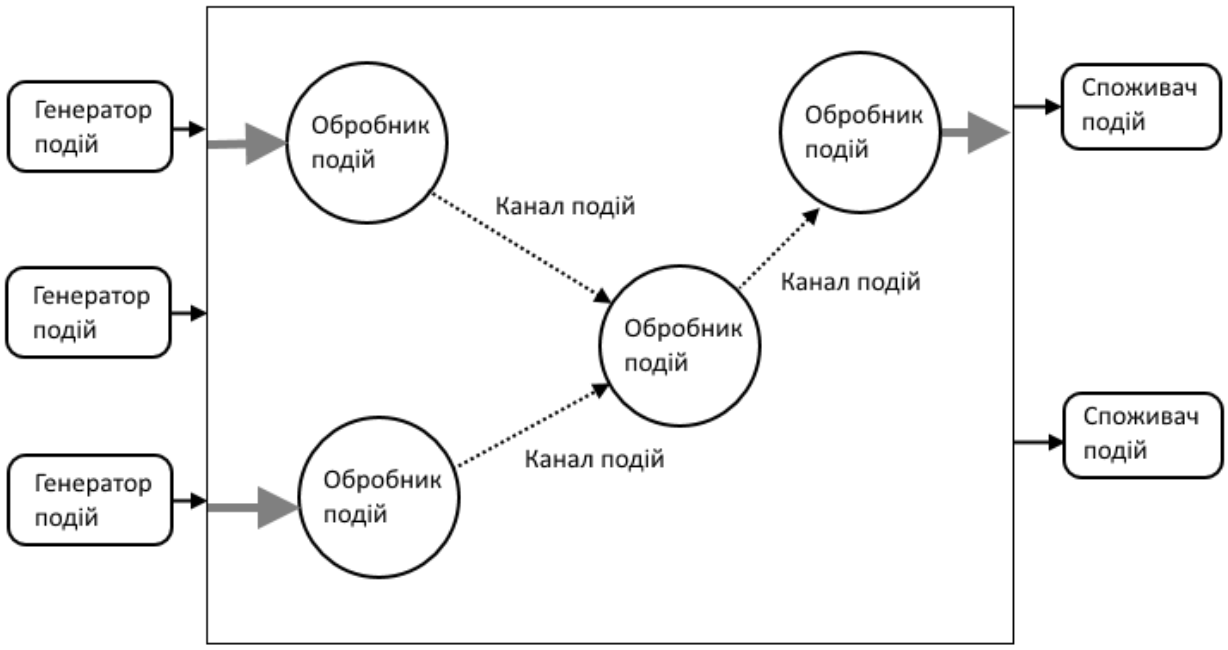

Рисунок 2.2 – Мережева концептуальна модель системи

EPN складається з чотирьох компонентів: генератор подій, споживач подій, обробник подій і компонент сполуки, канал подій (event channel). EPN описує як події, що надходять від генераторів направляються споживачам через обробники, що обробляють ці події, - наприклад, шляхом виконання перетворень, перевірки коректності або поповнення інформацією. Будь-яка подія, що приходить від одного компонента до іншого, повинно проходити по каналу. Канали подій з'єднують генератори подій з обробниками всередині EPN, різні EPA [11] між собою, якщо це необхідно, і EPA зі споживачами, що в результаті забезпечує проходження подій від генераторів подій через EPN до споживачів подій. Ця діаграма також демонструє, що різні події, що генеруються виробниками подій в EPN, можуть оброблятися відповідними групами EPA, з'єднаними за допомогою каналів, для

подальшого направлення цих подій (або породжених з них) різним споживачам подій в EPN.

2.1.2 Опис механізму доставки подій

Канал подій – це механізм доставки подій або потоків подій від генераторів подій і обробників до споживачів і іншим обробникам. На даному рівні абстракції не накладає ніяких обмежень ні на властивості каналу подій (наприклад, чи може канал передавати події кількох типів), ні на механізм передачі подій.

Канал подій може приймати кілька подій від різних генераторів, а також може передавати події, скомбіновані з декількох джерел, до декількох обробників і споживачам. Порядок надходження подій від декількох обробників і генераторів при створенні комбінованого набору подій залежить від реалізації і не регламентується концептуальною моделлю, а в деяких випадках ніякого упорядкування не потрібно. Проте канал подій відповідає за прийом подій від генератора і обробника, упорядкованих (при необхідності) і комбінованих, і їх надання відповідним споживачам подій і обробникам.

Ще одним обов'язком каналу подій може бути запам'ятовування хронології потоку подій для ретроспективної обробки подій у відповідності зі стратегією збереження, визначальною тривалість і умови фільтрації для всіх подій. Ретроспективна обробка подій – це виявлення характерних послідовностей подій по їх хронології, на відміну від оперативної обробки подій, при якій послідовності подій виявляються в міру виникнення нових подій.

2.1.3 Опис вузла-джерела і споживачі

Генератори передають події в канал подій для споживання усіма зацікавленими сторонами. Такий зацікавленою стороною може бути

споживач подій або обробник. Концептуальна модель не накладає жодних обмежень на механізм отримання подій від генератора подій чи обробника.

Усередині мережі вузлів і ребер генератор подій представляється у вигляді вузла-джерела, що має тільки вихідні ребра. Кількість вихідних ребер представляє кількість різних каналів подій, залучених до процесу передачі подій від генератора до обробника чи споживача, які приймають їх. Одна і та ж подія може публікуватися або бути доступною для декількох каналів, однак, виходячи з досвіду проектування, краще надати маршрутизацію компонентам для кращого управління, дизайну та розуміння загальних вимог обробки подій.

Споживач подій цікавиться подіями, необхідними йому для виконання його обов'язків. Після отримання такої події споживач подій виконує певні завдання асоційовані з цією подією.

Споживач подій представляється в моделі як точка поглинання, що має тільки вхідні ребра. Кількість вхідних ребер представляє кількість різних каналів подій, залучених до процесу передачі подій до даного споживача. Одна і та ж подія може прийматися з декількох каналів.

Не можна виключати, що споживач подій може бути також і генератором подій.

У розподілених і гетерогенних середовищах генератори подій можуть генерувати події не в тому вигляді, який очікує отримати споживач подій. Такі події можуть відрізнятися по синтаксису (структурі), по семантичному змістом або обом цим аспектам. Бувають ситуації, коли дія, що виконується споживачем активізується не однією подією, а складною комбінацією подій, що відбуваються в різний час і в різних контекстах. Обробники подій (event processing agent – EPA) [12] служать для виявлення шаблонів первинних подій, для обробки цих подій у вигляді поповнення інформацією, перетворення і перевірки коректності і для породження нових подій. EPA відповідає за генерування цих породжених подій і приймає рішення де і як ці події слід зробити доступними.

EPA може мати три стани:

1) фільтрація – стан відповідає за вибір подій для обробки згідно із зазначеним шаблоном;
2) обробка – даний стан, якщо воно передбачено, відповідає за застосування функцій обробки до обраних подій, що призводить до породження подій;
3) поширення – даний стан відповідає за поширення подій в каналі.

EPA приймає події з каналів для виявлення шаблонів чи іншої обробки, багато в чому аналогічно тому, як споживачі приймають події з каналів для виконання дій. EPA відправляє події по каналах подій під час стану поширення. Усередині мережі вузлів і ребер EPA представляється як вузол з ребрами, спрямованими до вузла і від нього. Кількість ребер, спрямованих до вузла, дорівнює кількості різних каналів подій, що використовуються обробником для виконання своїх функцій (наприклад, виявлення шаблонів). Кількість ребер, спрямованих з вузла, дорівнює кількості різних каналів подій, за якими обробником відправляє.

Таким чином, мережа обробки подій (EPN) – це спрямований граф операцій обробки подій, з'єднаних за допомогою каналів подій. EPA всередині мережі забезпечують сервіси обробки подій та посередницькі функції, тобто отримують набір з одного або декількох подій, виконують якусь обробку і повертають (можливо новий) набір подій. Основними завданнями мережі EPN є:

- прийом подій від виробників;
- передача їх групі обробників подій для обробки;
- доставка подій потрібним споживачам.

2.1.4 Представлення архітектури системи

Представлена концептуальна архітектура розвиває концепцію EPN шляхом визначення різних компонентів, які можуть бути залучені в реалізацію обробки подій. Деякі з цих компонентів є еквівалентами EPA або

набору з’єднаних EPA (на рівні EPN), тоді як інші залучені в потік подій і є еквівалентами каналам в EPN. На рисунку 2.3 показано, як моя концептуальна архітектура відображається на EPN.

Архітектура будь-якої системи, що підтримує обробку подій, повинна допускати гнучке визначення логіки обробки подій: виявлення шаблонів подій, породження нових подій, перетворення, маршрутизацію від генераторів до споживачів на основі необхідної бізнес-логіки. Це дає компанії можливість реагувати на зміни, виконувати відповідні процеси і впливати на триваючі процеси з урахуванням змін. Більше того, таке визначення обробки подій повинно легко змінюватися і швидко розгортатися відповідно до вимог (змінами в бізнес-процесах і стратегіях).

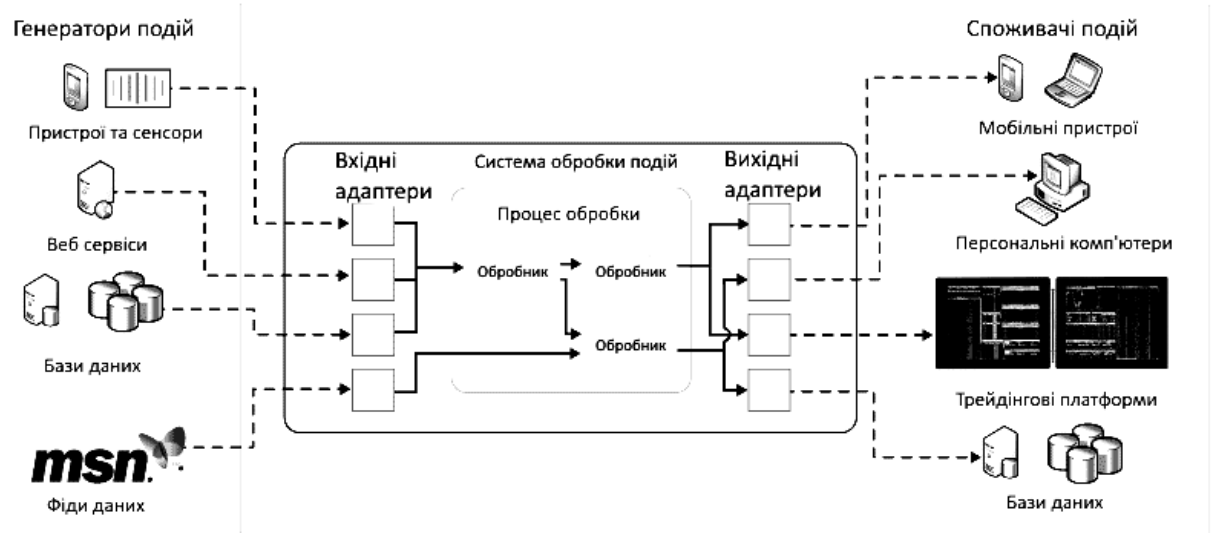

Рисунок 2.3 – Концептуальна архітектура системи

Ще однією базовою складовою є адаптери: вхідні і вихідні. Саме вони дозволяють пов'язувати SI з навколишнім світом і використовувати різні джерела даних.

Щоб зрозуміти, яку вигоду для бізнесу може дати система обробки подій, важливо розглянути більш дрібний рівень деталізації, ніж мережу обробки подій, розглянути компоненти, які можна використовувати для

створення системи обробки подій і їх взаємодії. Результатом цього процесу є те, що ми називаємо концептуальної архітектурою системи обробки подій. Крім трьох звичайних рівнів системи обробки подій (генератори подій та асоційовані компоненти, споживач подій та асоційовані компоненти, проміжний рівень шини подій - Event Bus) наша концептуальна архітектура передбачає включення компонентів системи захисту, моніторингу, аналітики та управління подіями і потоками подій.

На найпростішому рівні мінімальний набір концептуальних компонентів, необхідних для системи обробки подій, складається з рівня генератора подій (Event Emitter) для генерування подій від генераторів, шини подій (Event Bus) і рівня обробника подій (Event Handler).

Може виникнути потреба в додаткових компонентах крім генератора подій, шини подій і обробника (приймача) подій. На практиці згенеровані події не завжди можуть негайно вирушати споживачам. Так як генератор події не повинен знати про споживачів в системі, потрібно проміжний рівень між виробником і споживачем. Цей проміжний рівень виконує додаткові завдання пов'язані з подіями. Не всі події генеруються в потрібному форматі, і в подібних випадках вони повинні бути перетворені в потрібний формат до проміжного рівня. У деяких випадках звичайна подія може бути оцінений на значимість обробником подій, що призводить до генерування нового значущої події. Аналогічно не всі події, що приймаються на боці споживача, можуть бути придатні для використання. Тому може знадобитися деяка обробка на боці споживача. Споживач може ігнорувати деякі з прийнятих ним подій.

На рисунку 2.4 показані всі компоненти концептуальної архітектури обробки подій. Будь-яку реалізацію обробки подій можна описати за допомогою цього базового набору компонентів на концептуальному рівні, але не можна стверджувати, що в кожній реалізації обов'язково будуть потрібні всі компоненти. Аналогічно, не всі компоненти можуть бути необхідні в кожному конкретному сценарії. Як буде показано пізніше, коли

ми повернемося до розгляду сценаріїв і побачимо, як вони відображаються на концептуальну архітектуру, найчастіше у вирішенні беруть участь не всі компоненти.

Рисунок 2.4 – Концептуальна архітектура обробки подій

Потік подій у концептуальній архітектурі спрямований від генератора до споживача, і в цьому ж порядку розташовані компоненти, показані на схемі концептуальної архітектури.

Генератори подій не містить ні логіки управління подіями, ні будь-якої логіки прийняття рішень про те, що генерувати і коли. Типовими прикладами виробників подій є:

– датчики подій, які виявляють ситуації і генерують необроблені події або породжують події з потоків даних або бізнес-потоків;

– монітори та датчики, які генерують події про проблеми в системі ;
– сервіси і додатки, які генерують події в ключових точках обробки даних;
– кінцеві автомати, які генерують події при зміні стану.

Шина подій (Event Bus) – приймає події від генераторів подій, потенційно в дуже великих обсягах і активізує споживачів за допомогою обробників подій. Серед можливостей шини подій можна відзначити обробку з метою породження з вхідних подій більш інформативних подій меншого обсягу.

Обробник подій (Event Handler) – підготовлює події з шини подій для споживання споживачем подій, приймаючи події і вирішуючи, як реагувати на них. Обробник подій може також надавати сервіси простої обробки подій, які займаються обробкою на боці споживача з метою фільтрації та проміжного зберігання подій, прийнятих з шини подій. Обробники подій можуть також визначати належних споживачів для реагування на подію і активізувати їх з контекстом, породженим з події.

Споживач подій (Event Consumer) – виконує завдання, реагуючи на події. Споживач подій має обмежені знання про джерело події і просто знає про те, що він активізований в результаті події з наданням інформація про контекст цієї події. Прикладами типових споживачів подій є:

– виконавчі елементи (Event Actuators), які активуються для виконання таких фізичних завдань, як приведення в дію клапанів управління, перемикачів або сигналів тривоги;
– інформаційні панелі оператора, що відображають інформацію про поведінку інформаційних систем і відповідних сервісів;
– інформаційні панелі підприємства, які відображають інформацію про поведінку бізнес-процесів;
– бізнес-процеси, які можуть бути активізовані або продовжені у відповідь на подію;

– сервіси і додатки, які можуть бути активізовані у відповідь на подію і можуть включати в себе зовнішні системи ;
– кінцеві автомати, стан яких може бути змінене у відповідь на подію.

Цей огляд концептуальної архітектури заснований на ролях компонентів, але це не означає, що конкретний учасник архітектури не може виконувати більше однієї ролі.

Концептуальна архітектура може розглядатися як "вкладена", так як будь-який її учасник може містити в собі мережу з інших компонентів. Наприклад, виробник подій може генерувати подію в головну шину подій, але в процесі генерування цієї події може використовуватися міні-версія цієї ж загальної моделі, в якій генерує вихідну просту подію для зіставлення за допомогою шаблону з іншими подіями в міні-шині подій.

2.2 Концептуальна модель в дії

В даному підрозділі описано концептуальну модель в дії. У роботі моделі можна виділити три фази:

– фаза генерування події;
– фаза обробки події;
– фаза споживання події.

Ця схема обробки не є єдино можливою і між цими трьома фазами існує дуже обмежений взаємозв'язок, особливо за наявності складної обробки подій. Тому ці три фази можуть виникати абсолютно в різні періоди часу, і між згенерованими подією і спожитим подією може бути тільки причинно-наслідковий взаємозв'язок.

2.2.1 Фаза генерування подій

Компонентами концептуальної архітектури, залученими в цю фазу є головним чином компоненти генератора подій. Їх роль у генеруванні подій

може бути абсолютно технічною, оскільки вони відповідають в основному за початкову ініціалізацію простої події.

При виникненні події генератор відправляє повідомлення в шину подій, використовуючи свій логічний рівень. Це повідомлення потім відправляється в "Адаптер подій", який відповідальний за узгодження повідомлення про подію з транспортним протоколом, підтримуваним шиною подій.

У більш складних реалізаціях рівень "Генератор подій" може підтримувати більш складні схеми виявлення подій на стороні виробника і реалізовувати локальну обробку подій.

2.2.2 Фаза обробки подій

На локальному підрівні "Обробка подій" можуть бути реалізовані: фільтрація, об'єднання, поповнення інформацією і навіть перевірка елементарних подій, щоб до споживача надходило єдине повідомлення, характеризує виникнення складної бізнес-події.

Можливі також ситуації, коли одна вхідна подія може генерувати кілька вихідних подій у різних форматах, що залежать від їх корисного навантаження і представлених для них запитів підписки. Споживачі подій можуть оголосити через сервіси підписки про свою зацікавленість у подіях конкретного типу. Вони можуть також встановлювати додаткові параметри для вказівки способу повідомлення про відповідне подію.

При надходженні повідомлення про подію підрівень "Сервіс публікації" обробляє кожен окремий запит підписки на даний тип подій. Виконуються необхідні сервіси обробки подій для проміжної обробки (головним чином сервіси фільтрації, поповнення інформацією і перетворення). Потім сервіси повідомлення відправляють отримане в результаті обробки повідомлення про подію споживачеві подій далі через канал подій.

Важливу роль в обробці має реєстр подій, особливо при складній обробці. Він відстежує події, оброблені в потоках подій, вхідні події, або

події отримані в результаті обробки. По-перше, це дуже корисно для завдань моніторингу (наприклад, для відповіді на питання, яка комбінація вхідних подій сформувала даний результат). По-друге, оскільки набір подій може охоплювати великий інтервал часу, може знадобитися зберігати пов'язані з ним події.

2.2.3 Фаза споживання подій

На цій фазі споживач подій приймає повідомлення про подію з шини подій і обробляє його, покладаючись на свій локальний рівень абстракції та власні потреби. Підрівень "Адаптер подій" відповідає за отримання повідомлення про подію та узгодження з транспортними протоколами, підтримуваними шиною подій. Тут може виконуватися попередня обробка повідомлення про подію, реалізована на локальному підрівні "Обробка подій". Такою обробкою може бути, наприклад, технічна адаптація корисного навантаження повідомлення з метою узгодження з конкретним вхідним форматом споживача подій.

2.3 Реалізація архітектури системи

Спроектовано концептуальну модель обробки подій, що складається з концептуальної архітектури, заснованої на концепціях мережі обробки подій. Ця концептуальна архітектура демонструє, як можна підготувати і обробляти події, що генеруються, для використання споживачами подій із застосуванням проміжної шини подій, що надає сервіси, які дозволяють фільтрувати, поповнювати інформацією, форматувати, об'єднувати або розділяти події в міру необхідності.

Ця архітектура включає можливості, які можуть знадобитися на стороні виробника або споживача, наприклад, можливості деякої простої обробки подій і адаптери подій. Призначення цієї концептуальної архітектури полягає в ідентифікації всіх компонентів, які можуть

знадобитися при реалізації будь-якої системи обробки подій, але для конкретної реалізації або сценарію швидше за все будуть потрібні тільки обрані компоненти.

На основі спроектованої архітектури можна реалізувати базовий функціонал обробки та маршрутизації подій не залежний від предметної області та описати базові сутності системи.

2.3.1 Сутності системи

Проаналізувавши вимоги до системи та концептуальну архітектуру можна виділити базові сутності які не будуть залежати від імплементації в конкретній предметній області, це:

- подія, базовий клас, що описує настання події на будь-якому рівні системи;
- вхідний адаптер – приводить дані до потрібного формату, що приходять від генераторів подій;
- вихідний адаптер – приводить дані до потрібного формату, що надсилаються споживачам подій;
- реєстратор подій – відповідає за збереження інформації про настання події на будь якому рівні системи;
- ядро системи – сутність, що відповідає за маршрутизацію подій та зв’язує всі інші сутності системи;
- базові налаштування – надає можливість налаштування базових параметрів системи.

На UML діаграмі 2.5 та 2.6 показано вищеописані типи та зв’язки між ними.

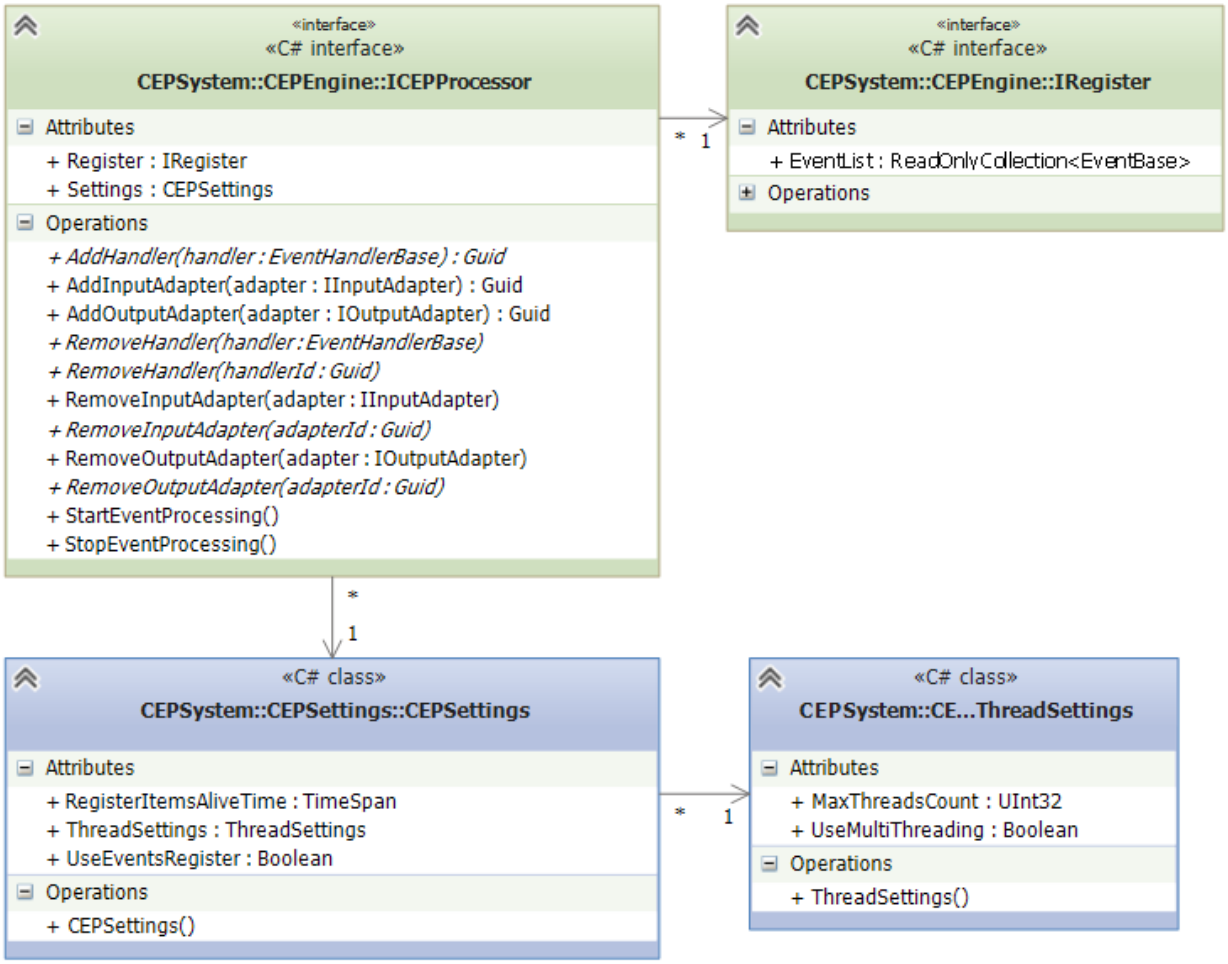

Рисунок 2.5 – Залежності між сутностями системи

На рисунку 2.5 видно, що `ICEPProcessor` – ядро системи. `ICEPProcessor` агрегує сутність `IRegister` – реєстратор подій, і надає доступ до нього за допомогою публічного поля та містить налаштування.

`CEPSetting` – містить базові налаштування системи, та дозволяє змінювати їх за допомогою публічних полів. Цей клас дозволяє налаштувати багатопоточність та період часу протягом якого реєстр подій буде зберігати дані про події.

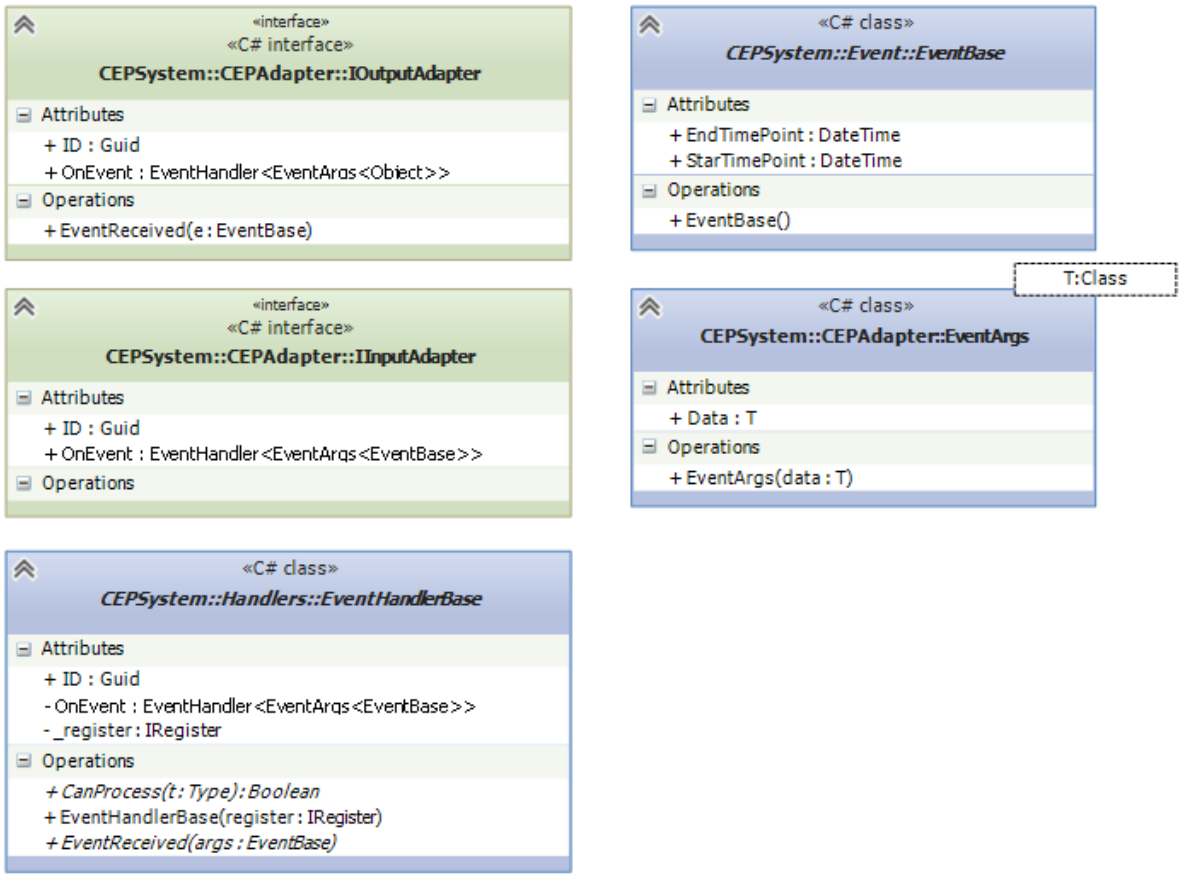

Рисунок 2.6 – Структура базових типів

Адаптери є одними з основних типів системи, що дозволять нам уніфікувати типи вхідних та вихідних потоків.

Вхідний адаптер повинен вміти приводити типи даних, що надає генератор подій до базового типу `EventBase` події системи. Тобто вхідний адаптер буде володіти інформацією про генератора подій та реагувати на прості події що приходять з нього. Вхідний адаптер також може проводити просту обробку події, наприклад:

– фільтрацію;

– доповнення додатковими параметрами;

– базові обчислення на основі простої події.

Вихідний адаптер повинен привести базовий тип події, що використовується в системі до типу, що потребує споживач.

Слід також зазначити що адаптери можуть провести просту обробку події і не мають доступу до реєстру подій.

Таким чином система отримує повністю незалежну від джерела даних та кінцевого споживача подій логіку роботи шини комплексної обробки подій, це графічно зображено на рисунку 2.7.

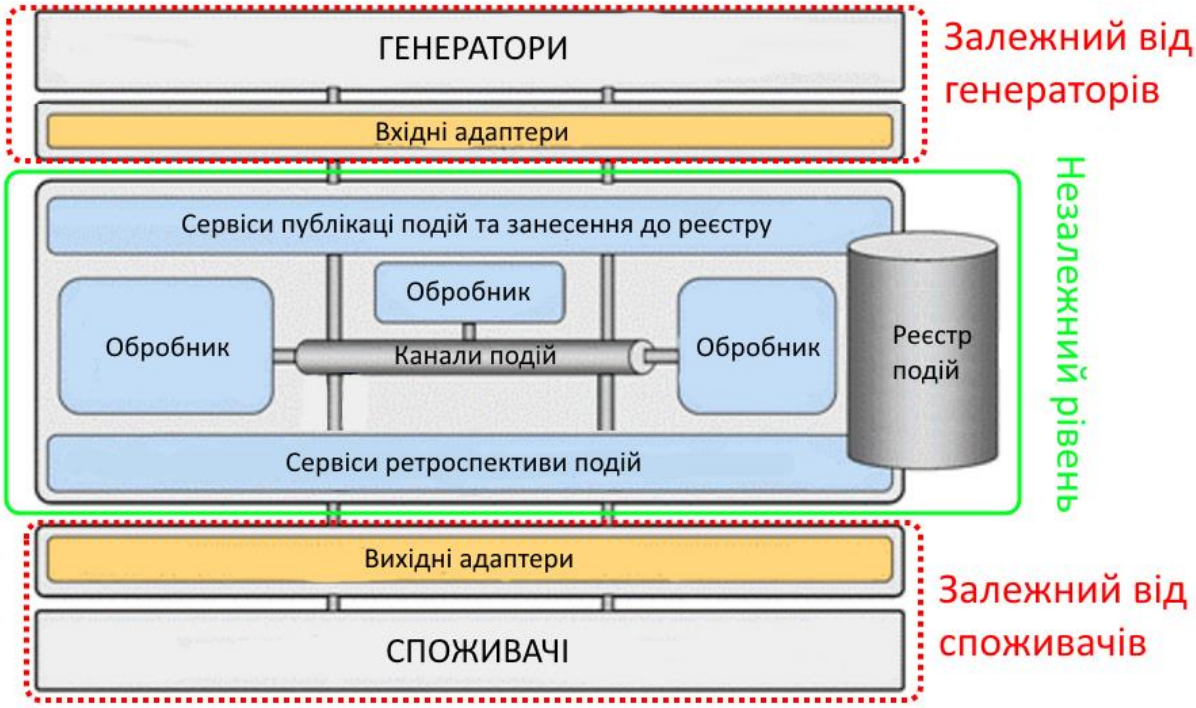

Рисунок 2.7 – Структура системи

2.3.2 Суть принципів використання системи

Принцип використання системи передбачає імплементацію вхідних та вихідних адаптерів для уніфікації даних, що приходять з різних джерел. Для цього потрібно використовувати інтерфейси адаптерів передбачені системою.

Кожний конкретний адаптер повинен вміти приводити типи даних з вхідного потоку та конвертувати їх до базового типу події з можливістю розширення. Тобто адаптер повинен реалізовуватися кінцевим користувачем системи для конкретних джерел даних.

Базовий тип подій спроектований з розрахунком на його розширення для потреб конкретної реалізації та розширенням відповідно до потреб. Основними полями події ї час її початку та тривалість. В системі можуть використовуватися 3 типи подій залежно від їх тривалості та того процесу який вони описують, це:

- інтервальні події;
- точкові;
- граничні.

Інтервальна подія – представляє подію з фіксованою тривалістю, дані якого дійсні лише протягом певного періоду часу. Час початку та кінця події визначений і повинні бути встановлені в поля `StartTimePoint` та `EndTimePoint` в екземплярі `EventBase`;

Точкова – подія, що відбувається в конкретний момент часу. Необхідно тільки початковий час події. Час початку та кінця події визначений і одинаковий, також повинні бути встановлені в поля `StartTimePoint` та `EndTimePoint` в екземплярі `EventBase`;

Гранична – подія, що має тільки початкову часову точку, тривалість якої не відома. На момент прибуття відомо тільки початковий час, тому кінцевий час встановлюється на максимально віддалений час в майбутньому. Фактичний кінцеве час події стає відомо пізніше.

Тип цієї події повинен визначатися самим адаптером, а подія повинна рахуватися актуальною для обробки в даному часовому вікні.

Після визначення і реалізації потрібних адаптерів їх потрібно дадати в систему, для цього є 2 методи `AddInputAdapter` та `AddOutputAdapter` що на вхід приймають об’єкт адаптера та повертають його ідентифікатор. Ці методи реєструють адаптери в системі та підписуються на події які будуть ініційовуватися ними.

Щоб видалити адаптери з системи можна скористатися одним з 2ох методів `RemoveInputAdapter` та `RemoveOutputAdapter` які можуть приймати об’єкт адаптера чи його ідентифікатор для пошуку та видалення. Після

видалення адаптера генератор подій більше не буде ініційовувати події в системи через нього.

Базовим полем вхідного адаптера є подія на яку при додаванні адаптера буде підписуватися система. Параметром цієї події є базовий клас `EventBase`. Тобто вхідний адаптер моніторить генератора подій, уніфікує дані та проводить просту обробку подій (при потребі) та ініціює подію в системі з уже уніфікованими даними які відповідають протоколу в каналі подій.

Один генератор подій може мати різноманітні адаптери які можуть обробляти різні типи подій або один тип подій по-різному. Схема взаємодії вхідного адаптера з генератором подій та системою зображено на рисунку 2.8.

Адаптери реалізовується ззовні системи – тому він може мати доступ до будь яких зовнішніх ресурсів таких як база диних, файли, інші об’єкти. В процесі роботи системи внутрішня їхня реалізація може мінятися відповідно до зміни протоколу передачі даних від генератора.

Уніфікація даних дозволить нам не змінювати саму систему комплексної обробки, що в свою чергу зекономить нам час на відлагоджування та тестування і зменшить можливість виникнення помилок.

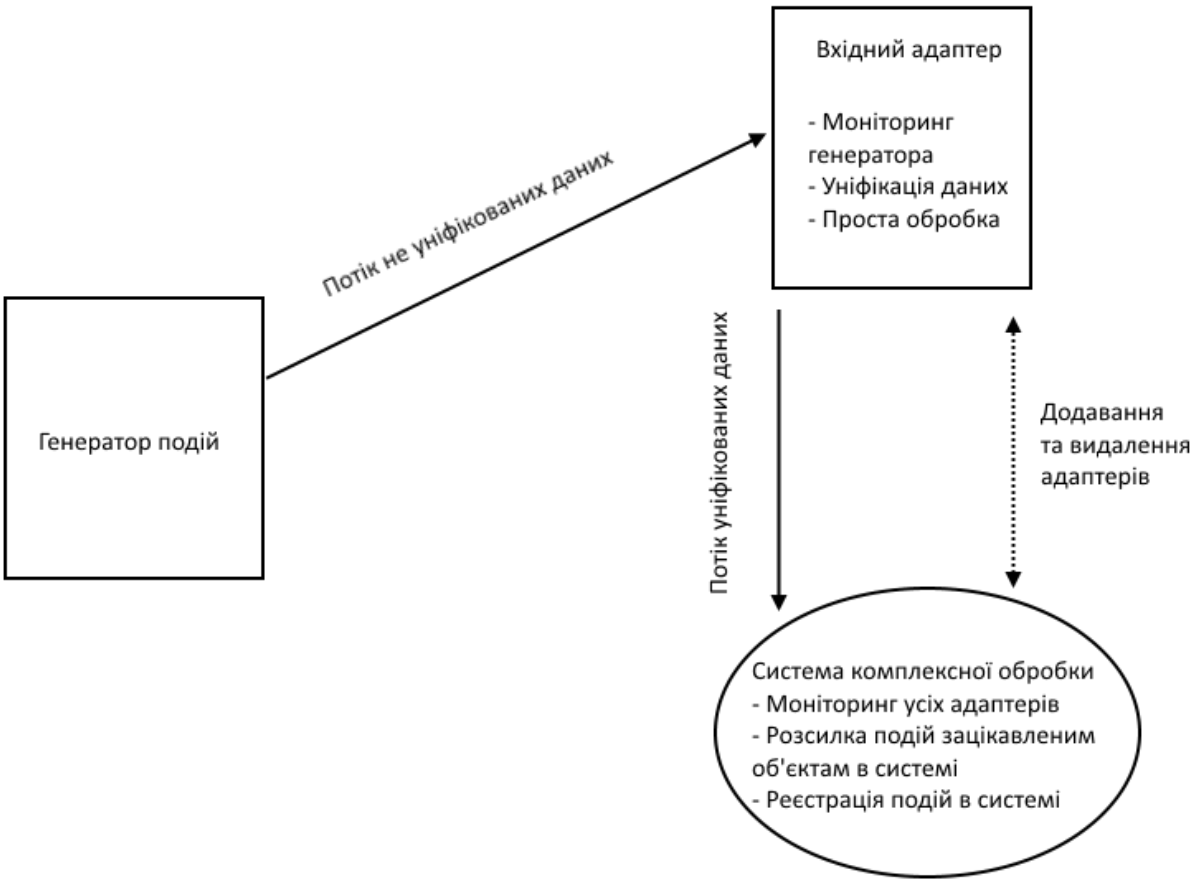

Рисунок 2.8 – Схема взаємодії вхідного адаптера
з генератором подій та системою

Вихідні адаптери схожі за своєю структурою та функціональністю з вхідними, але призначені для приведення даних до формату який потрібний споживачам подій. Враховуючи те що ми не знаємо хто є споживачем і які типи даних потрібні йому, вихідний адаптер повинен запаковувати дані в базовий тип (базовий тип буде залежати від платформи реалізації системи). Такий адаптер теж реалізовується кінцевим користувачем, залежно від предметної області і повинен володіти інформацією про кінцевого споживача подій. Механізм додавання та видалення його в системі ідентичний до вхідних адаптерів. Окрім поля-події, що буде представляти потік вихідних даних тут повинен бути імплементований ще й метод який буде викликатися системою для передачі подій від обробників. Залежно від типу події що прийшла та її параметрів вихідний адаптер може проігнорувати її або ж

згенерувати подію у вихідний потік для споживача. Схема взаємодії вихідного адаптера з системою та споживачем зображено на рисунку 2.9.

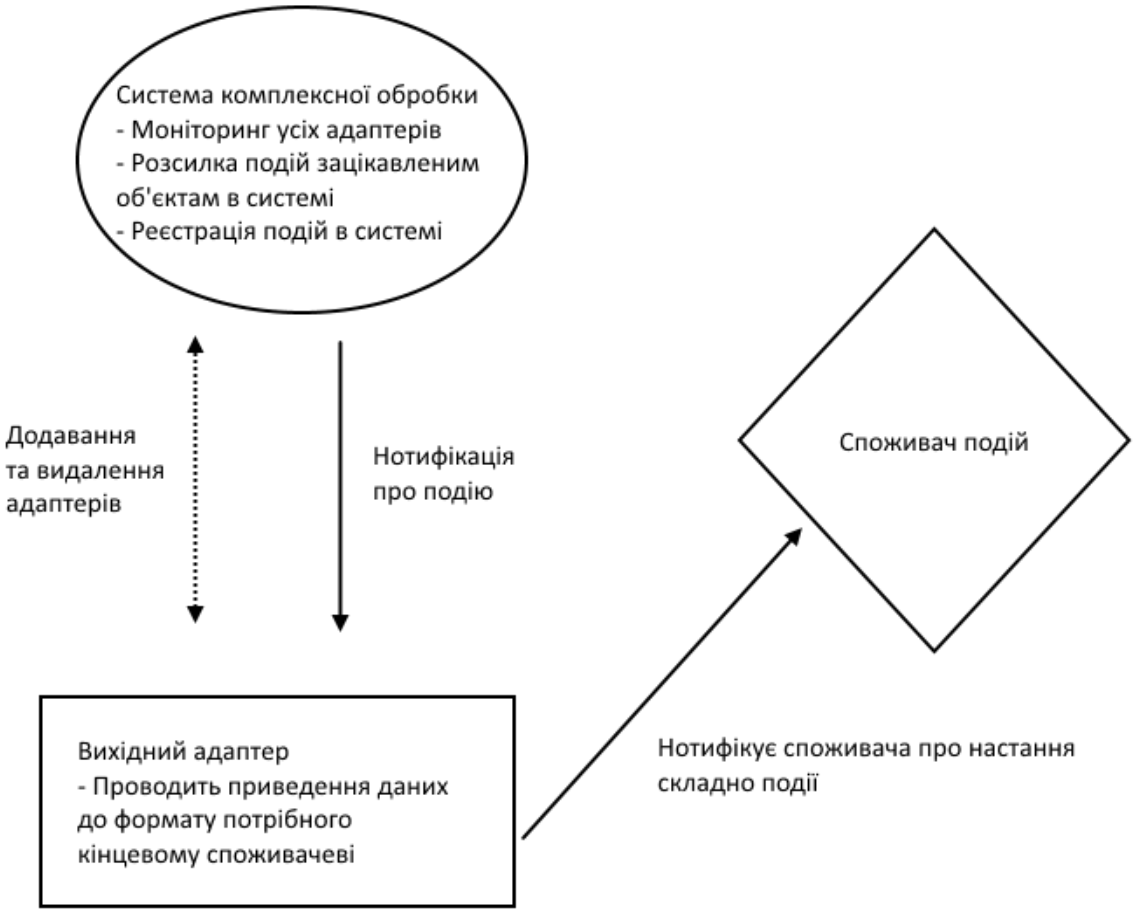

Рисунок 2.9 – Схема взаємодії вихідного адаптера з системою та споживачем

Також в системі надані інтерфейси для реалізації фільтрів. У фільтрів є декілька відмінностей від адаптерів в системі – це:

- уніфікованість даних;
- доступ до реєстру подій;
- можливість направляти події назад в систему для подальшої їх обробки, при цьому доповнивши її чи навіть змінивши тип;
- маючи доступ до реєстру визначати шаблони подій.

Основною функцією обробників є виявлення шаблонів подій, їхня агрегація та доповнення даними. Реалізація цієї функціональності не залежить ні від генераторів подій (джерел даних) ні від кінцевих споживачів. Дані що надходять до обробника і відправляються ним є уніфікованими чого і потрібно було добитися при проектуванні даної системи.

Додавати фільтри в систему можна за допомогою метода `AddHandler`, що приймає на вхід об'єкт обробника і повертає його ідентифікатор. Цей метод також як і `AddInputAdapter/AddOutputAdapter` проводить реєстрацію обробника в системі для подальшої розсилки подій. Базові матоди та поля обробника показано на рисунку 2.10.

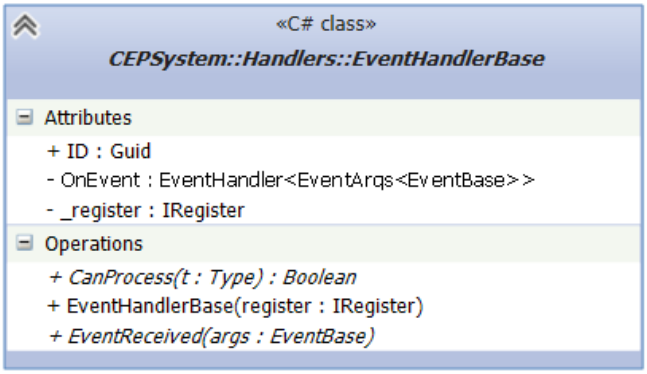

Рисунок 2.10 – UML діаграма обробника подій

Отже користувач зможе незалежно від джерел даних реалізовувати логічний рівень обробки подій та в подальшому заміняти чи додавати нові генератори поді та споживачів.

2.4 Аналіз якості спроектованої системи

Під якісним програмним забезпеченням розуміється програмне забезпечення, яке не містить помилок або дефектів, доставлене вчасно і в

рамках встановленого бюджету, відповідає вимогам і / або очікуванням і є придатним для корегування та внесення правок.

В контексті розробки програмного забезпечення його якість має два аспекти: функціональний (відображає, наскільки добре воно відповідає заданому дизайну, на підставі функціональних вимог або специфікацій) та структурний (стосується обробки не функціональних вимог, які підтримують виконання функціональних вимог, таких як надійність або ремонто-придатність, і ступеня, в якому програмне забезпечення було створено правильно).

Software Quality Assurance - Software Quality Assurance (SQA) - це комплекс заходів, спрямованих на забезпечення якості процесів розробки програмного забезпечення, які в кінцевому підсумку призводять до отримання якісного програмного продукту. Діяльність визначає і оцінює процеси створення програми, та включає процесно-орієнтовані дії.

В цілому, контроль якості програмного забезпечення (SQC) - це комплекс заходів щодо забезпечення якості програмних продуктів. Ці дії спрямовані на виявлення дефектів у фактичній продукції та включає в себе дії, орієнтовані на продукт.

Таким чином підсумовуючи ,буду вважати свій продукт якісним, якщо він буде:

1. коректним;
2. надійним;
3. практичним;
4. безпечним.

Технічні вимоги які ставляться до ПЗ при тестуванні описані відповідним стандартом ISO 9126. А вже вміст і зміст супровідної документації безпосередньо процесу тестування як правило визначено та описано в стандарті IEEE 829–1998 Standard for Software Test Documentation. Хоча є велика кількість підходів до проведення процесу тестування програмного продукту, однак ефективним тестуванням складних продуктів є

за своєю суттю достатньо творчий процес, з елементами пошуку та дослідництва. Навіть більше, можна сказати, що це не лише можливість створити та виконати достатньо «нудну» та рутинну процедуру.

Тестуванням можна назвати одну з технік контролю за якостю. На сьогодні на ринку присутня достатня кількість найрізноманітніших видів тестування, серед них ,як основні, можна назвати:

1. інсталяційний принцип тестування;
2. тестування за принципом сумісності;
3. так зване димове тестування;
4. тестування за модульним принципом;
5. тестування за регресивним принципом;
6. тестування за функціональним принципом;
7. тестування за деструктивним принципом;
8. тестування за швидкодією;
9. тестування на зручність використання;
10. так зване тестування на навантаження.

Я проаналізую лише його функціональну якість оскільки спроектована системи є бібліотекою класів без графічної частини та інсталятора і саме цей критерій для нас найбільш важливий. Для цього застосуємо інтегрований в Visual Studio інструмент, що має назву «Code Metrics». Результат аналізу показаний на рисунку 2.11.

Code Metrics Results

Filter: None Min:

Hierarchy	Maintainability I...	Cyclomatic Com...	Depth of Inherita...	Class Coupling
CEPSystem (Debug)	97	46	2	15
CEPSystem.CEPAdapter	98	10	2	5
CEPSystem.CEPEngine	100	14	0	8
CEPSystem.CEPSettings	94	12	1	2
CEPSystem.Event	94	5	1	1
CEPSystem.Handlers	94	5	1	4

Рисунок 2.11 – Метрика системи

Перша колонка містить назви проектів по яких проводився аналіз. Для детальнішого перегляду проекти можна розгортати.

Друга колонка – це Maintainability Index, комплексний показник якості коду. Цей показник розроблений фахівцями з Carnegie Mellon Software Engineering Institute. ЇЇ значення є найважливішими, діапазон може коливатися від 0 до 100, це так би мовити підсумкова оцінка аналізу. Чим вище значення в цій колонці там легше підтримувати код. Розраховується метрика за такою формулою [9]:

$$MI = MAX(0, (171 — 5.2 * \ln(HV) — 0.23 * CC — 16.2 * \ln(LoC)) * 100 / 171) \quad (1.1)$$

де, HV – Halstead Volume, обчислювальна складність. Чим більше операторів, тим більше значення цієї метрики, CC – Cyclomatic Complexity. Ця метрика описана нижче, LoC – кількість рядків коду.

Третя колонка – це Cyclomatic Complexity, показує структурну складність коду, тобто кількість різних гілок в коді. Чим більше цей показник, тим більше тестів має бути написано, для повного покриття коду.

Четверта колонка – це Depth of Inheritance, глибина наслідування. Ця метрика показує для кожного класу, який він по рахунку в ланцюжку наслідувань. Наприклад, є 3 класу A, B, C, B успадкований від A, а C успадкований від B, то значення цієї метрики для класів A, B і C буде дорівнює відповідно 1, 2 і 3.

Четверта колонка – це Class Coupling, показує ступінь залежності класів один від одного. Хороший дизайн програмного забезпечення передбачає невелику кількість пов'язаних класів. Чим їх більше, тим складніше повторно використовувати цей клас, а також підтримувати, тому, що існує дуже багато залежностей.

На основі отриманих результатів метрики можна зробити висновок про якість коду. Залежно від значень Maintainability Index код умовно розділяють на:

1 - «неякісний», Maintainability Index менший 10;

2 - «середньої якості», Maintainability Index від 10 до 20;

3 - «якісний», Maintainability Index більше 20.

В загальному, як висновок, варта зазначити що мій продукт, може містити незначні дефекти, проте вони ніяким чином не повинні впливати на його функціональність та коректність роботи.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Опис алгоритму

Алгоритмічна торгівля – формалізований процес здійснення торгових операцій на фінансових ринках за заданим алгоритмом з використанням спеціалізованих комп'ютерних систем (торгових роботів).

Завдання будь-якого біржового робота базується на математичній моделі поведінки фондового ринку. Алгоритм зіставляє реальні параметри і визначає найкращі для проведення транзакцій моменти.

Торгова система буде базуватися на певному алгоритмі визначення тренду ринку і залежно від його напрямку здійснювати торгівельні операції. Вона буде реалізована на платформі .Net 4.0 та мові програмування C#.

За основу системи візьмемо технічний індикатор Dynamic Trio, він визначає тренд ринку та дозволяє на основі 2-ох серій прогнозувати його рух.

Графічно цей індикатор представлено на рисунку 3.1. Його калькуляція на мові C# приведена в додатку Г. На вхід цей він приймає історичні дані ринку певного часового періоду та калькулює результат у вигляді масиву чисел. Якщо останнє значення індикатора перетинає ціну ринку знизу в верх – це означає що тренд ринку йде на спад і навпаки, якщо зверху в низ то він зростає.

Отже коли ринок починає спадати на вигідно продавати для того щоб в майбутньому купувати за нижчою ціною і отримувати прибуток і навпаки, якщо ринок росте то нам вигідно купувати для того що потім продати за вищою ціною.

Звідси зміна тренду індикатора Dynamic Trio і є тим моментом коли нам потрібно купувати чи продавати.

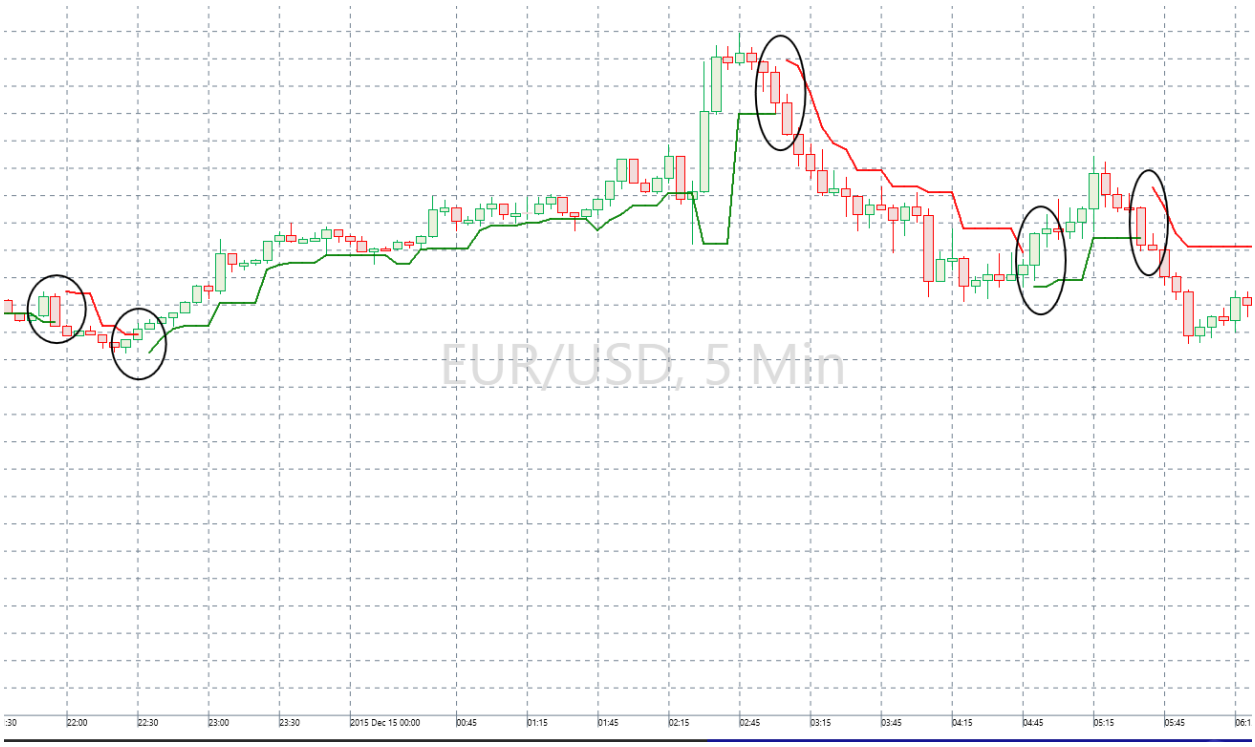

Рисунок 3.1 – Dynamic Trio на графіку цін

На рисунку 3.1 виділені ті місця де потрібно здійснювати торгівельні операції.

Якщо ми щось купили чи продали, щоб отримати прибуток нам потрібно зробити протилежну операцію з вигідною для нас різницею цін, іншими словами закрити позицію. Для цього нам потрібно визначити який прибуток з однієї торгівельної операції ми хочемо отримати для того, щоб закрити позицію в потрібний момент та максимальний можливий програш який ми можемо допустити. Прибуток по позиції визначається як різниця в поточній ціні та ціні відкриття позиція помножено на кількість того що було куплено/продано.

Отже, щоб реалізувати описаний алгоритм нам потрібно отримувати історичні дані цін та ціни в режимі реального часу з деякого джерела даних (фіда), та здійснювати торгівельні операції через брокера. Момент здійснення торгівельної операції ми будемо вираховувати за допомогою технічного індикатора Dynamic Trio. Перед здійсненням наступної операції нам потрібно

закривати попередній трейд, момент закриття ми будемо теж вираховувати з бажаного нами прибутку.

3.2 Визначення компонентів системи

Проаналізувавши алгоритм можна виділити генератора подій – це сервіс який надає історичні та поточні ціни, споживач подій – брокер, який буде проводити торгівельні операції, та 2 обробника подій – технічний індикатор який буде визначати момент здійснення операції та ще один обробник, що буде слідкувати за прибутком.

Для реалізації даної схеми нам потрібно реалізувати вхідний та вихідний адаптери які б конвертували дані від фіда залежно від формату даних який він використовує та відсилали дані брокеру у потрібному йому форматі.

Основними параметрами поточної ціни які нам потрібно для системи є ціна купівлі (Ask price) та ціна продажу (Bid price), а також час. Цей клас поточної ціни показано на рисунку 3.2.

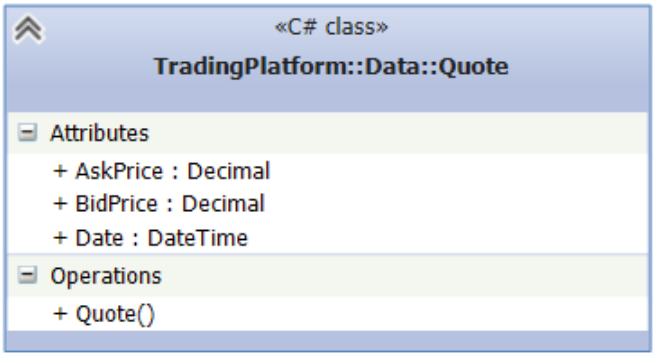

Рисунок 3.2 – UML діаграма класу поточної ціни

Інша важлива сутність це торгова операція (трейд), вона має набагато більше полів. Оскільки трейд повинен нести інформацію про тип операції (купівля/продаж), час її здійснення, ціна відкриття, поточна ціна, прибуток, комісія брокера та іншу службову інформацію. На рисунку 3.3 показано UML

діаграму спроектованого класу трейда. В додатку Д приведено лістинг коду реалізації цього класу.

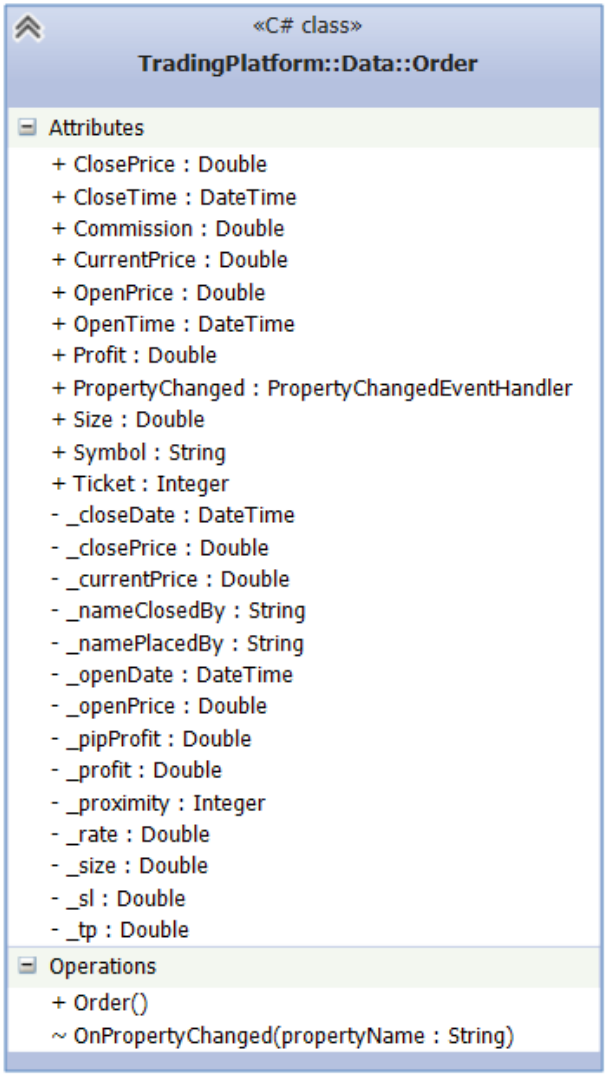

Рисунок 3.3 – UML діаграма класу торгівельної операції

Далі нам потрібно визначитися з протоколами даних від генератора (тобто фіда). Генератором в системі буде відомий сервіс FOREX. Він надає доступ до поточних цін за допомогою сокетного з’єднання та присилає дані в вигляді стрічки наступного формату:

```
[валютна пара],[ціна продажу],[ціна купівлі],[час]
```

Приклад отриманих даних наведено в лістингу 3.1. Реалізація з’єднання з сервером та отримання даних приведено в лістингу в додатку Е.

Лістинг 3.1 – Отримані дані з сервера у вигляді стрічки

```
USD/CAD,1.37346,1.37569,12/15/2015 07:15:15 PM;
CAD/JPY,88.615,88.712,12/15/2015 07:15:15 PM;
USOIL,37.37,37.59,12/15/2015 07:15:14 PM;
USD/CAD,1.37348,1.37392,12/15/2015 07:15:15 PM;
EUR/CAD,1.49911,1.49993,12/15/2015 07:15:15 PM;
```

Отже вищеописаний протокол вхідних даних від фіда буде потоком який потрібно обробляти і направляти у вхідний адаптер.

Споживачем даних буде брокер Metatrader, нижче на рисунку 3.4 показано інтерфейс цього брокера. Основною функцією яка нас цікавить є OrderSend, що приймає 10 параметрів різного роду типів що описують трейд. Це в основному стрічки, числа та тип, що описує операцію `TradingAPI.MT4Server.Op` з наступними можливими значеннями:

- Balance;
- Buy;
- BuyLimit;
- BuyStop;
- Credit;
- Sell;
- SellLimit;
- SellStop.

З цих операцій нам потрібно використати тільки Buy та Sell – це операції купівлі та продажу відповідно.

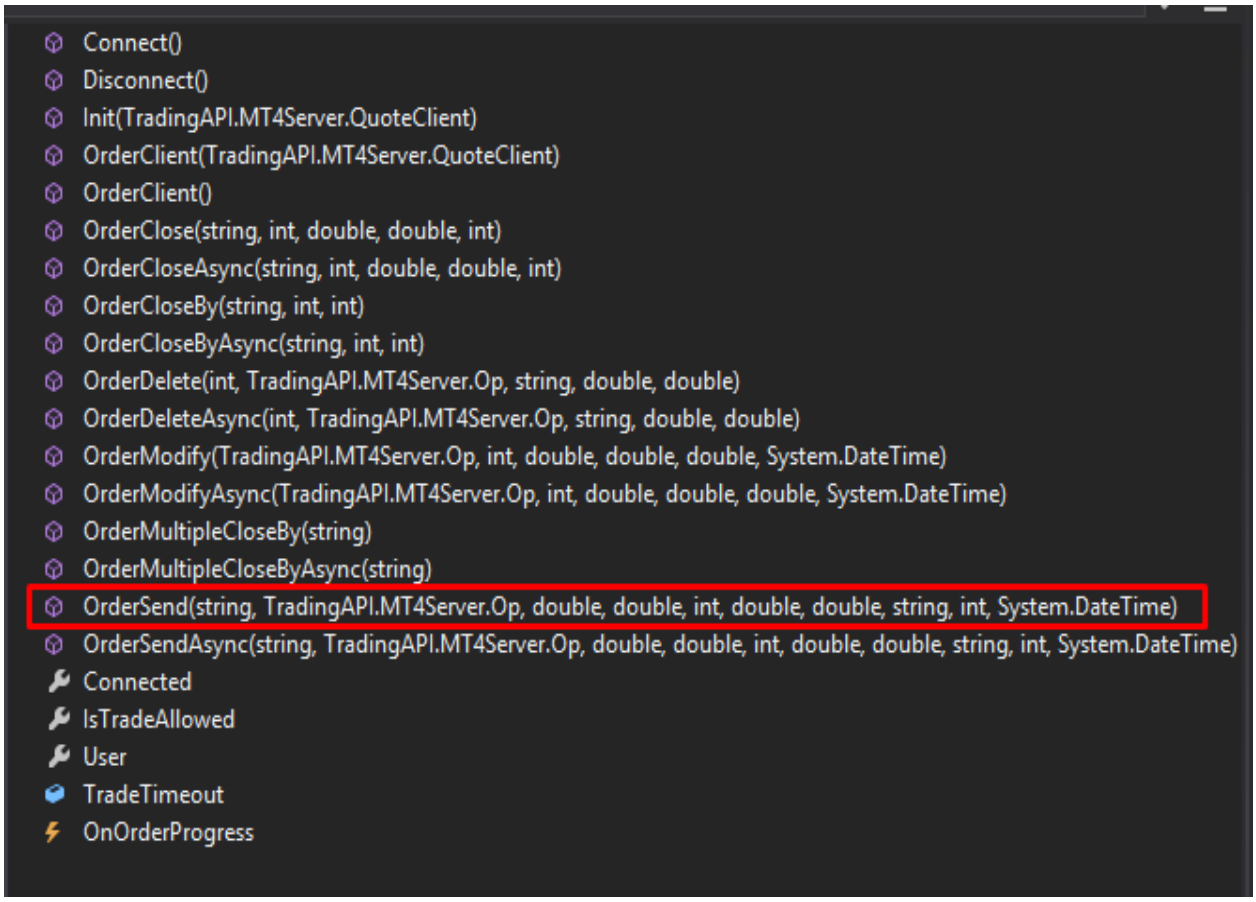

Рисунок 3.4 – Інтерфейс брокера Metatrader

Тобто нам потрібно сформувати клас який би містив всі необхідні поля для того щоб брокер міг здійснити торгову операцію. Цей клас буде формуватися у вихідному адаптері на основі подій які будуть приходити від технічного індикатора та направлятися споживачеві тобто брокерові. UML діаграма його приведена на рисунку 3.5.

Таке розмежування внутрішньої логіки роботи системи, тобто алгоритму по якому здійснюється торгівельна операція і фіда з брокером дозволить нам в майбутньому легко заміняти джерело даних чи споживача або додавати нові.

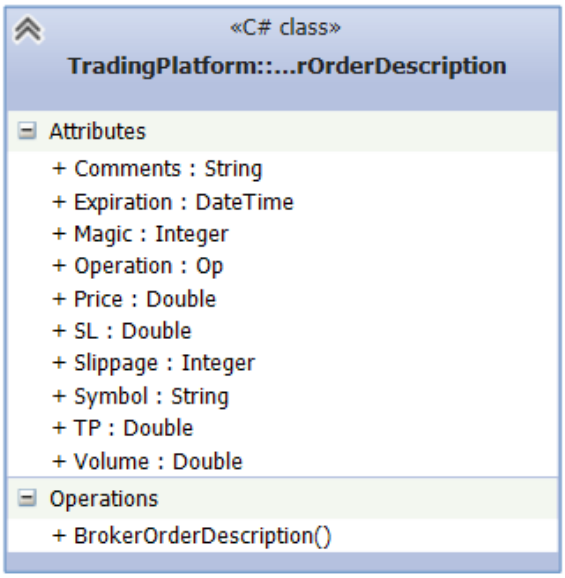

Рисунок 3.5 – UML діаграма класу трейда з вихідного адаптера

Отже, в мене є типи даних які приходять на вхід системи, типи даних якими маніпулюватиме системи і потрібні на виході параметри для здійснення торгівельної операції. На основі цих даних можна імплементовувати вхідні та вихідні адаптери.

3.3 Проектування та реалізація вхідного та вихідного адаптерів

На основі спроектованого інтерфейсу адаптерів системи і тих даних які на потрібні складемо діаграму обміну повідомленнями між рівнями системи, рисунок 3.6, з описом формату даних.

Отже, для реалізації вхідного адаптера нам потрібно клас наслідувати від інтерфейсу `IInputAdapter`, та реалізувати його поле типу `event`. В середині цього адаптера потрібно реалізувати підписку на подію фіда про нову ціну на ринку або це можна зробити ззовні. Далі адаптер повинен розпарсити і заповнити всі поля класу `Quote`, тобто: `BidPrice`, `AskPrice` і `Date`. Також він повинен провести валідацію цих параметрів і обрати ціни лише потрібної валютної пари.

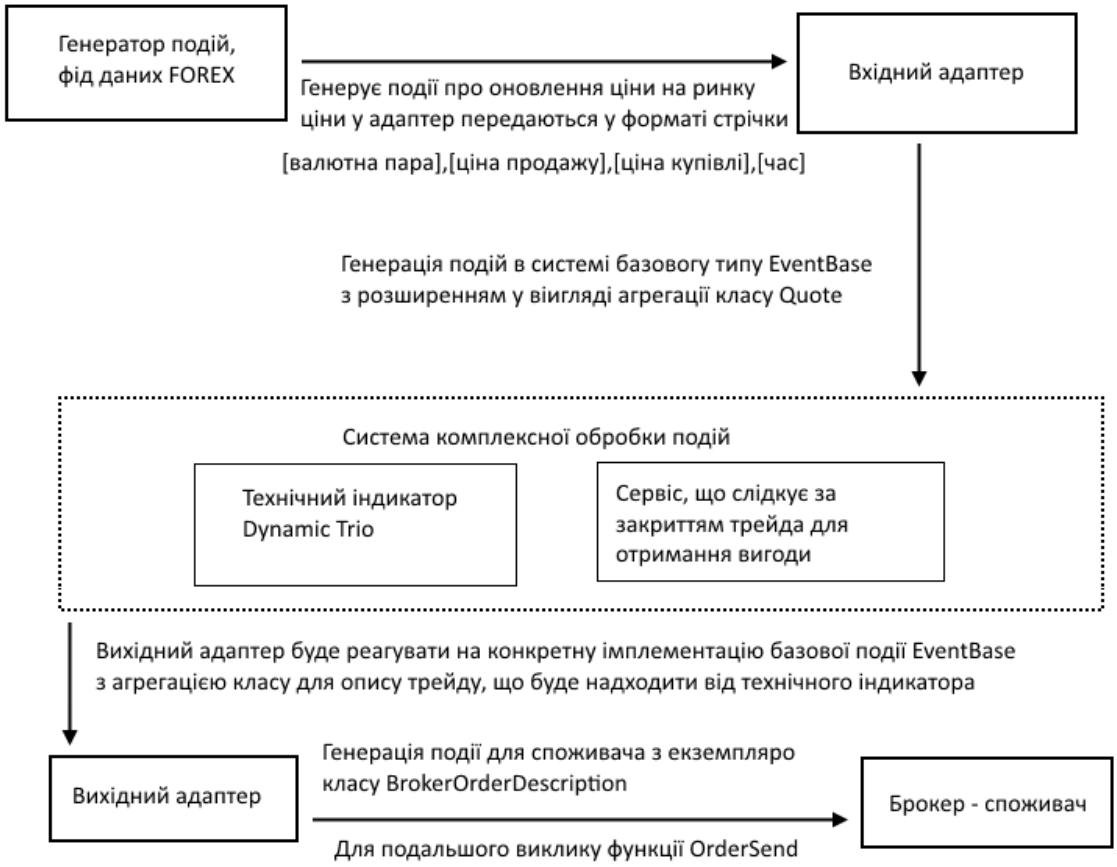

Рисунок 3.6 – Потік подій в системі з описом типів даних

Вхідний адаптер буде генерувати подію про оновлення ціни, UML діаграма класу цієї події показано на рисунку 3.7, а код приведено в лістингу 3.2.

На діаграмі видно, що `QuoteEvent` наслідується від базового класу `EventBase` та агрегує об’єкт класу `Quote`. Тобто `QuoteEvent` несе інформацію про поточну ціну в систему.

В системі цей об’єкт буде представлений через базовий клас події, але всі бажаючі зможуть перевірити його реальний тип та привести його до потрібного. Наприклад так буде робити індикатор для отримання з поля Quote події поточної ціни.

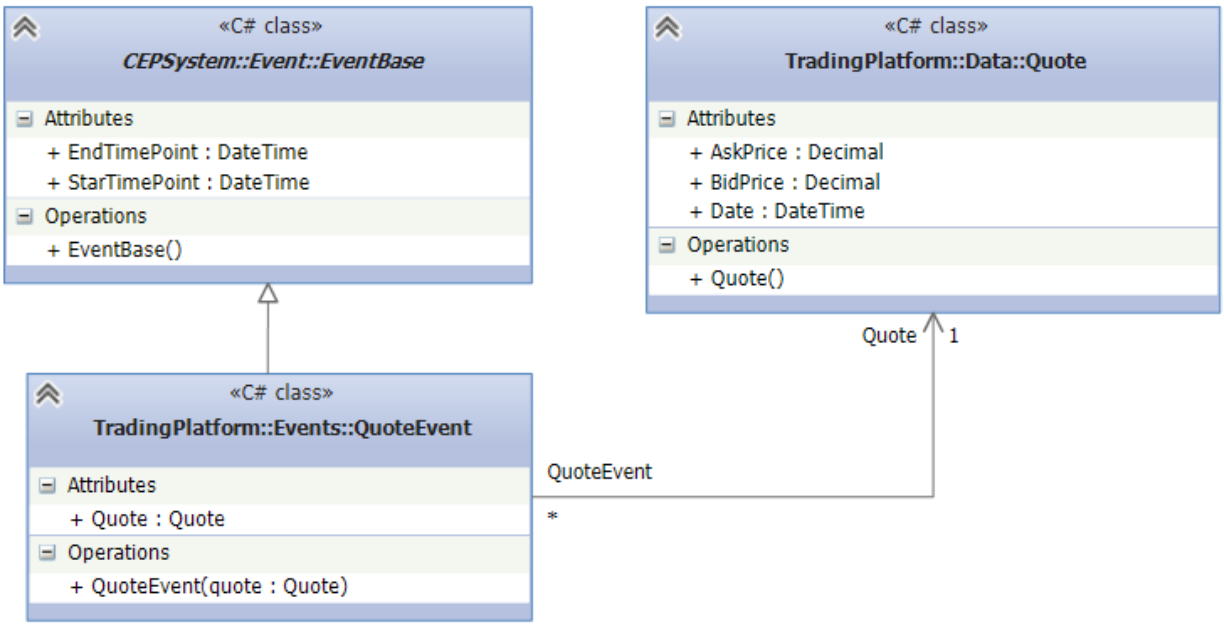

Рисунок 3.7 – UML діаграма класу події що генерується вхідним адаптером

Лістинг 3.2 – Реалізація класу `QuoteEvent` та базового класу `EventBase`

```
namespace TradingPlatform.Events
{
    public class QuoteEvent : EventBase
    {
        public Quote Quote { get; set; }

        public QuoteEvent(Quote quote)
        {
            Quote = quote;
        }
    }
}

namespace CEPSystem.Event
{
    public abstract class EventBase
    {
        public DateTime StarTimePoint { get; private set; }

        public DateTime EndTimePoint { get; private set; }
    }
}
```

Вихідний адаптер в свою чергу буде генерувати подію `OrderEvent`, що містить об'єкт `BrokerOrderDescription` для брокера. Це клас показано на рисунку 3.8, а код приведено в лістингу 3.3.

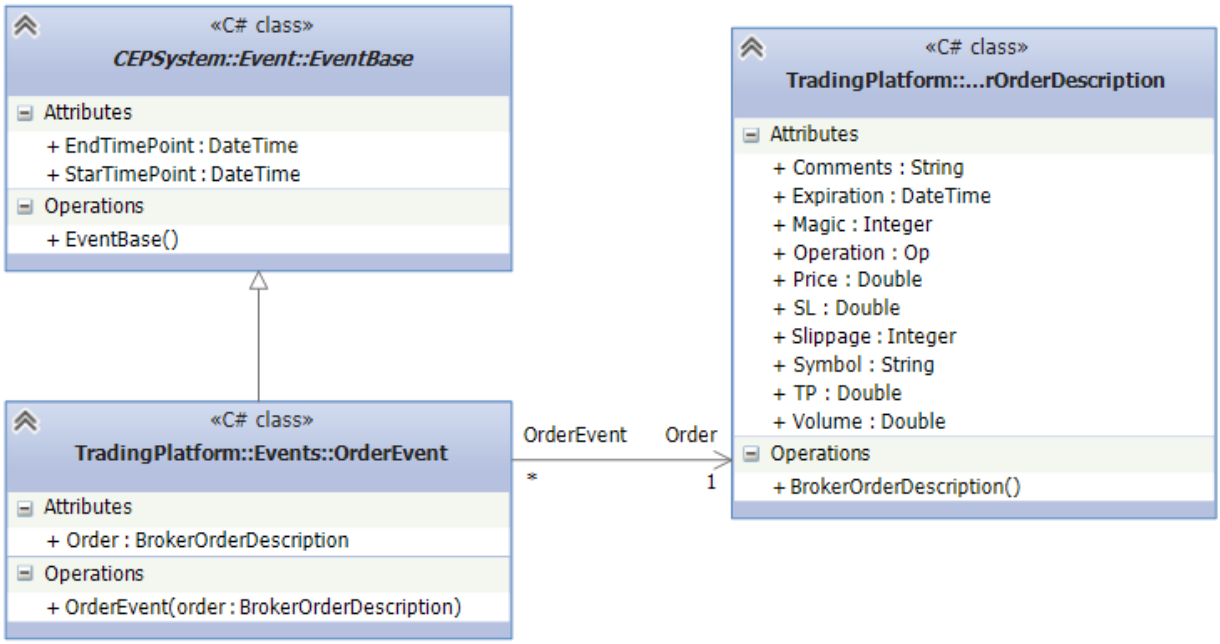

Рисунок 3.8 – UML діаграма класу події що генерується вихідним адаптером

Лістинг 3.3 – Реалізація класу `OrderEvent` та класу, що описує трейд `BrokerOrderDescription`

```
namespace TradingPlatform.Events
{
    public class OrderEvent : EventBase
    {
        public BrokerOrderDescription Order { get; set; }

        public OrderEvent(BrokerOrderDescription order)
        {
            Order = order;
        }
    }
}

namespace TradingPlatform.Broker
{
    public class BrokerOrderDescription
    {
```

```
            public string Symbol { get; set; }
            public Op Operation { get; set; }
            public double Volume { get; set; }
            public double Price { get; set; }
            public int Slippage { get; set; }
            public double SL { get; set; }
            public double TP { get; set; }
            public string Comments { get; set; }
            public int Magic { get; set; }
            public DateTime Expiration { get; set; }
        }
    }
```

Тепер коли ми визначилися зі всіма компонентами системи і тими типами даних які будуть використовуватися на всіх її рівнях можна реалізувати вхідний та вихідний адаптери. Реалізація вхідного адаптера приведена в лістингу 3.4, його UML діаграма приведена на рисунку 3.9, а вихідного в лістингу 3.5, та на рисунку 3.10.

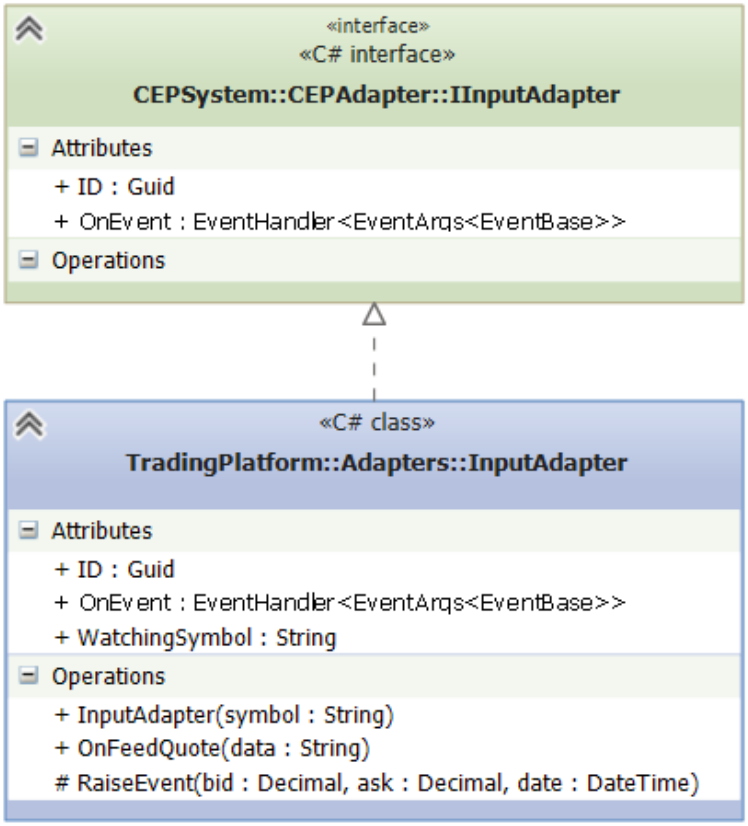

Рисунок 3.9 – UML діаграма вхідного адаптера

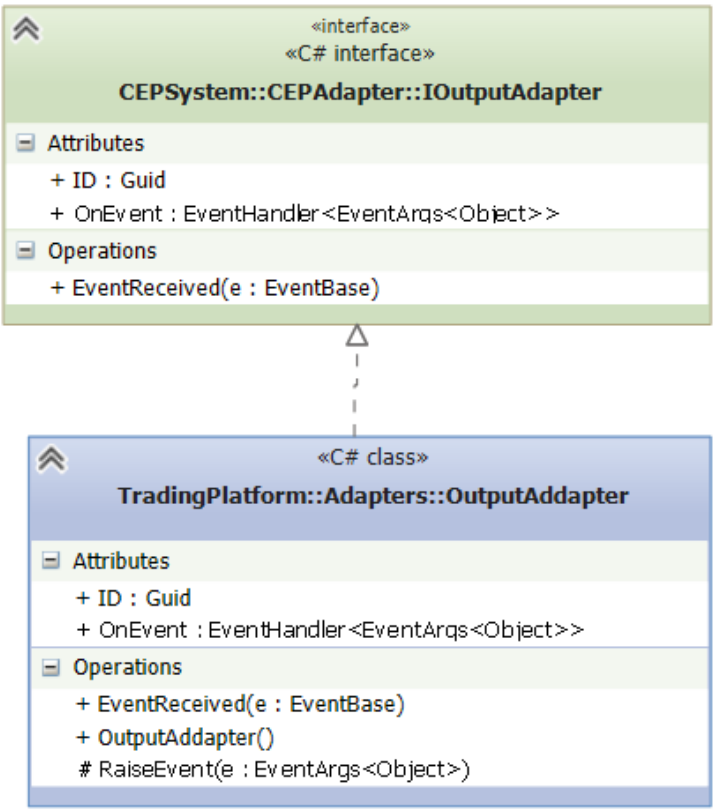

Рисунок 3.10 – UML діаграма вихідного адаптера

Лістинг 3.4 – Реалізація вхідного адаптера системи

```
public class InputAdapter : IInputAdapter
{
    public event EventHandler<EventArgs<EventBase>> OnEvent;

    public Guid ID { get; private set; }

    public string WatchingSymbol { get; set; }

    public InputAdapter(string symbol)
    {
        WatchingSymbol = symbol;
        ID = Guid.NewGuid();
    }

    public void OnFeedQuote(string data)
    {
        var quoteArray = data.Split(';');
        foreach (var line in quoteArray)
        {
            var currentTick = line.Substring(1, line.Length -
1).Split(',');
            if (currentTick.Length != 4)
                continue;
```

```
                decimal bid, ask;
                DateTime date; // Server return data in UTC
format

                 if            (!decimal.TryParse(currentTick[1],
NumberStyles.Any, CultureInfo.InvariantCulture, out bid) ||
                    !decimal.TryParse(currentTick[1],
NumberStyles.Any, CultureInfo.InvariantCulture, out ask) ||
                    !DateTime.TryParse(currentTick[2],
CultureInfo.InvariantCulture,   DateTimeStyles.AdjustToUniversal,
out date))
                    continue;

                date         =         DateTime.SpecifyKind(date,
DateTimeKind.Utc);

                if (date > DateTime.UtcNow.AddDays(1))
                    continue;

                if (date < DateTime.UtcNow.AddDays(-1))
                    continue;

                if (!WatchingSymbol.Equals(currentTick[0]))
                    continue;

                RaiseEvent(bid, ask, date);
            }
        }

        protected virtual void RaiseEvent(decimal bid, decimal
ask, DateTime date)
        {
            var e = new QuoteEvent(new Quote
            {
                Date = date,
                AskPrice = ask,
                BidPrice = bid
            });

            var handler = OnEvent;
            if    (handler    !=    null)    handler(this,    new
EventArgs<EventBase>(e));
        }
    }
```

Лістинг 3.5 – Реалізація вихідного адаптера системи

```
public class OutputAddapter : IOutputAdapter
{
    public event EventHandler<EventArgs<object>> OnEvent;

    public Guid ID { get; private set; }
```

```
    public OutputAddapter()
    {
        ID = Guid.NewGuid();
    }

    public void EventReceived(EventBase e)
    {
        var orderEvent = e as OrderEvent;

        if(orderEvent == null)
            return;

        RaiseEvent(new                          EventArgs<object>(new
 BrokerOrderDescription
        {
            Comments = string.Empty,
            Expiration = DateTime.UtcNow.AddDays(1),
            Magic = 0,
            Operation  =  orderEvent.Order.Size  >  0  ?  Op.Buy  :
     Op.Sell,
            Symbol = orderEvent.Order.Symbol,
            Volume = Math.Abs(orderEvent.Order.Size)
        }));
    }

    protected virtual void RaiseEvent(EventArgs<object> e)
    {
        var handler = OnEvent;
        if (handler != null) handler(this, e);
    }
}
```

Отже реалізовано вхідний та вихідний адаптери системи, вхідний адаптер буде приймати потік даних через функцію `OnFeedQuote` ззовні у вигляді стрічки, далі парсити та валідувати і якщо все добре направляти потік даних в систему. Система передбачена для моніторингу лише однієї валютної пари тому цей адаптер в конструктор приймає стрічку – валютну пару за якою буде проводити фільтрацію.

Вихідний адаптер приймає події з системи за допомогою функції `EventReceived` з параметром – подією, та слідкує лише за подіями які б ініційовували трейд, тобто типом `OrderEvent`.

Результат роботи системи показано на рисунку 3.11, валютна пара EUR/USD, дві лінії на графіку це відображення технічного індикатора Dynamic Trio.

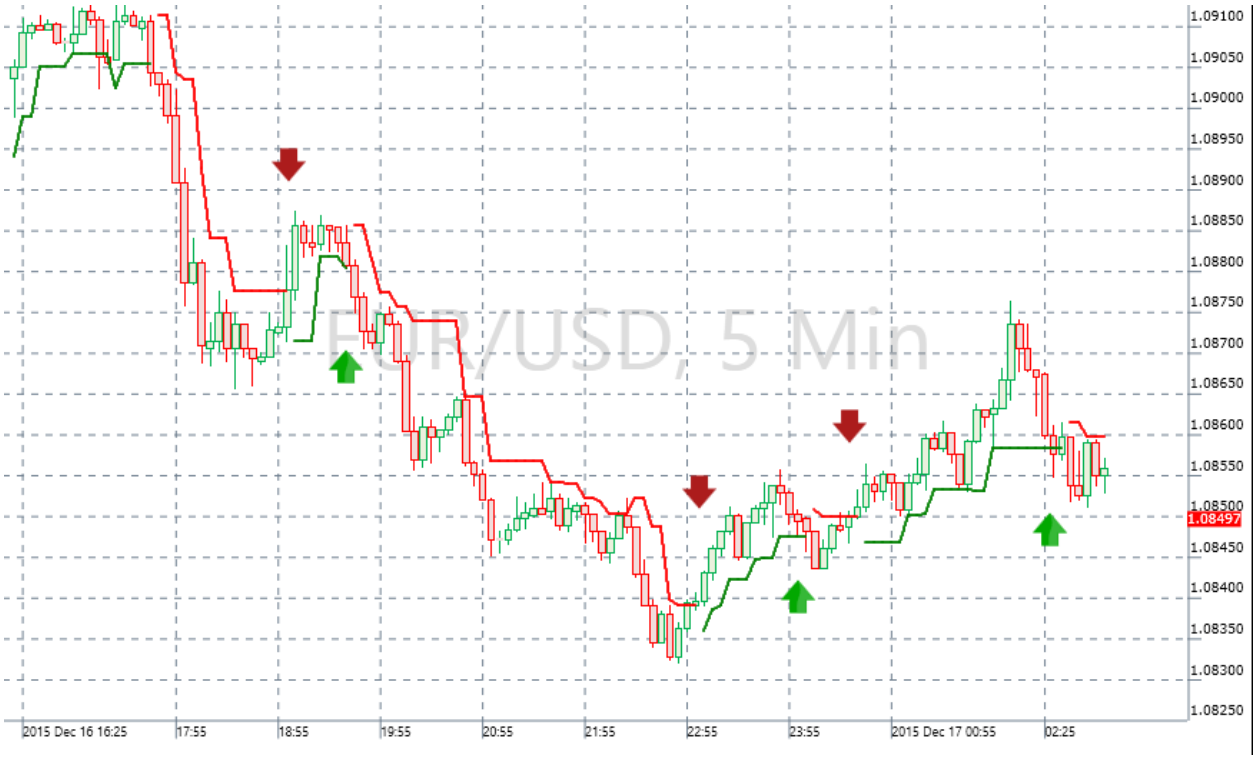

Рисунок 3.11 – Результати роботи

Стрілка “Down” – показує моменти де було здійснено купівлю, стрілка “Up” – продаж. Тобто торгівельна операція відбувається в момент зміни тренду ринку, про що нам сигналізує індикатор.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Темою дипломної роботи є розробка системи уніфікації птоків даних. Дотримання основних вимог з охорони праці при роботі з ПК дозволяє безпечно використовувати програмні продукти. Тому необхідно розглянути принципи та схему захисного відключення та правила що стосуються приміщень та робочих місць де використовується ПК.

4.1 Розробка та схема захисного відключення в установці, що проектується

Захисним відключенням називається автоматичне відключення електроустановок при однофазному (однополюсному) дотику до частин, що знаходяться під напругою, неприпустимим для людини, і (або) при виникненні в електроустановці струму витоку, що перевищує задані значення.

Призначення захисного відключення - забезпечення електробезпеки, що досягається за рахунок обмеження часу впливу небезпечного струму на людину. Захист здійснюється спеціальним пристроєм захисного відключення (ПЗВ), яке, працюючи в черговому режимі, постійно контролює умови ураження людини електричним струмом[13].

Принцип роботи ПЗВ полягає в тому, що воно постійно контролює вхідний сигнал і порівнює його з наперед заданої величиною (уставкою). Наприклад, значення уставок повинні вибиратися для мереж з глухозаземленою нейтраллю - з ряду 0,002; 0,006; 0,01; 0,02; 0,03; 0,1; 0,3; 0,5; 1,0 А. Якщо вхідний сигнал перевищує уставку, то пристрій спрацьовує і відключає захищену електроустановку від мережі. В якості вхідних сигналів пристроїв захисного відключення використовують різні параметри електричних мереж, які несуть у собі інформацію про умови ураження людини електричним струмом.

Область застосування - електроустановки в мережах з будь-яким напругою і будь-яким режимом нейтралі. Найбільшого поширення захисне відключення отримало в електроустановках, що використовуються в мережах напругою до 1 кВ із заземленою або ізольованою нейтраллю.

Основними елементами будь-якого пристрою захисного відключення є датчик,перетворювач і виконавчий орган.

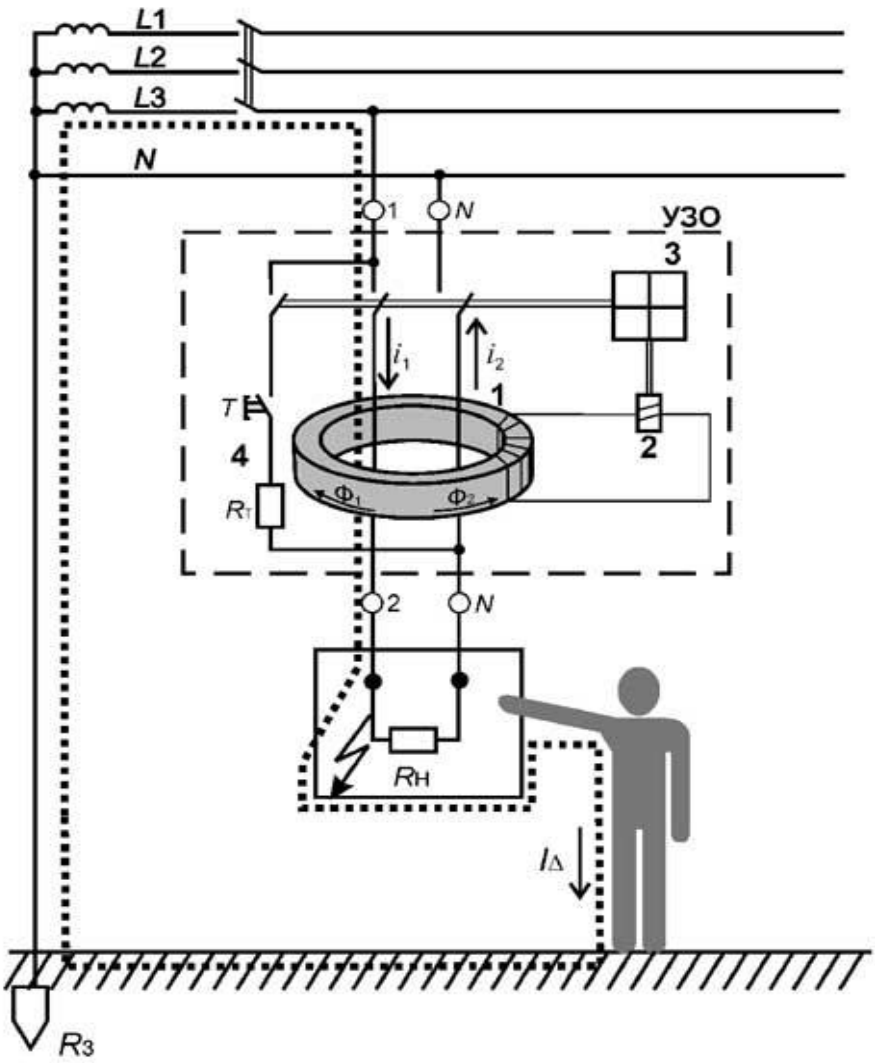

Рисунок 4.1 Схема, що ілюструє принцип дії ПЗВ

При дотику людини до відкритих струмопровідних частин або до корпуса електроприймача, який в результаті пробою ізоляції опинився під напругою, за фазного провідника через ПЗВ крім струму навантаження L1 потече додатковий струм ID (струм витоку), що є для трансформатора струму

диференціальним (різницевим). Нерівність струмів в первинних обмотках - I1 + ID в фазному провіднику і L2 = L1 в нульовому робочому провіднику - викликає небаланс магнітних потоків і, як наслідок, виникнення у вторинній обмотці трансформованого диференціального струму. Якщо цей струм перевищує задане значення струму порогового елемента пускового органу 2, останній спрацьовує і впливає на виконавчий механізм 3. Виконавчий механізм, зазвичай складається з пружинного приводу, спускового механізму і групи силових контактів, розмикає електричний ланцюг. У результаті захищається ПЗВ електроустановка знеструмлюється. Дляздійснення періодичного контролю справності (працездатності) ПЗВ передбачена ланцюгтестування 4. При натисканні кнопки "Т" штучно створюється ланцюг протікання вимикаючого диференціального струму. Спрацювання ПЗВ в цьому випадку означає, що пристрій в цілому справно.

4.2 Технічні та організаційні заходи щодо зменшенню рівня шуму та вібрацій на дільниці, що проектується

Шум - неприємний або небажаний звук чи сукупність звуків, що заважають сприйняттю корисних звукових сигналів, порушують тишу, чинять шкідливу або подразливу дію на організм людини, знижують її працездатність[14].

Боротьба з шумом на виробництві є однією з найскладніших проблем, оскільки джерела шуму різноманітні й потребують комплексу заходів технічного, організаційного і медичного характеру на всіх стадіях проектування, будівництва, експлуатації машин і устаткування. Відомі три основні напрямки боротьби з шумом:

1. зменшення рівня шуму у джерелі виникнення, застосування раціональних конструкцій, нових матеріалів і технологічних процесів;

2. звукоізоляція устаткування за допомогою глушників, резонаторів, кожухів, захисних конструкцій, оздоблення стін, стелі, підлоги тощо;

3. використання засобів індивідуального захисту.

Дуже часто як супутній фактор шуму на робочих місцях виникає вібрація, тому система профілактичних засобів зниження шуму є комплексною проблемою загального захисту працюючих від механічних коливань.

Технологічні заходи охоплюють характеристику і розміщення устаткування і машин, вимоги до розрахунку характеристик шуму на стадії проектування, обмеження шуму звукопоглинаючих конструкцій і екранів, фільтровентиляційних установок, заміну технологічних процесів і механізмів на менш шумні, обладнання звукоізолюючих кабін операторів, дистанційне керування обладнанням, автоматизацію виробничих процесів зі зменшенням кількості операторів тощо.

Планувальні заходи передбачають ізоляцію шумних цехів від тихих приміщень, збільшення відстані між ними (на стадії проектування виробництва), розташування шумних цехів з підвітряного боку і торцем до фасаду інших будівель. Зелені насадження навколо шумних цехів і шумозахисна зона так само сприяють поглинанню шуму.

У виробничих умовах поряд із звукоізоляцією широко застосовують засоби звукопоглинання. З метою поглинання шуму приміщеннями цехів малого об'єму (400-500 м3) їх оздоблюють пористими матеріалами. Позитивний ефект звукопоглинання дає застосування мінеральних плит, матів з базальтового волокна, штукатурки пінистої або зернистої структури тощо. У приміщеннях великого об'єму ефективні звукопоглинаючі бар'єри і об'ємні поглиначі (куби, конуси тощо), які підвішують над шумними агрегатами для зниження рівня шуму на 5-12 дБ. Застосування звукопоглинаючих матеріалів у комплексі із заміною устаткування в окремих випадках знижує рівень шуму до нормативного (ткацькі цехи).

Засоби індивідуального захисту від шуму - протишуми - використовують тоді, коли технічні засоби не забезпечують його зниження до безпечного рівня. Тип засобу протишуму вибирають за рівнем і спектром шуму. Застосовують десятки варіантів вкладишів (втулки, тампони тощо), навушники і шоломи для ізоляції зовнішнього слухового ходу від шуму різного спектрального складу. До протишумових вкладишів, які вставляють у слуховий хід, належать заглушки у вигляді тампонів, гумові ковпачки, циліндри із спеціального пінопласту, пластичні вкладиші (виготовлені індивідуально за формою слухового ходу), а також вкладиші одноразового використання. Ефективними вважаються вкладиші із суміші волокон органічної бактерицидної вати і ультратонких полімерних волокон – беруши.

Людина сприймає звуки частотою 16…20 000 Гц. Звуки з частотою до 16 Гц називаються інфразвуками, а понад 20 000 Гц - ультразвуками. Хоча вони вухом не сприймаються, зате відчуваються тканинами організму.

На виробництві шум може бути постійним і непостійним, коли рівень його під час роботи змінюється більше ніж на 5 дБ. Непостійні шуми поділяються на перервні, імпульсні та флюктуючі, коли рівень шуму весь час коливається.

Негативний вплив шуму на нервову систему працівника виявляється у головних болях, безсонні, швидкій стомлюваності, підвищеному потовиділенні, треморі пальців і рук, підвищеному роздратуванні, порушеннях пам'яті і уваги, а на серцево-судинну систему - у болях в області серця, зменшенні частоти пульсу, гіпотонії або гіпертонії.

Слід зазначити, що при певних умовах тиша може справляти негативний вплив і знижувати продуктивність праці, оскільки навіть незначні звуки привертають увагу працівника, відволікаючи від роботи. Нормальний шумовий фон підвищує рівень збудження і позитивно впливає на працездатність людини. Тому при виконанні багатьох робіт доцільним є стимулюючий рівень шуму у вигляді музики, яка до того ж створює гарний настрій.

Однак подальше підвищення шуму знижує працездатність, а сам шум починає дратувати людину, внаслідок чого погіршується її увага. До того ж залежно від емоційного забарвлення, мажорності та інтенсивності звукового подразника звуки можуть сприйматися як неприємні, страшні, гнітючі, неспокійні, втомливі, стимулюючі, веселі, надоїдливі і т. п. Тому на виробництві слід уникати шумів, які справляють негативний вплив на психічні стани працівників, перешкоджають контактам між ними.

Основними напрямками боротьби з шумом на виробництві є розробка і впровадження заходів технічного характеру, які виключали б причини генерування шуму; виведення персоналу із зон з високим рівнем шуму за рахунок впровадження дистанційного управління; впровадження фізіологічно обґрунтованих режимів праці і відпочинку; застосування індивідуальних захисних засобів тощо.

При обслуговуванні ультразвукового обладнання профілактичні заходи передбачають попередження контактного озвучування через тверді та рідкі середовища і боротьбу з поширенням ультразвуку й шуму в повітрі робочої зони. Ультразвукове устаткування слід обладнувати звукоізолюючими кожухами, конструкції ультразвукових верстатів і устаткування для зварювання та паяння повинні мати екрани з органічного скла, які забезпечують зниження рівнів звукового тиску на робочих місцях. Забороняється контакт з робочими поверхнями устаткування у процесі його роботи, з оброблюваними рідинами і деталями. Для боротьби з контактним озвучуванням слід застосовувати дистанційне керування, автоблокування, тобто автоматичне вимкнення устаткування і приладів при завантажуванні та розвантажуванні продукції, нанесенні контактних мастил, а також спеціальні пристрої для завантажування і виймання деталей, затискачі, щипці, ручки яких повинні мати еластичне покриття, що поглинає ультразвук.

Індивідуальний захист органу слуху досягається застосуванням протишумів. Для захисту рук від впливу ультразвуку в зоні контакту з твердим або рідким середовищем слід застосовувати захисні рукавички. До

роботи з ультразвуковим устаткуванням допускаються особи віком понад 18 років.

Під інфразвуком розуміють акустичні коливання з частотою до 20 Гц. Фізична природа чутного звуку, ультразвуку та інфразвуку однакова, їх поділ зумовлений особливостями сприйняття їх слуховим аналізатором людини. Для інфразвуку характерні дуже великі пороги слухового сприйняття, що робить його практично нечутним. Фізичні особливості інфразвукових коливань зумовлені їх малою частотою і великою довжиною хвиль. Характерною ознакою інфразвуку є його здатність поширюватися на значну відстань без істотної втрати енергії, огинати перепони внаслідок дифракції або проникати крізь них.

Боротьба з несприятливим впливом виробничого інфразвуку охоплює комплекс заходів, які належать до технічної і медичної компетенції. Розглянемо окремі з них:

1) Ослаблення інфразвуку в межах джерела, усунення причин його виникнення, що є найрадикальнішим способом боротьби з низькочастотними коливаннями машин і механізмів;

2) Ізоляція інфразвуку. Важливе місце у боротьбі з інфразвуком належить методам будівельної акустики. Велике значення має раціональне планування і розміщення виробничого устаткування, ізоляція в окремих приміщеннях агрегатів - джерел шуму та інфразвуку. Водночас слід наголосити, що застосування звукопоглинаючого оздоблення звичайного типу практично не ослаблює енергії звукових коливань;

3) Поглинання інфразвуку. Для цього застосовують багатошарові звукопоглинаючі покриття.

4) Медична профілактика. Одним з найважливіших заходів медичної профілактики шкідливого впливу інфразвуку є здійснення запобіжних і періодичних медичних оглядів. Протипоказаннями для прийняття на роботу є порушення вестибулярної і слухової функції, виражені неврози, вегетативна

дисфункція, захворювання центральної нервової та серцево-судинної систем, органів травлення.

Ультразвук високочастотного діапазону викликає підвищення проникності судин шкіри, що виражається гіперемією аж до крововиливів на поверхні шкіри (петехій).

Під час контактної дії ультразвуку підвищується серцевий ритм, помітно змінюється ЕКГ; при збільшенні його інтенсивності виникає аритмія, а в окремих випадках - зупинка серця (у піддослідних тварин). Аналогічні реакції спостерігаються і в людей: виникають неприємні відчуття при озвучуванні грудної клітки, згодом розвиваються тахікардія та стенокардія.

При обслуговуванні ультразвукового обладнання профілактичні заходи передбачають попередження контактного озвучування через тверді та рідкі середовища і боротьбу з поширенням ультразвуку й шуму в повітрі робочої зони. Ультразвукове устаткування слід обладнувати звукоізолюючими кожухами, конструкції ультразвукових верстатів і устаткування для зварювання та паяння повинні мати екрани з органічного скла, які забезпечують зниження рівнів звукового тиску на робочих місцях. Забороняється контакт з робочими поверхнями устаткування у процесі його роботи, з оброблюваними рідинами і деталями. Для боротьби з контактним озвучуванням слід застосовувати дистанційне керування, автоблокування, тобто автоматичне вимкнення устаткування і приладів при завантажуванні та розвантажуванні продукції, нанесенні контактних мастил, а також спеціальні пристрої для завантажування і виймання деталей, затискачі, щипці, ручки яких повинні мати еластичне покриття, що поглинає ультразвук.

Індивідуальний захист органу слуху досягається застосуванням проти шумів. Для захисту рук від впливу ультразвуку в зоні контакту з твердим або рідким середовищем слід застосовувати захисні рукавички. До роботи з ультразвуковим устаткуванням допускаються особи віком понад 18 років.

Вібрація — рух матеріальної точки або механічної системи, при якому почергово зростають і спадають за часом значення величини, що характеризує цей рух .

Для захисту від вібрації застосовують такі методи: зниження віброактивності машин; відбудова від резонансних частот; вібродемпфірованіє; віброізоляція; виброгашение, а також індивідуальні засоби захисту. Налаштування від резонансних частот полягає в зміні режимів роботи машини і відповідно частоти вимушених вібросіли; власної частоти коливань машини шляхом зміни жорсткості системи з наприклад установкою ребер жорсткості або зміни маси системи (наприклад шляхом закріплення на машині додаткових мас).

Вібродемпфірування - це метод зниження вібрації шляхом посилення в конструкції процесів тертя, розсіюють коливальну енергію в результаті необоротного перетворення її в теплоту при деформаціях, що виникають у матеріалах, з яких виготовлена конструкція. Вібродемпфірування здійснюється нанесенням на вібруючі поверхні шару упруговязкіх матеріалів, що володіють великими втратами на внутрішнє тертя, - м'яких покриттів (гума, пінопласт ПХВ-9, мастика ВД17-59, мастика «Анти-вібро») та жорстких (листові пластмаси, Стеклоїзола, гідроізол, листи алюмінію); застосуванням поверхневого тертя (наприклад, прилеглих один до одного пластин, як у ресор); установкою спеціальних демпферів.

Підвищення жорсткості системи, наприклад шляхом встановлення ребер жорсткості. Цей спосіб ефективний тільки при низьких частотах вібрації.

Профілактичні заходи щодо захисту від вібрацій полягають у зменшенні їх у джерелі освіти і на шляху поширення, а також у застосуванні індивідуальних засобів захисту, проведення санітарних та організаційних заходів.

Зменшення вібрації в джерелі виникнення досягають зміною технологічного процесу з виготовленням деталей з капрону, гуми, текстоліту,

своєчасним проведенням профілактичних заходів та мастильних операцій; центруванням і балансуванням деталей; зменшенням зазорів у з'єднаннях. Передачу коливань на підставу агрегату або конструкцію будинку послаблюють допомогою екранування, що є одночасно засобом боротьби і з шумом.

Тривалість роботи з вібруючим інструментом не повинна перевищувати 2 / 3 робочої зміни. Операції розподіляють між працівниками так, щоб тривалість безперервної дії вібрації, включаючи мікропаузи, не перевищувала 15 ... 20 хв. Рекомендується робити перерви на 20 хв через 1 ... 2 год після початку зміни і на 30 хв через 2 години після обіду.

ВИСНОВКИ

Таким чином метою виконаної кваліфікаційної роботи на здобуття освітнього ступеня «магістр» є необхідність в розробці спеціалізованої комп'ютерної системи, робота якої передбачає певний визначений алгоритм виявлення тренду ринку і, залежно від його тенденції, прийняття відповідного рішення та здійснення відповідного втручання.

Для досягнення поставленої мети було вирішено наступні задачі

- визначено логічний рівень в системі який буде відповідати за уніфікацію вхідних та вихідних даних;
- визначено протоколи даних які будуть достатніми з функціональної точки зору для передачі даних між рівнями системи;
- визначено методи обробки даних, які будуть доступні в системі
- визначено яким чином методи обробки даних за необхідності можуть розширюватись;
- спроектовано оптимальну концептуальну модель системи;
- спроектовано оптимальну концептуальну архітектуру системи;
- визначено та реалізовано ті особливості системи, які будуть необхідні саме для обраного об'єкту дослідження.

Було проаналізовані та обґрунтовані рішення щодо обраних технологій розробки. Зокрема, методи розробки базується на технології C#, для реалізації застосовано платформу .Net.

Дипломна робота складається з вступу, чотирьох розділів, висновку, списку літератури та додатків.

Роботу пройшла апробацію в рамках VIII науково-технічнічної конференції «Інформаційні моделі, системи та технології», м. Тернопіль, 2020 р. – Тернопіль: ТНТУ ім. І. Пулюя (м. Тернопіль, 11-12 грудня 2020 року), 2020.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. K. Mani Chandy Event-Driven Applications: Costs, Benefits and Design Approaches, California Institute of Technology, 2006

2. Jeff Hanson, Event-driven services in SQA**,** Javaworld, January 31, 2005 [Електронний ресурс] – Режим доступу: URL : Event-driven services in SOA

3. Event-driven archiyecture poised for wide adoption [Електронний ресурс] – Режим доступу: URL: Carol Sliwa Event-driven architecture poised for wide adoption, Computerworld, May 12, 2000

4. Staged event-driven architecture [Електронний ресурс] / Wikipedia – URL: https://en.wikipedia.org/wiki/Staged_event-driven_architecture – 18.11.2015 р.

5. Event-driven architecture (EDA) [Електронний ресурс] / Wikipedia – URL: https://en.wikipedia.org/wiki/Event-driven_architecture – 20.11.2015 р.

6. [Електронний ресурс] – Режим доступу: URL: http://eventdrivenpgm.sourceforge.net/

7. [Електронний ресурс] – Режим доступу: URL: https://www.w3.org/TR/xml-events/Overview.html#section-eventhandlers

8. Software architecture [Електронний ресурс] / Wikipedia – URL: [Електронний ресурс] / Wikipedia – URL: https://en.wikipedia.org/wiki/Software_architecture – 22.11.2015 р.

9. Метрика коду в VS2012 [Електронний ресурс] / URL: http://habrahabr.ru/post/111524/

10. Conceptual model [Електронний ресурс] / Wikipedia – URL: https://en.wikipedia.org/wiki/Conceptual_model – 25.11.2015 р.

11. [Електронний ресурс] – Режим доступу: URL: http://epthinking.blogspot.com/2008/04/on-event-processing-agents.html

12. [Електронний ресурс] – Режим доступу: URL: https://www.sciencedirect.com/science/article/pii/S0898122113004677

13. Пристрої захисного відключення як ефективний засіб запобігання спалахів [Текст] Монаков В.К. – М.:ЗАО Енергосервіс - 2003. - 195с.

14. Охорона праці : Навчальний посібник [Текст] Геврик Є.О. - Ельга: Ніка-Центр. - 2003. – 279 с.
15. Запобігання виникненню надзвичайних ситуацій [Електронний ресурс] / Wikipedia – URL: https://uk.wikipedia.org/wiki/Запобігання_виникненню_надзвичайних_ситуацій_характеру – 02.12.2015 р.
16. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020- 51с.

ДОДАТКИ

Індикатор Dynamic Trio

```
internal class CustomIndicators
{
            public    Recordset    DynamicTrio(Navigator    nav,
Recordset ohlc, int barsBack, bool wuc, double pips, double
pipsValue)
            {
                var recordCount = nav.RecordCount;
                var result = new Recordset();
                var longStartField = new Field();
                var shortStartField = new Field();
                longStartField.Initialize(recordCount,
"LongStart");
                shortStartField.Initialize(recordCount,
"ShortStart");
                var directionField = new Field();
                directionField.Initialize(recordCount,
"Direction");

                var longSource = wuc ? ohlc.GetField("Close") :
ohlc.GetField("Low");
                var shortSource = wuc ? ohlc.GetField("Close") :
ohlc.GetField("High");

                for (var i = 0; i < barsBack; i++)
                {
                    longStartField.SetValue(i, null);
                    shortStartField.SetValue(i, null);
                    directionField.SetValue(i, null);
                }

                var highField = ohlc.GetField("High");
                var lowField = ohlc.GetField("Low");
                var    switchToLong    =    new    List<int>(new
int[recordCount + 2]);
                var    switchToShort    =    new    List<int>(new
int[recordCount + 2]);

                var sp = new List<double>(new double[recordCount
+ 2]);
                var lp = new List<double>(new double[recordCount
+ 2]);

                var general = new General();
                var    hhv    =    general.HHV(nav,    highField,
barsBack).GetField("HHV");
                var    llv    =    general.LLV(nav,    lowField,
barsBack).GetField("LLV");
```

```
                for (var i = barsBack; i <= recordCount; i++)
                {
                    var highValue = highField.ValueEx(i);
                    var lowValue = lowField.ValueEx(i);

                    if (highValue >= hhv.ValueEx(i))
                        longStartField.SetValue(i,
llv.ValueEx(i));
                    else
                        longStartField.SetValue(i,
longStartField.Value(i - 1));


                    if (lowValue <= llv.ValueEx(i))
                        shortStartField.SetValue(i,
hhv.ValueEx(i));
                    else
                        shortStartField.SetValue(i,
shortStartField.Value(i - 1));

                    sp[i]     =     shortSource.ValueEx(i)     -
pips*pipsValue;
                    lp[i]     =     longSource.ValueEx(i)     +
pips*pipsValue;

                    if (sp[i] >= shortStartField.ValueEx(i) &&
sp[i-1] < shortStartField.ValueEx(i - 1))
                        switchToLong[i] = 1;
                    else
                        switchToLong[i]= 0;
                    if (lp[i] <= longStartField.ValueEx(i) &&
lp[i - 1] > longStartField.ValueEx(i - 1))
                        switchToShort[i] = 1;
                    else
                        switchToShort[i] = 0;

                    if (directionField.Value(i - 1) == null)
                        directionField.SetValue(i, 0);
                    else
                    {
                        if (directionField.Value(i - 1) <= 0.0
&& switchToLong[i] == 1)
                            directionField.SetValue(i, 1);
                        else
                        {
                            if (directionField.Value(i - 1) >=
0.0 && switchToShort[i] == 1)
                                directionField.SetValue(i, -1);
                            else
                    directionField.SetValue(i,
directionField.Value(i - 1));
                        }
                    }
```

```
            }

            for (var i = barsBack; i <= recordCount; i++)
            {
                if (directionField.ValueEx(i) > 0)
                    shortStartField.SetValue(i, null);
                else
                    longStartField.SetValue(i, null);
            }

            result.AddField(longStartField);
            result.AddField(shortStartField);


            return result;
        }
    }
```

Реалізація класу Order

```
public class Order : INotifyPropertyChanged
{
    private DateTime _closeDate;
    private DateTime _openDate;
    private double _closePrice;
    private double _rate;
    private double _openPrice;
    private double _profit;
    private int _proximity;
    private double _sl;
    private double _tp;
    private double _currentPrice;
    private double _size;
    private string _namePlacedBy;
    private string _nameClosedBy;
    private double _pipProfit;

    public int Ticket { get; set; }

    public double OpenPrice
    {
        get { return _openPrice; }
        set
        {
            if (value != _openPrice)
            {
                _openPrice = value;
                OnPropertyChanged("OpenPrice");
            }
        }
    }

    public DateTime CloseTime
    {
        get
        {
            return _closeDate;
        }
        set
        {
            if (value != _closeDate)
            {
                _closeDate = value;
                OnPropertyChanged("CloseTime");
            }
        }
    }
```

```
public DateTime OpenTime
{
    get
    {
        return _openDate;
    }
    set
    {
        if (value != _openDate)
        {
            _openDate = value;
            OnPropertyChanged("OpenTime");
        }
    }
}

public double ClosePrice
{
    get
    {
        return _closePrice;
    }
    set
    {
        if (value != _closePrice)
        {
            _closePrice = value;
            OnPropertyChanged("ClosePrice");
        }
    }
}

public double Size
{
    get { return _size; }
    set
    {
        if (value.Equals(_size)) return;
        _size = value;
        OnPropertyChanged("Size");
    }
}

public string Symbol { get; set; }

public double CurrentPrice
{
    get { return _currentPrice; }
    set
    {
        if (value.Equals(_currentPrice)) return;
        _currentPrice = value;
        OnPropertyChanged("CurrentPrice");
```

```
            }
        }

        public double Commission { get; set; }

        public double Profit
        {
            get
            {
                return _profit;
            }
            set
            {
                if (value != _profit)
                {
                    _profit = value;
                    OnPropertyChanged("Profit");
                }
            }
        }

        public Order()
        {
            Ticket = -1;
            Size = 0.1;
            OpenTime = DateTime.MinValue;
            CloseTime = DateTime.MinValue;
            Symbol = string.Empty;
        }

        public event PropertyChangedEventHandler
PropertyChanged;

        protected internal void OnPropertyChanged(string
propertyName)
        {
            if (PropertyChanged != null) PropertyChanged(this,
new PropertyChangedEventArgs(propertyName));
        }
    }
```

Реалізація з’єднання з сервером та отримання даних

```
    private void ReceiveData()
    {
        while (!String.IsNullOrEmpty(_sessionId))
        {
            int length = 1024*1;
            var recBytes = new List<byte>();
            try
            {
                var rBytesAmount = 0;
                do
                {
                    var tmpBuffer = new byte[length];
                    rBytesAmount = _client.Receive(tmpBuffer);
                    recBytes.AddRange(tmpBuffer);
                } while (rBytesAmount == length);

                if (recBytes.Count == 0)
                    continue;
            }
            catch (Exception ex)
            {
                Debug.WriteLine("Data Feed ReceiveData: " + ex);
                _logger.Error("Data Feed ReceiveData: " + ex);
                if (_heartBeatThread != null)
                    _heartBeatThread.Abort();
                Reconnect();
                return;
            }

            var                         res                         =
Encoding.UTF8.GetString(recBytes.ToArray()).Split(';');
            foreach (var s in res)
            {
                if (String.IsNullOrEmpty(s))
                    continue;

                var  currentTick  =  s.Substring(1,  s.Length  -
1).Split(',');

                if (currentTick.Length != 3)
                    continue;

                double price;
                // Server return data in UTC format
                DateTime date;
```

```
                if              (!Double.TryParse(currentTick[1],
NumberStyles.Any, CultureInfo.InvariantCulture, out price) ||
                    !DateTime.TryParse(currentTick[2],
CultureInfo.InvariantCulture,   DateTimeStyles.AdjustToUniversal,
out date) ||
                    AllowedSymbols.All(symbol  =>  symbol.Name  !=
currentTick[0]))
                    continue;

                date         =         DateTime.SpecifyKind(date,
DateTimeKind.Utc);

                if (date > DateTime.UtcNow.AddDays(1))
                {
                    //RaiseDataFeedJournal(String.Format("Invalid
realtime quote detected (max limit): {0}", s));
                    continue;
                }

                if (date < DateTime.UtcNow.AddDays(-1))
                {
                    //RaiseDataFeedJournal(String.Format("Invalid
realtime quote detected (min limit): {0}", s));
                    continue;
                }

                RaiseOnQuote(new QuoteData
                {
                    Symbol = currentTick[0],
                    BidPrice = price,
                    Date = date,
                    Price = price,
                    AskPrice = price,
                    PipSize                                        =
WebClientHelper.SymbolsPipSize[currentTick[0]],
                    FeedName = Name // MS 140415 We should know
this
                });
            }
        }
    }
```