МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ЛЬВІВСЬКА ПОЛІТЕХНІКА»

На правах рукопису

Пастернак Ірина Ігорівна

УДК 004.777

Підвищення ефективності мережних інтерфейсів навігаційних сервісів кіберфізичних систем

05.13.05 - *комп'ютерні системи та компоненти*

Дисертація на здобуття наукового ступеня
кандидата технічних наук

Науковий керівник -
кандидат технічних наук,
доцент **Морозов Ю. В.**

Ідентичність всіх примірників дисертації
ЗАСВІДЧУЮ:

Учений секретар спеціалізованої вченої ради **/Я.Луцик/**

Львів-2016

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CORBA – архітектура посередника запитів на об’єкт (анг. Common Object Request Broker Architecture);

RMI – технологія віддаленого виклику методів (анг. Remote Method Invocation);

SOAP – простий протокол доступу до об’єктів (анг. Simple Object Access Protocol);

TCP – один з основних мережних протоколів Інтернету, призначений для управління передачею даних в мережах і підмережах у стеку протоколів TCP/IP (анг. Transmission Control Protocol);

IP – протокол мережного рівня для передачі данограм між мережами. (анг. Internet Protocol);

ОПЗ – об’єкт по значенню;

ПЗ – програмне забезпечення;

КФС- кіберфізична система;

НС – навігаційний сервіс.

ВСТУП

Актуальність теми. Конкуренція змушує виробників скорочувати термін розробки нових електронних виробів. Якщо раніше оптимальним вважався 3-4-ох річний час експлуатації електронного пристрою, то сьогодні він скоротився вже до 2-ох років і тенденція йде до скорочення цього терміну. Збільшення складності апаратних і програмних засобів навігаційних сервісів кіберфізичних систем та скорочення термінів експлуатації вимагає відповідних заходів для зменшення термінів розробки нових електронних пристроїв, кіберфізичних систем тощо. Навігаційний сервіс кіберфізичної системи складається з:

1) навігаційних пристроїв, які встановлені на рухомих об’єктах;
2) навігаційних сервісів, які забезпечують взаємодію пристроїв з користувачами;
3) комп’ютерних мереж, які об’єднують ці всі складові в одну систему.

Навігаційні пристрої постійно вдосконалюють, одні знімають з експлуатації, з’являються нові. Відповідно, швидка зміна типів навігаційних пристроїв вимагає побудови для них нових мережних інтерфейсів.

На даний час існує багато навігаційних пристроїв та сервісів і з часом їх стає все більше. Специфікою навігаційних сервісів є безкабельна передача даних від навігаційних пристроїв, які встановлені на рухомих об’єктах, в тому числі і автомобілях, навігаційним сервісам. Ці навігаційні засоби потрібно певним чином з’єднувати з безкабельними мережами, що здійснюється з допомогою відповідних мережних інтерфейсів. У навігаційних сервісах кіберфізичних систем існують мережні інтерфейси для трьох типів з’єднань:

1) з’єднання між навігаційними пристроями і навігаційним сервісом. Навігаційні пристрої швидко змінюються, одні знімаються з експлуатації, з’являються нові. Існують десятки типів цих пристроїв, які постійно оновлюються, а також вони не сумісні між собою. Відповідно швидка зміна типів навігаційних пристроїв вимагає постійно реалізовувати мережні інтерфейси для них. Протягом року з’являється біля десятка мережних інтерфейсів. Для того, щоб зменшити обсяг роботи програмістів над

реалізацією мережного інтерфейсу для з’єднання навігаційного пристрою з навігаційним сервісом під кожний новий тип навігаційного пристрою, потрібно уніфікувати мережний інтерфейс до різних типів пристроїв.

2) з’єднання між навігаційним сервісом і інтерфейсом користувача. Для різних користувачів потрібна різна інформація, яка формується з одних і тих самих навігаційних даних. Існує велика кількість видів навігаційної інформації і постійно з’являються нові види викликані потребами користувача. Це у свою чергу вимагає великої кількості різних форм представлення інформації. Інтерфейс користувача повинен відображати всі види навігаційної інформації у вигляді екранних форм, звітів тощо. Всі вони генерують потік запитів до навігаційного сервісу і інтерпретують відповіді навігаційного сервісу для свого оновлення. Відповідно завжди потрібно опрацьовувати розподілений потік запитів, який надходить від елементів інтерфейсу користувача до навігаційного сервісу, тому необхідно було б реалізувати один мережний інтерфейс, який реалізовуватиме цю взаємодію.

3) з’єднання між навігаційним сервісом і базою даних. Навігаційні задачі вимагають інформаційної підтримки на рівні БД. У користувачів постійно виникають нові навігаційні задачі. Відповідно до появи нових і зміни існуючих задач вміст БД змінюватиметься. БД може фізично знаходитися віддалено і навіть якщо вона знаходиться фізично на одній комп’ютерній системі з сервером програмно-апаратного навігаційного сервісу, вони взаємодіють між собою через мережний інтерфейс, тому що використовується класична клієнт-серверна взаємодія.

Вище наведені мережні інтерфейси відрізняються між собою протоколами зв’язку та даними, які через них передаються.

Для всіх вище наведених класів з’єднань, а саме: з’єднання між навігаційними пристроями і навігаційним сервісом, навігаційним сервісом і інтерфейсом користувача, навігаційним сервісом і базою даних спільною є проблема підвищенням ефективності цих мережних інтерфейсів шляхом покращенням критеріїв ефективності, а саме: час реакції та час розробки

мережного інтерфейсу.

Вище наведені проблеми, для всіх типів з'єднань, пропонується вирішити шляхом представлення структури мережних інтерфейсів у вигляді функціональних модулів, що розробляються на основі єдиного шаблону.

Таким чином, **актуальною є наукова задача** підвищення ефективності мережних інтерфейсів в навігаційних сервісах кіберфізичних систем.

Зв'язок роботи з науковими програмами, планами і темами. Дисертаційна робота виконана в межах наукового напряму кафедри електронних обчислювальних машин ДБ/КІБЕР № 015 - 97 «Питання теорії, проектування та реалізації комп'ютерних систем та мереж, а також комп'ютерних засобів, вузлів, приладів і пристроїв вимірювальних, інформаційних, керуючих, телекомунікаційних та кіберфізичних систем» (номер державної реєстрації 0115U000446).

Мета і завдання дослідження. Метою дисертаційної роботи є підвищення ефективності засобів та вдосконалення методів розробки мережних інтефейсів в навігаційних сервісах кіберфізичних систем.

Для досягнення цієї мети в роботі були розв'язані такі задачі :

- аналіз відомих методів та засобів забезпечення мережної взаємодії між навігаційними сервісами кіберфізичних систем та користувачами і навігаційними пристроями у безкабельній мережі;
- вдосконалення методу для поділу мережного інтерфейсу навігаційного сервісу кіберфізичних систем на компоненти;
- вдосконалення методів реалізації мережних інтерфейсів в процесі їх розробки для мережної взаємодії між навігаційними сервісами кіберфізичних систем та користувачами і навігаційними пристроями у безкабельній мережі;
- розроблення засобів реалізації мережних інтерфейсів навігаційного сервісу кіберфізичних систем;
- практична реалізація розроблених мережних інтерфейсів навігаційних сервісів кіберфізичних систем;

- порівняння з точки зору ефективності запропонованих рішень та відомих підходів для реалізації мережних інтерфейсів навігаційного сервісу кіберфізичних систем.

Об’єкт дослідження: взаємодія навігаційних сервісів кіберфізичних систем в безкабельних мережах з навігаційними пристроями та в мережі Інтернет з користувачами.

Предмет дослідження: мережні інтерфейси навігаційного сервісу кіберфізичних систем.

Методи дослідження. Методи досліджень базуються на принципах системного аналізу (ієрархічності, декомпозиції та інше). Для розв’язання поставлених у дисертаційній роботі задач використано методи об’єктно-орієнтованого програмування, теорії ймовірностей та обчислювальної математики, теорії комп’ютерних систем і мереж, які дали можливість визначити характеристики розроблених засобів мережних інтерфейсів, моделювання роботи безкабельних мереж, розробити критерії поділу мережних інтерфейсів.

Наукова новизна одержаних результатів полягає в наступному. На основі виконаних досліджень розв’язано науково-практичну задачу підвищення ефективності мережних інтерфейсів в навігаційних сервісах кіберфізичних систем. При цьому отримані такі наукові результати:

1. Вперше запропоновано метод розробки мережного інтерфейсу для навігаційних сервісів кіберфізичних систем шляхом параметризації системи команд та даних, за якою для кожного класу задач визначають підходи для розробки мережного інтерфейсу та його специфікації, що дає можливість зменшити час реакції мережного інтерфейсу на запити від клієнтів.

2. Вдосконалено існуючі методи динамічної, статичної та статично-динамічної мережної взаємодії в навігаційних сервісах кіберфізичних систем шляхом оптимізації функції мережного інтерфейсу, що дає можливість зменшити час розробки мережних інтерфейсів навігаційних сервісів кіберфізичних систем.

3. Вперше запропоновано спосіб порівняння розроблених і традиційних засобів реалізації мережних інтерфейсів навігаційного сервісу кіберфізичних систем, який дав можливість на етапі проектування навігаційних сервісів вибрати один із вдосконалених методів для реалізації мережних інтерфейсів.

Практичне значення одержаних результатів полягає в наступному:

- розроблений спосіб та проведене порівняння мережних інтерфейсів побудованих з застосуванням вдосконалених методів та традиційних підходів до побудови мережних інтерфейсів, які дали можливість скоротити у 2 рази час розробки мережних інтерфейсів навігаційного сервісу ZiTtrack;
- реалізовані методи розробки мережних інтерфейсів навігаційного сервісу кіберфізичних систем дали можливість на 19% покращити ефективність мережної взаємодії кіберфізичних систем для моніторингу рухомих об’єктів.

Використання результатів. Теоретичні і практичні результати дисертаційної роботи використано і впроваджено у:

1. Львівському державному університеті безпеки життєдіяльності, у аналітичній системі прогнозування надзвичайних ситуацій.
2. Комунальному підприємстві «Львівавтодор» у навігаційній системі диспетчеризації організації пасажирських перевезень у місті Львові.
3. Товаристві з обмеженою відповідальністю «Львівська пивна компанія» у навігаційній системі прокладання оптимальних маршрутів та контролю за транспортними засобами.

Особистий внесок здобувача. Основний зміст роботи, всі теоретичні та практичні результати, висновки і дослідження, які представлено до захисту, одержані автором особисто. Роботи [22, 23, 24, 25, 29, 30,31, 32] опубліковані самостійно. У публікаціях, написаних у співавторстві, автору належать: класифікація засобів модульної взаємодії між клієнтом і сервером [19]; модель об’єктної клієнт-серверної взаємодії [20]; огляд мережних інтерфейсів та їх взаємодія в мережі [21]; принципи розробки модульного

мережного інтерфейсу [25]; принцип розподілених обчислень [26]; навантажувальне тестування мережного інтерфейсу [27]; математичний опис мережного інтерфейсу [28]; принцип модульності мережного інтерфейсу [33]; опис мережного інтерфейсу та клієнт-серверної взаємодії в глобальній мережі [34].

Апробація результатів дисертації. Результати досліджень, викладені в дисертаційній роботі, доповідались та обговорювалися на наукових семінарах кафедри ЕОМ, а також на міжнародних конференціях: Міжнародна конференції ITS (м. Мінськ, Білорусь, 2012), IX Міжнародна конференція MEMSTECH (м. Поляна, 2013), VI Міжнародна конференція ACSN (м. Львів, 2013), VI Міжнародна конференція молодих вчених CSE (м. Львів, 2013).

Публікації. За результатами проведених досліджень опубліковано 16 наукових праць, в тому числі 6 статей у фахових наукових виданнях, затверджених Міністерством освіти і науки України, 1 стаття у виданні України, що входить до наукометричних баз.

Обсяг і структура дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, викладених на 127-ох сторінках друкованого тексту, списку використаних джерел (121 найменуваня) та додатків. Робота містить 42 рисунків, 6 таблиці та 5 додатків.

РОЗДІЛ 1

МІСЦЕ ТА ВАЖЛИВІСТЬ МЕРЕЖНОГО ІНТЕРФЕЙСУ У КІБЕРФІЗИЧНІЙ СИСТЕМІ

1.1 Дослідження особливостей кіберфізичних систем та їх основних складових

Дослідження в сфері кіберфізичних систем передбачають отримання нових наукових і технологічних досягнень, які забезпечать швидке та надійне проектування та інтеграцію кібер- (обчислювально-) і інфоцентрованних фізичних та інженерних систем.

Перед розробником кіберфізичної системи виникало завдання реалізації повного циклу проектування, починаючи від засобу розробки, функціонування і закінчуючи впровадженням цієї системи. Нові методи аналізу системного рівня і моделювання потрібні, для забезпечення передбачуваності і можливості компонування при декомпозиції функцій під час проектування. На проектування мало вплив на вартість, експлуатаційні характеристики і якість групи продуктів. Вибір архітектури проводиться до розробки та інтеграції кіберфізичної системи. У цьому процесі були визначені методи, функції для реалізації фізичної архітектури та зроблена оцінка якості та вибір найкращої апаратної платформи по відношенню до продуктивності, надійності і цінових показників [121].

На даний час сформована методологія проектування кіберфізичних систем, що була представлена так, як показано на рисунку 1.1.

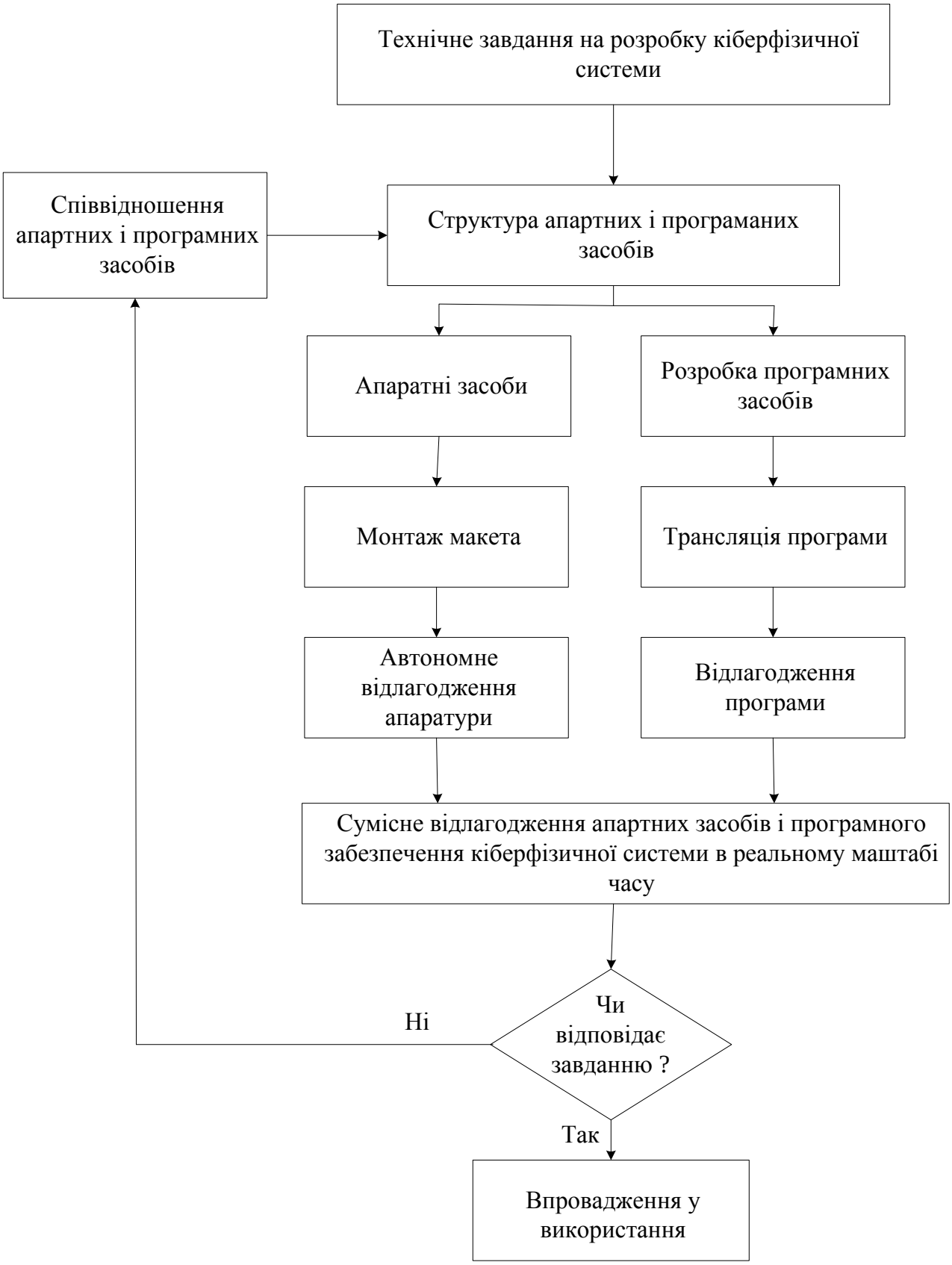


Рисунок 1.1 - Схема розробки кіберфізичної системи

У технічному завданні формулювалися вимоги до кіберфізичної системи, з погляду реалізації її певних функції. Це завдання містило у собі набір вимог, які визначали, що користувачу необхідно від кіберфізичної системи і, що повинні виконувати апаратні засоби. Відповідно до вимог користувача складалася функціональна специфікація, яка визначала функції, які реалізовувала кіберфізична система, для користувача після завершення проектування, уточнюючи тим самим, наскільки апаратні засоби відповідали вимогам. Кіберфізична система містила у собі описи форматів даних, як на вході та і на виході, а також зовнішні умови, що керувалися діями апаратури. Максимальне використання апаратних засобів спрощувало розробку і забезпечувало високу швидкодію кіберфізичної системи в цілому, але супроводжувалося збільшенням вартості і споживаної потужності [121].

Враховуючи високу вартість досліджень кіберфізичних систем, підготовки відповідних фахівців і придбання ліцензій для проектування на системному рівні, необхідним було використання набору методів та інструментальних засобів у всьому життєвому циклі кіберфізичної системи. Прикладом кіберфізичної системи був навігаційний сервіс, адже він містив в собі усі складові цієї системи.

Навігаційний сервіс кіберфізичної системи, складався з навігаційних пристроїв, які називаються трекерами, навігаційних сервісів кіберфізичних систем та віддалених інтерфейсів користувачів (Рисунок 1.2).

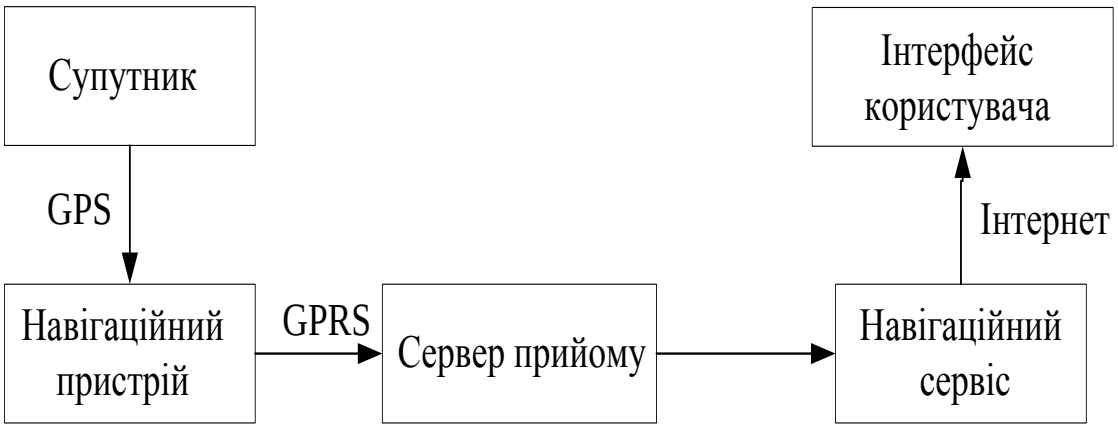


Рисунок 1.2 - Схема навігаційного сервісу кіберфізичних систем

Навігаційні сервіси (НС) кіберфізичних систем побудовані на базі системи глобального геопозиціонування GPS, часових вимірювань та вимірювань відстаней, що дозволяло визначати координати місцезнаходження рухомого об’єкта та його швидкість, обчислювати відстань й напрямок до пункту прибуття, час прибуття й відхилення від заданого курсу за допомогою приладу трекер. Цей сервіс замість геодезичних знаків і радіомаяків використовував супутники, що випромінювали спеціальні сигнали. Поточне місцезнаходження супутників, що знаходилися на високих орбітах [1]. Для визначення відстаней супутники й приймачі генерувалися складні двійкові кодові послідовності. Визначення часу розповсюдження сигналу відбувалося шляхом порівняння псевдокоду супутника по відношенню до такого ж коду приймача GPS. Кожний супутник мав свої власні два псевдовипадкових коди. Щоб розрізняти ці коди та інформаційні повідомлення різних супутників, в приймачі GPS відбувався виклик відповідних функцій. Супутники, постійно передавали інформацію про своє місцезнаходження. Відстань до них, визначалася шляхом вимірювання проміжку часу, який потрібний радіосигналу, щоб дійти від супутника до радіоприймача, та множенням його на швидкість розповсюдження електромагнітних хвиль. Точність вимірювання відстаней до супутників забезпечувалася шляхом синхронізації годинників супутників, в яких використовувалися атомні еталонні генератори частоти.

Функції НС кіберфізичних систем:

- Збирання великої кількості різнорідної інформації, яка обмежена тільки об'ємом власної пам'яті НС кіберфізичних систем.
- Передача даних НС кіберфізичних систем до комп'ютера диспетчера, здійснюється через бездротові мережі.
- Через використання НС кіберфізичних систем інформація про переміщення і стан об'єкта доступна диспетчеру в реальному часі (можлива затримка обумовлена способом комунікацій) [23].

- Час відгуку НС кіберфізичних систем залежав від каналу передачі даних (завжди була можливість підібрати компромісний варіант за співвідношенням ціна/оперативність).

Навігаційні засоби потрібно було, якимось чином з’єднувати через безкабельні мережі з серверами прийому, що здійснювалося за допомогою відповідних мережних інтерфейсів. Предметом дослідження в даній роботі є мережний інтерфейс (Рисунок 1.3).

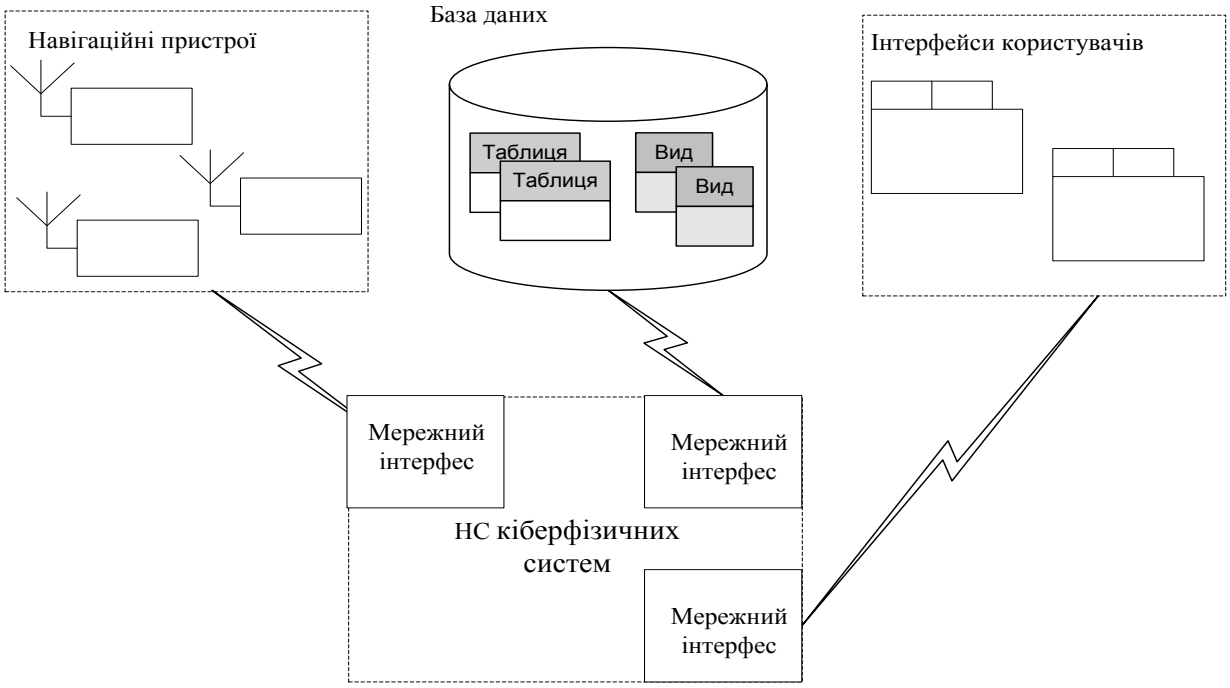


Рисунок 1.3 - Схема складових частин НС кіберфізичних систем

У навігаційних сервісах кіберфізичних систем існують мережні інтерфейси, для трьох класів з’єднань:

1) з’єднання між навігаційними пристроями і навігаційним сервісом. Навігаційні пристрої швидко змінювалися, одні знімалися з експлуатації, з’являлися нові. Існували десятки типів цих пристроїв, які постійно оновлювалися, а також вони були не сумісні між собою. Відповідно, швидка зміна типів навігаційних пристроїв, вимагала постійно реалізовувати мережні інтерфейси для кожного з них. Протягом року, з’являлося біля десятка мережних інтерфейсів. Для того, щоб зменшити обсяг роботи програмістів над реалізацією мережного інтерфейсу, для з’єднання навігаційного пристрою з

навігаційним сервісом, під кожний новий тип навігаційного пристрою, потрібно було уніфікувати мережний інтерфейс до різних типів цих пристроїв.

2) з'єднання між навігаційним сервісом і інтерфейсом користувача. Для різних користувачів, була потрібна різна інформація, яка формувалася з одних, і тих самих навігаційних даних. Існувала велика кількість видів навігаційної інформації і постійно з'являлиля нові види, викликані потребами користувача. Це у свою чергу, вимагало великої кількості різних форм представлення інформації. Інтерфейс користувача, повинен був відображати всі види навігаційної інформації у вигляді екранних форм, звітів тощо. Всі вони, генерували потік запитів до навігаційного сервісу і інтерпретували відповіді навігаційного сервісу, для свого оновлення. Відповідно, завжди потрібно було опрацьовувати розподілений потік запитів, який надходить від елементів інтерфейсу користувача до навігаційного сервісу, тому необхідно було реалізувати мережний інтерфейс, який виконував цю взаємодію.

3) з'єднання між навігаційним сервісом і базою даних. Навігаційні задачі вимагали інформаційної підтримки на рівні бази даних (БД). У користувачів постійно виникали нові навігаційні задачі. Відповідно, до появи нових і зміни існуючих задач, вміст БД змінювався. БД могла фізично знаходитися віддалено і навіть, якщо вона знаходилася фізично на одній комп'ютерній системі з сервером навігаційного сервісу кіберфізичних систем, вони взаємодіяли між собою через мережний інтерфейс, тому що використовувалася класична клієнт-серверна взаємодія. Ці мережні інтерфейси відрізнялися, між собою протоколами зв'язку та даними, які через них передавалися.

Для всіх вище наведених класів з'єднань спільним була проблема зменшення обсягу роботи над розробкою мережних інтерфейсів і підвищення їх ефективності, що було пов'язано зі скороченням терміну експлуатації. Вище наведені проблеми, пропонується вирішити шляхом декомпозиції мережних інтерфейсів на універсальну і спеціальну частини, таким чином цей інтерфейс стає модульним. Універсальна частина, одна для деякої кількості мережних інтерфейсів, а спеціальна частина унікальна, для кожного мережного

інтерфейсу і подібною до універсальної, адже розробляється на основі єдиного шаблону. Отже, завдяки цьому можна зменшити обсяг роботи і час на розробку.

Мережні інтерфейси для НС кіберфізичних систем, відіграють важливу роль. Адже за рахунок них НС кіберфізичних систем отримував можливість працювати з об'єктами БД, та мав доступ і можливість керування навігаційними пристроями, та взаємодіяти з інтерфейсами користувачів. Функції мережних інтерфейсів та їх роль у НС кіберфізичних систем розглянемо далі.

1.2 Взаємодія клієнта з сервером за допомогою мережного інтерфейсу

Програмна частина комп'ютерної мережі – це мережна операційна система, що забезпечувала роботу персонального комп'ютера, протоколи і прикладні програми, що підтримують роботу в цій мережі.

До апаратної частини комп'ютерної мережі відносили: персональні комп'ютери, кабелі, модеми, які забезпечували функціональність цієї мережі і формували її структуру.

Архітектура клієнт-сервер припускала розподіл праці в масштабах комп'ютерної мережі. Клієнтські системи, з якими мали справу користувачі, взаємодіють з серверами, що надавали формальний набір сервісів (комунікації, управління базами даних, глобальне іменування, картографія, тощо). Розподіл обов'язків в комп'ютерних мережах відбувався в основному за рахунок того, що функції орієнтовані на клієнта і виносилися в клієнтські системи (функціонуючі на ПК або навігаційних пристроях). Нижче наводиться декілька прикладів функцій, що орієнтовані на клієнта:

- створення і перевірка допустимості запитів до сервера;
- отримання інформації з сервера і інтерпретація результатів, для клієнта відповідно до його сприйняття;
- накопичення статистичної інформації роботи системи, що надається в цілому та представлення цієї інформації, для користувача відповідно, до деяких шаблонів або інших правил подання.

Серверна програма, надавала інформацію або інші послуги клієнтським програмам, які звертаються до неї [19]. Тенденції розвитку клієнт-серверних систем, приводять до переносу функціональності клієнта в спеціалізоване програмне забезпечення, яке також знаходиться на стороні сервера і завантажується на робочу станцію користувача [7,10].

Стандартне програмне забезпечення, що реалізується технологією клієнт - сервер має наступні характеристики: масштабованість, стійкість в роботі, захист від несанкціонованого доступу, продуктивність у роботі з великими проектами в галузі БД.

Клієнти і сервери є незалежними один від одного і також функціонують паралельно. Немає прив'язки клієнтів до серверів. Більш, ніж типовою була ситуація, коли один сервер одночасно обробляв запити від різних клієнтів; з іншого боку, клієнт звертається то до одного сервера, то до іншого. Клієнти знали про доступні сервери, але можуть не мають жодного уявлення про існування інших клієнтів.

Технології розробки клієнтського програмного забезпечення, базуються на використанні різних типів протоколів. Одним із них, є стек протоколів TCP/IP, який використовує сокети. Сокети, використовувані стеком протоколів TCP/IP, високостандартизировані і широкодоступні, але програмування з застосуванням сокетів розглядається програмістами, як занадто низькорівневе. Саме необхідність програмування на низькому рівні, перешкоджає продуктивному написанню розподілених додатків. Іншим типом протоколу є протокол RPC. Але протокол віддаленого виклику процедур (RPC), був досить складним у використанні, і до того ж існує велика кількість його різновидів. А також , мають велику популярність такі протоколи високого рівня, як CORBA, RMI і DCOM. Для організації їхньої роботи потрібно наявність спеціального середовища, як на стороні сервера так і на стороні клієнта. Їм притаманні також і інші недоліки. Наприклад, у процесі використання даних протоколів можливе виникнення проблем при проходженні пакетів з даними через брандмауер (системи мережного захисту).

Це протокол передачі гіпертекстових файлів HTTP (Hypertext Transfer Protocol) [3,4,5]. Саме через широке використання протоколу HTTP, компанії Microsoft і іншим виробникам мережного програмного забезпечення, їм довелося розробити новий протокол, що одержав назву SOAP. Для кодування запитів від об’єкту і даних у протоколі SOAP використовуються тексти мовою XML (Extensible Markup Language). Великою перевагою використання протоколу SOAP є його простота. Внаслідок своєї простоти, цей протокол реалізований на багатьох пристроях. Протокол SOAP (Simple Object Access Protocol) працює на верхньому рівні будь-якого стандартного протоколу.

1.3 Функції мережного інтерфейсу

Архітектура клієнт-сервер домінуюча концепція у створенні розподілених мережних додатків, яка передбачає взаємодію та обмін даними між ними [2,4,6]. Ця архітектура передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують інформацію, що надається серверами;
- мережні інтерфейси, які забезпечують взаємодію між клієнтами та серверами.

У комп'ютерних мережах мережним інтерфейсом називають:

1. Точку з'єднання між комп’ютером користувача і приватною чи громадською мережею;
2. Мережну карту комп’ютера (найбільш часте використання терміну);
3. Точку з’єднання комутованої телефонної мережі загального користування і телефону;
4. Точку з’єднання двох мереж між собою.

Для обміну даними між клієнтом і сервером, необхідний мережний інтерфейс. Мережний інтерфейс функціонує згідно деякого методу розробки клієнт-серверної взаємодії.

Метод розробки клієнт-серверної взаємодії, перш за все визначається розподілом обов’язків між клієнтською та серверною програмами (Рисунок 1.4).

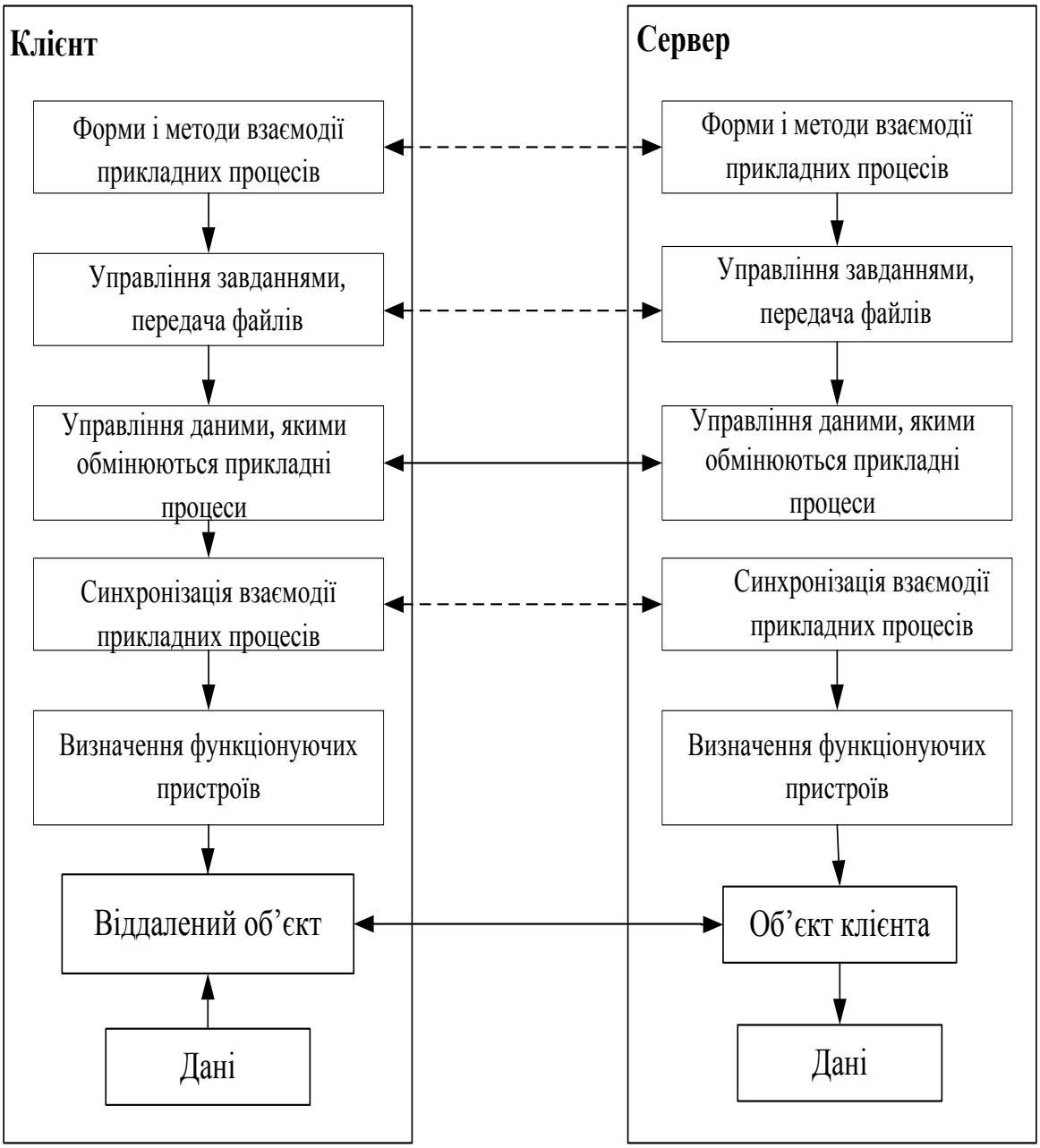


Рисунок 1.4 - Схема клієнт-серверної взаємодії

Структуру клієнт – серверної взаємодії описуємо наступним алгоритмом:

- користувач змінює стан, якогось об’єкту клієнтської програми, який в свою чергу формує і посилає запит до сервера, тобто до програми, яка обробляла запити;
- ця програма проводить відповідні зміни з даними, що знаходяться на сервері, у відповідності з запитом, міняє стан пов’язаного об’єкту і передає цю зміну клієнтській програмі;
- клієнтська програма отримує об’єкт, враховувала новий стан цього об’єкта і чекала подальших дій користувача. Цикл повторюється до того часу, поки користувач не завершить роботу з сервером.

Мережне програмне забезпечення складається з двох найважливіших компонентів:

- мережного програмного забезпечення встановленого на комп’ютерах-клієнтах;
- програмного забезпечення встановленого на серверах.

Основним завданням, що вирішувалося під час створення таких неоднорідних комп’ютерних мереж, було забезпечення сумісності обладнання за електричними та механічними характеристиками і забезпечення сумісності програм та даних, за системою кодування і форматуванням. Вирішенням цього завдання в мережі Інтернет є метод розробки взаємодії відкритих систем (OSI). Відповідно, до цього методу розробки, стандартизація апаратури і програмного забезпечення в мережі Інтернет, проводиться на підставі протоколів.

Стек протоколів TCP/IP вирішує проблему транспортування пакетів, у свою чергу їх вміст може бути різним (дані користувача, службова інформація). Пакети переміщуються від вузла до вузла, утворюючи нижній рівень функціонування мережі. Наповнення пакетів даними здійснюється програмами верхнього рівня, з якими працював клієнт. Саме ці програми, визначають для клієнта його можливості в мережі. Розроблялися вони на підставі одних правил (прикладних протоколів) і називаються мережними сервісами (службами, послугами).

Оглянувши основні проблеми, що постають при розробці мережного інтерфейсу типу клієнт-сервер, перейдемо до опису такого інтерфейсу через його функції. Мережний інтерфейс має стандартні і спеціальні функції, які показані на рисунку 1.5.

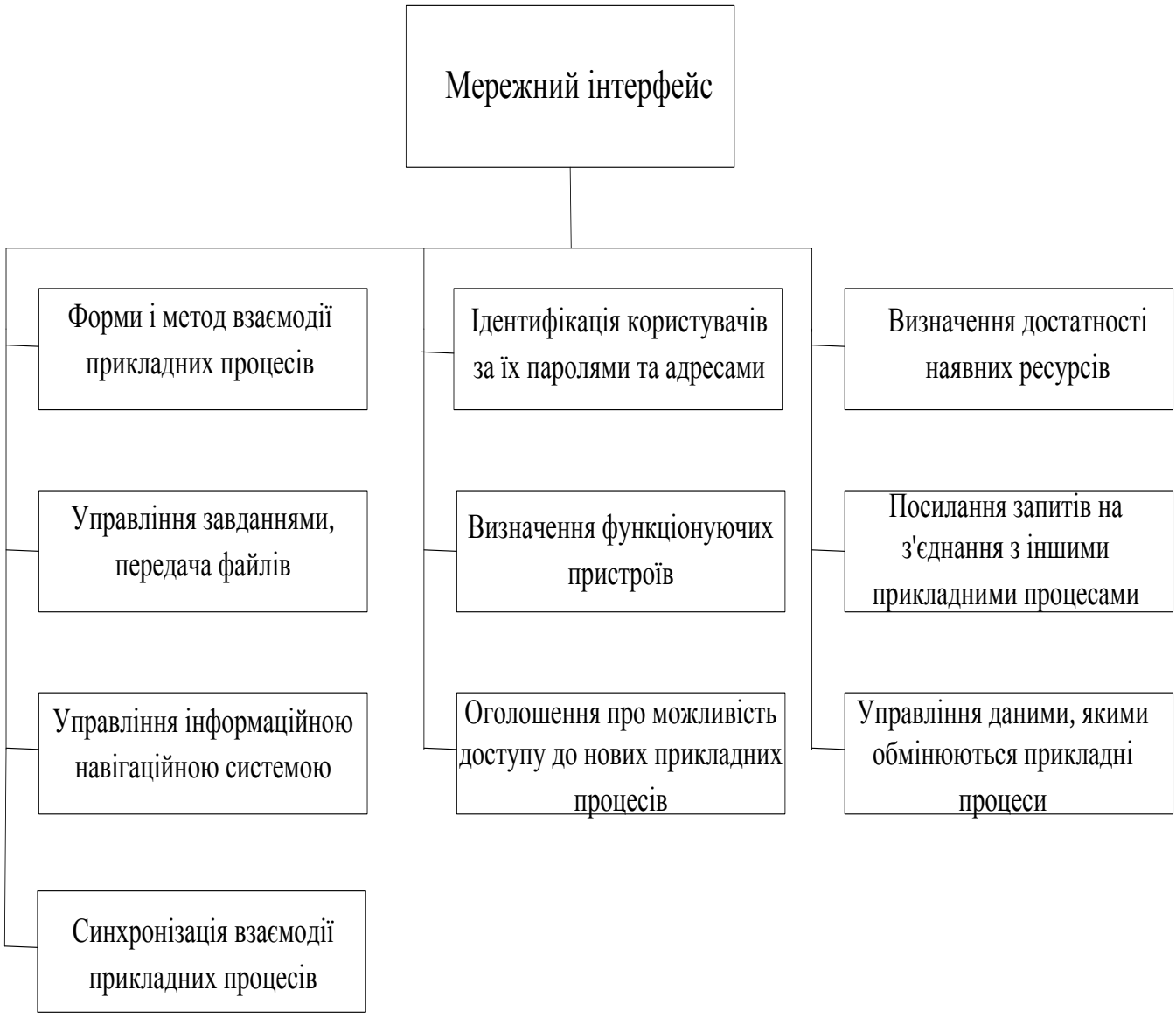


Рисунок 1.5 - Схема функцій мережного інтерфейсу

Аналіз цих функції показує, що деякі з них не залежать від реалізації і їх можна узагальнити, а інші залежать від реалізації їх не можливо узагальнити.

Виходячи з основних функцій опису мережного інтерфейсу, виділимо основні компоненти мережного інтерфейсу – його входи та виходи. Входом мережного інтерфейсу є множина запитів клієнтів, виходами - множина

відповідей на запити. У загальному випадку, реакція мережного інтерфейсу визначається не лише поточним запитом, а й усіма попередніми. Для усунення необхідності, кожен раз обробляти усі попередні запити клієнта, вводиться поняття стану мережного інтерфейсу. Стан мережного інтерфейсу є агрегованою історією запитів до даного інтерфейсу. У мережному інтерфейсі виділяються окремі рівні, тобто вона є багаторівневою і на кожному рівні міститься не менше однієї автономної підсистеми (Рисунок 1.6).

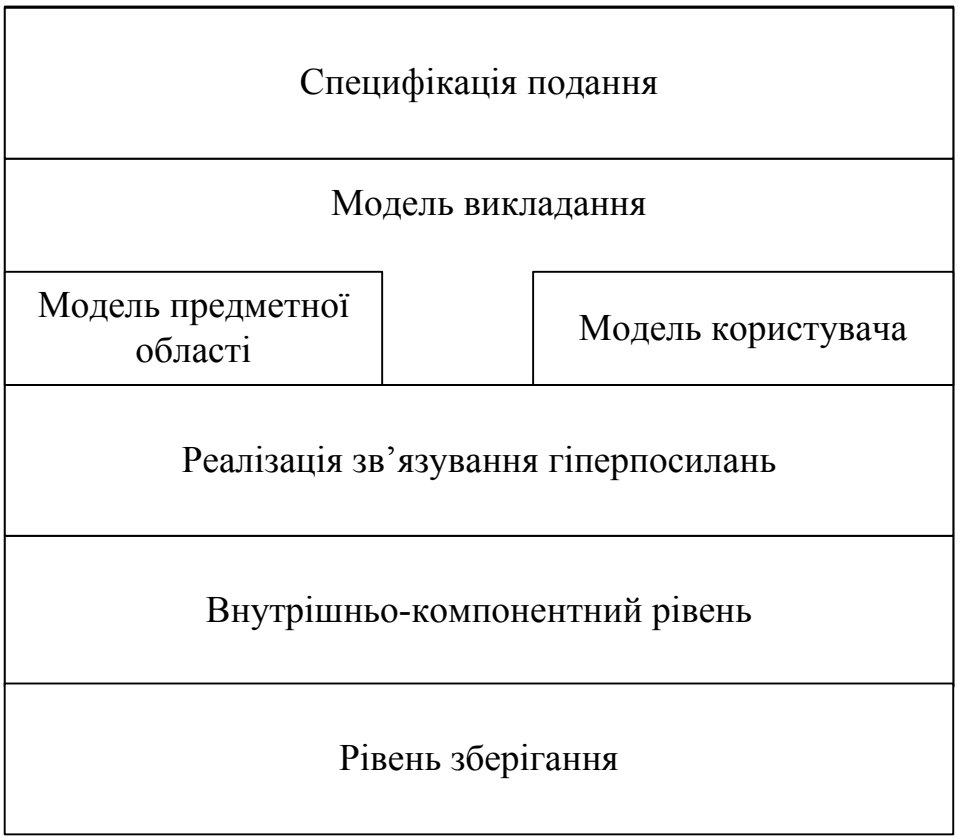


Рисунок 1.6 - Схема рівнів мережного інтерфейсу

Традиційно на кожному з рівнів (крім верхнього та нижнього), розглядалося три вхідних потоки, а саме: потік зовнішніх запитів, потік розпоряджень з вищого рівня – специфікація подання, потік реакцій нижчого рівня – модульний. Мережний інтерфейс, повинен був забезпечувати можливість ієрархічного з’єднання клієнта з сервером в мережі. Таке з’єднання базувалося на спрямуванні вихідного потоку нижчого рівня у вищий рівень

мережного інтерфейсу. Це з’єднання формуав вхідний сигнал, для вищого рівня мережного інтерфейсу [2,4,6]. У формуванні відповіді сервера на запит користувача з використанням мережного інтерфейсу, виділимо такі етапи:

1. Отримання запиту від користувача - рівень моделі викладання;
2. Формування запитів до нижчих рівнів – рівень специфікації подання;
3. Отримання відповідей від нижчих рівнів – внутрішньо-компонентний (модульний) рівень ;
4. Формування відповіді на запит – рівень зв’язування для гіперпосилань.

Аналізуючи ці етапи, відзначимо два моменти:

1) якщо розглядати тільки верхній рівень мережного інтерфейсу, то при його описі потрібно виділяти, як потік запиту користувача так і потік реакцій підкомпонент;

2) в мережному інтерфейсі в цілому (зі всіма рівнями включно), слід визначати лише один вхідний потік - запити користувачів (адже відповіді нижчого рівня мережного інтерфейсу визначаються запитами вищого НС кіберфізичних систем, а отже, вхідними запитами користувачів).

Деталізація мережного інтерфейсу проводилася до певного найнижчого рівня. Для функцій цього рівня розглядалися лише вхідні запити користувача [33]. Такий підхід, узгоджувався з твердженням, про не важливість опису вхідних потоків від нижчого рівня мережного інтерфейсу, для мережного інтерфейсу взагалі. Мережний інтерфейс з усіма його рівнями, розглядався, як мережний інтерфейс найнижчого рівня.

Аналіз поведінки мережного інтерфейсу, спрощувався завдяки неврахуванню параметрів з запитів користувача та його реакцій. Якщо досліджувати певні особливості функціонування мережного інтерфейсу, можна взагалі не враховувати або запит користувача, або вхідний потік від нижчого рівня мережного інтерфейсу [23]. Досліджуючи мережний інтерфейс в цілому, а не тільки його певний рівень, розглядалося лише вхідні запити користувачів.

При розгляді правил декомпозиції мережного інтерфейсу та його використання, внаслідок застосування певних операцій до його компонент, враховувалося, що запити до компонент формувалися мережним інтерфейсом.

Мережний інтерфейс описуємо такою множиною:

$$IS = (Q, R, A, St, \phi, \psi) \tag{1.1}$$

де Q - вхідні запити мережного інтерфейсу, R - відповіді нижчого рівня мережного інтерфейсу, A - вихідний алфавіт мережного інтерфейсу, St- множина станів мережного інтерфейсу, ϕ, ψ - функції переходів та виходів.

Розглянемо детальніше кожен із складових частин мережного інтерфейсу. Множину символів, що складають вхідний алфавіт мережного інтерфейсу, опишемо у такий спосіб:

$$Q = \{Qi\}, \tag{1.2}$$

$$Qi = \left\{Id_Q^{(i)}, Pq^{(1)}_1, \cdots, Pq^{(1)}_{N_Q^{(1)}}\right\}, \tag{1.3}$$

$$P_j^{(i)} \subset D^{(1)}_{Q_j^{(1)}} \times \cdots \times D^{(l)}_{Q_j^{(1)}}, l = N^{(j)}_{P_{Qi}}, \tag{1.4}$$

де множина IdQ - множина унікальних ідентифікаторів запитів.

Множину символів, що складають вихідний алфавіт мережного інтерфейсу, опишемо у такий спосіб:

$$A = \{Ai\} \tag{1.5}$$

$$Ai = \left\{Id_A^{(i)}, Pa^{(1)}_1, \cdots, Pa^{(1)}_{N_A^{(1)}}\right\} \tag{1.6}$$

$$P_j^{(i)} \subset D^{(1)}_{A_j^{(1)}} \times \cdots \times D^{(l)}_{A_j^{(1)}}, l = N^{(j)}_{P_{Ai}} \tag{1.7}$$

де IdA - множина унікальних ідентифікаторів відповідей.

Множину символів, що складають відповіді нижчого рівня мережного інтерфейсу, опишемо наступним чином:

$$R = \{Ri\} \tag{1.8}$$

$$Ri = \left\{Id_R^{(i)}, \mathrm{Pr}^{(1)}_1, \cdots, \mathrm{Pr}^{(1)}_{N_R^{(1)}}\right\} \tag{1.9}$$

$$P_j^{(i)} \subset D_{R_j^{(1)}}^{(1)} \times \cdots \times D_{R_j^{(1)}}^{(l)}, l = N_{P_{Ri}}^{(j)} \tag{1.10}$$

де IdR - множина унікальних ідентифікаторів відповідей нижчого рівня мережного інтерфейсу[34].

Кожен запит до мережного інтерфейсу та його відповідь, містить дві частини: ідентифікаційну та параметричну. Ідентифікаційна - однозначно визначає тип запиту. Параметрична частина є фактично додатковою інформацією, що супроводжує запити та відповіді на них мережним інтерфейсом. Опис стану мережного інтерфейсу, також базується на аналогічному підході. У загальному вигляді, конкретний стан St мережного інтерфейсу можна описати так:

$$St^{(i)} = \left\{ Id_{St}, St^{(i)}_{1}, \cdots, St^{(i)}_{N_{St}} \right\}, \tag{1.11}$$

мережний інтерфейс має складові,

або

$$St^{(i)} = St^{(i)} \subset D_{St}^{(1)} \times \cdots \times D_{St}^{(l)}, \tag{1.12}$$

мережний інтерфейс немає складових.

де $St^{(i)}_{1}, \cdots, St^{(i)}_{N_{St}}$ - стани компонент мережного інтерфейсу.

Таким чином, формальний опис стану мережного інтерфейсу має рекурсивний характер (до певного найнижчого рівня деталізації, на якому стани мають атомарний характер).

Кожен стан $St^{(i)}$ містить ідентифікатор стану Id_{St} та стани усіх компонент, що належать до мережного інтерфейсу. Кількість станів обмежується кількістю можливих станів компонент. Проте, насправді їх кількість істотно менша, внаслідок обмежень та взаємозв'язків, що можуть існувати між станами мережного інтерфейсу.

Реакція мережного інтерфейсу неповністю визначена у звичайному описі і у випадку параметричного опису [3,4,5]. Тобто, реакція мережного інтерфейсу

у НС кіберфізичних систем на кожен запит, описується множиною можливих переходів, виходів та їх ймовірностей.

Враховуючи формули 1.1, 1.11, 1.12 отримаємо функцію переходу, для мережного інтерфейсу:

$$\varphi(Q_i, St_j) = \left\{(St_k, \Pr\ (St_k, Q_i, St_j)\right\}, \tag{1.13}$$

$$\sum_k \Pr\ (St_k, Q_i, St_j) = 1, \tag{1.14}$$

Враховуючи формули 1.5, 1.15, 1.16 отримаємо функцію виходу, для мережного інтерфейсу:

$$\Psi(Q_i, St_j) = \left\{(A_k, \Pr\ (St_j, Q_i, A_k)\right\}, \tag{1.15}$$

$$\sum_k \Pr\ (St_j, Q_i, A_k) = 1, \tag{1.16}$$

Ось так, математично описується мережний інтерфейс.

Розглянемо проблему формування ймовірностей, для опису реакції мережного інтерфейсу на запит [20]. Можливі два підходи до її розв'язання:

1) визначення ймовірності конкретної реакції мережного інтерфейсу на запит користувача, із врахуванням значення конкретного параметра;
2) визначення загальної ймовірності конкретної реакції мережного інтерфейсу на запит користувача, без врахування значення конкретного параметра.

Перший підхід, забезпечує можливість визначенити реакцію мережного інтерфейсу на запит користувача. Проте, недоліком такого підходу є складність у визначенні ймовірності переходу. Крім того, при використанні операції видалення параметра, потрібно перераховувати ймовірності переходів. Для проведення фізичної оптимізації, також доцільно використовувати загальні значення ймовірностей реакції мережного інтерфейсу. Тому, доцільніше використовувати другий підхід. У такому разі, ймовірність ефективної роботи мережного інтерфейсу, визначається для кожної пари (St,Q) без врахування значень параметрів запиту до мережного інтерфейсу та його бази даних [39].

Реакція мережного інтерфейсу є меншою, ніж у першому випадку, проте для видалення параметрів не потрібно додаткових перетворень. Якщо в цьому інтерфейсі, існують запити з такими параметрами, що суттєво впливають на його реакцію та є важливими для його оптимізації, тоді пару - параметр, запит можна перетворити в окремий запит.

Тестування мережного інтерфейсу необхідне у випадках, коли потрібно отримати відповіді на наступні питання:

- що відбудеться з сервером, при збільшенні числа користувачів;
- де межа масштабованості мережного інтерфейсу;
- яке серверне програмне забезпечення краще відповідає потребам мережного інтерфейсу, з точки зору продуктивності;
- яке устаткування необхідне, для комфортної роботи з мережним інтерфейсом;
- що є потенційним «вузьким місцем» мережного інтерфейсу із сторони продуктивності.

На перший погляд здається, що на перераховані вище питання, можна відповісти і без проведення комплексних досліджень продуктивності мережного інтерфейсу. Але, це твердження справедливо тільки для простих мережних інтерфейсів і навіть, для них справедливо не завжди.

Під час тестування мережного інтерфейсу тими або іншими способами відтворювалося навантаження від всіх джерел запитів до мережного інтерфейсу (користувачі, зовнішні джерела даних і так далі), а не лише звернення користувачів до певних функціональних модулів мережного інтерфейсу.

Фази процесу тестування мережного інтерфейсу:

1. Дослідження мережного інтерфейсу;
2. Підготовка проектної документації;
3. Створення сценарію тестування;
4. Проведення тестування;
5. Обробка результатів і підготовка звіту.

На етапі дослідження мережного інтерфейсу відбувалося ознайомлення з тестувальним комплексом на різних рівнях:

- рівень кінцевого користувача;
- рівень клієнта, що обслуговував мережний інтерфейс;
- прикладний рівень;
- системний рівень;
- апаратний (фізичний рівень).

Рівень кінцевого користувача мережного інтерфейсу, дає уявлення про функції і часові рамки відгуку на його запит. На цьому етапі, відбувалося визначення варіантів використання мережного інтерфейсу, спектр навантаження мережного інтерфейсу.

Рівень клієнта, що обслуговував мережний інтерфейс, необхідний для виділення функціональних частин системи. Визначалися зв’язки між частинами мережного інтерфейсу і проводилася їх оцінка.

Прикладний рівень, дає уявлення про внутрішнє влаштування системи з погляду розробників ПЗ. Цей рівень, використовувався для оцінки архітектурних рішень, які застосовувалися при створенні системи для тестування. На цьому етапі оцінювалася продуктивність і ресурси.

Системний рівень необхідний для оцінки якості розгортання мережного інтерфейсу і визначення точок установки моніторів системних ресурсів [36].

Аналіз апаратного рівня, дозволяє визначити апаратні засоби, як використовувалися для побудови комп’ютерної мережі.

Системний аналіз досліджуваного ПЗ для реалізації мережного інтерфейсу не можливий без проведення його тестування. Цей етап дає відповідь на питання: як проводилося тестування, які інструменти були потрібні, які зовнішні мережні інтерфейси повинні працювати, як здійснювався моніторинг продуктивності досліджуваного ПЗ.

На даному етапі, як правило, застосовувалося реверсивне проектування. В найпростішому випадку потрібно визначити, які запити – це просто навантаження даних з довідників, а які є реакцією аплікації на дії користувача і

вимагають параметризації. Коректна параметризація – умова створення навантаження на мережний інтерфейс, адекватного реальному.

Для мережного інтерфейсу, що працював з великими об’ємами даних, завдання проведеного випробування набуло особливого значення. Часто при роботі з мережним інтерфейсом, отримувалася краща продуктивність на даних невеликого об’єму, але із зростанням навантаження виникали різноманітні проблеми, які можуть бути пов’язані з не оптимальною побудовою запитів, технологічними конфліктами різних елементів мережного інтерфейсу, обмеженнями системних ресурсів, тощо.

Тестування мережного інтерфейсу з клієнт-серверною архітектурою було призначене для визначення максимальної кількості запитів від клієнта (пропускної здатності) та оптимальної конфігурації сервера для наступних станів мережного інтерфейсу:

- нормальний;
- навантажений;
- критичний.

У всіх випадках при тестуванні відбувалося визначення часу стабільного перебування у фіксованому стані і межа переходу у наступний. Зазначимо, що після перевищення межі критичного навантаження відбувався збій мережного інтерфейсу різного ступеня складності: помірної (прості запити від клієнтів), максимальної і мінімальної [19].

Більшість розвинених інструментальних засобів тестування мережного інтерфейсу, наприклад, Apache, JMeter, Mercury LoadRunner, WAPT дозволили організувати випробувальний стенд з розподіленим виконанням тестів і централізованим управлінням (Рисунок 1.7).

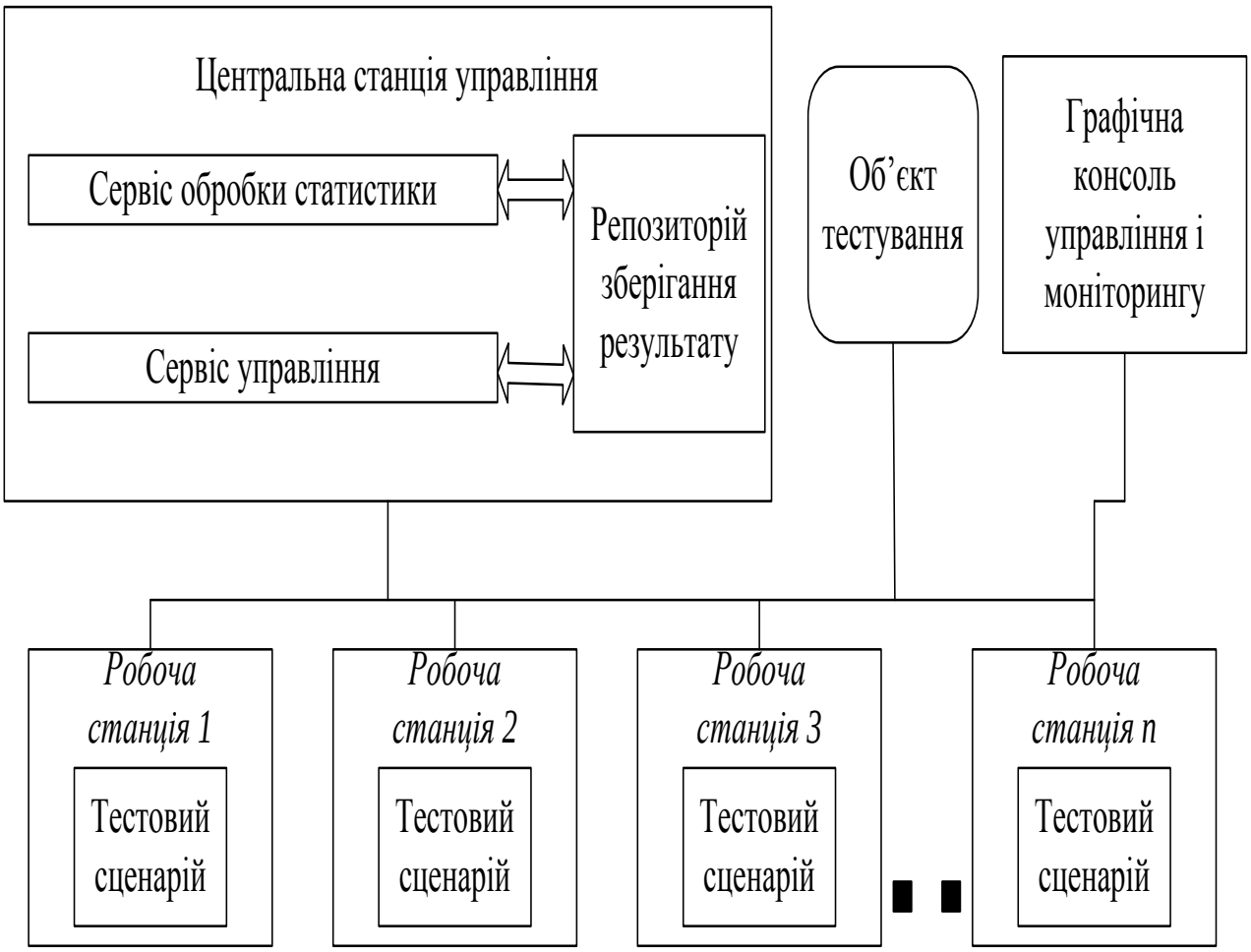


Рисунок 1.7 - Схема тестування мережного інтерфейсу

Структура такого стенду містила:

1. Об'єкт випробувань.
2. Центральну станцію управління, де розташовувалися НС кіберфізичних систем, репозиторій результатів тестування і графічна консоль користувача.
3. Декілька робочих станцій моделювання навантаження, на яких розташовавалися компоненти для імітації діяльності користувачів.
4. Додаткові станції моніторингу і генерації тестових даних, для забезпечення безперебійного постачання тестовими даними робочих станцій. Також, на окремих станціях запускалися вимірювальні монітори, що здійснювали спостереження за процесами обробки і вони ж передавали виміряну статистику на центральну станцію.

1.4 Взаємодія мережних інтерфейсів

Класифікації функцій мережного інтерфейсу на основі взаємодії між клієнтом і сервером наведена в таблиці 1.1 [8].

Таблиця 1.1 Класифікація функцій мережного інтерфейсу

Активація режиму	Стан віддаленого об’єкта	Основа протоколів	Форматування
сервер-активований об’єкт клієнт-активований об’єкт	за значенням за посиланням	HTTP-канал TCP-канал	Текстовий формат двійковий формат

1.4.1 Активація режиму через сервер-активований і клієнт-активований об’єкти

Класифікація функцій мережного інтерфейсу на основі взаємодії між клієнтом і сервером наступна:

1. Сервер-активований об’єкт: об’єкти існують у просторі з адресою сервера, клієнт отримує доступ до цих об’єктів за посиланням через ВЕБ-сервер (Рисунок 1.8).

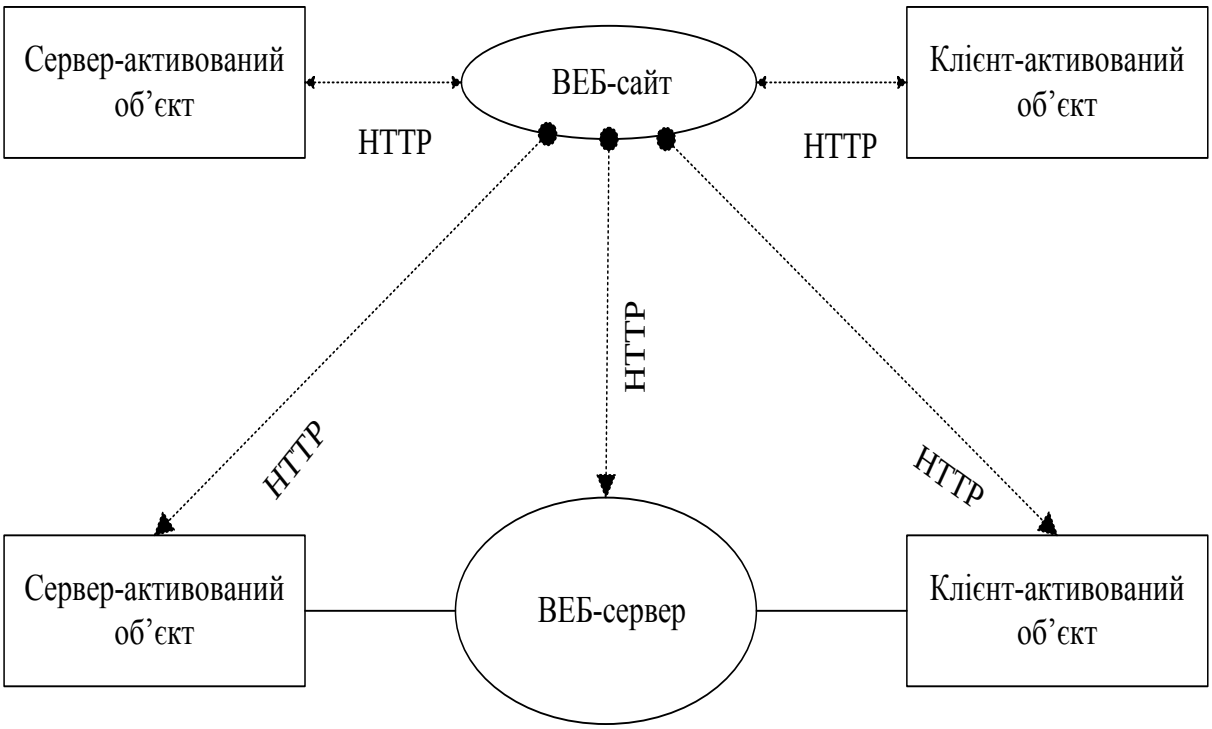


Рисунок 1.8 - Схема взаємодії клієнта з сервером через сервер-активований та клієнт-активований об’єкт

Для того, щоб клієнт отримав доступ до об’єкту необхідно активувати сервер. Існує два режим для активації сервера: виклик (Single Call), тон (Single Ton). Виклик відбувався кожен раз, при запиті клієнта на об’єкт. При цьому створювався новий екземпляр об’єкту, для даного запиту. Всі запити клієнта були послідовними і в тому ж порядку обслуговувалися екземпляри об’єкту, навіть якщо сервером ще не був виконаний попередній екземпляр. Сервер-активований об’єкт використовував тон, тільки при одному екземплярі об’єкту у будь-який момент часу. Якщо екземпляр об’єкту не існує, сервер створював нові екземпляри об’єктів, щоб обслуговувати всі подальші запити клієнта.

2. Клієнт-активований об’єкт - це віддалені об’єкти, які активувалися за запитом клієнта. При активації клієнта, ним створювався екземпляр віддаленого об’єкта і встановлювався двосторонній зв’язок, між клієнтом і сервером. «Клієнт» - користувач, який налагоджує свій зв'язок з сервером через клієнт-активований об’єкт. Для того, щоб була сформована відповідь на запит клієнта, сервер використовував свою базу даних та віддалені об’єкти.

Коли клієнт посилав запит на віддалений об’єкт, віддалений додаток отримував повідомлення про його активацію. Потім сервер, створював екземпляр об’єкту відповідного класу, на який прийшов запит і повертав посилання на об’єкт, щоб клієнтський додаток зміг його використовувати.

1.4.2 Використання віддаленого об’єкту в залежності від його типу

Стан віддаленого об’єкту (Remote Object), застосовувався для того, щоб використати об’єкт клієнта для доступу до функціональності об’єкта сервера через процеси [9,11]. При цьому об’єкт сервера був віддаленими. Існує два типи віддалених об’єктів:

1. За значенням. Використовувався тоді, коли об’єкт сервера, отриманий у вигляді значення, тоді сервер використовував серіалізацію. Потім сервер, кодував об’єкт в послідовність бітів і передавав його клієнту, який декодував цю послідовність.

2. За посиланням. У цьому типі сервера об’єктів, сервер відправляв посилання на об’єкт клієнту. Об’єкт клієнта використовував це посилання, для виклику методів об’єкта сервера.

1.4.3 Протоколи для організації мережної взаємодії та переваги їх використання

Протоколи надавали канали, по яких повідомленнями могли обмінюватися віддалені об’єкти між собою через домени додатків. Передача об’єкта сервера на одному кінці і об’єкта клієнта на іншому. Об’єкт сервера вказував протокол і номер порта для отримання запиту. Об’єкт клієнта відправляв запит на об’єкт сервера, за допомогою зазначеного протоколу та номера порта. При цьому використовувалося два канали протоколів:

1. HTTP-канал. Використовувався HTTP-канал, якщо потрібно було встановити з’єднання клієнта з сервером через глобальну мережу Інтернет так, як брандмауери не перешкоджають зв’язку створеному протоколом HTTP.

2. TCP-канал. TCP-канал використовувався, якщо необхідно було обмінюватися даними між клієнтом і сервером всередині мережі. Якщо

використовувався TCP-канал, тоді можна було відкрити певні порти в брандмауерах між клієнтом і сервером [10].

Прикладом протоколу зв’язку є Remote Object. Протокол Remote Object - це найшвидший спосіб зв’язку для передачі даних. Цей протокол використовував безпосередньо отримані дані. Remote Object, як компонент, використовував HTTP-канал для передачі бінарних даних в мережі. На рисунку 1.9 показана архітектура Remote Object.

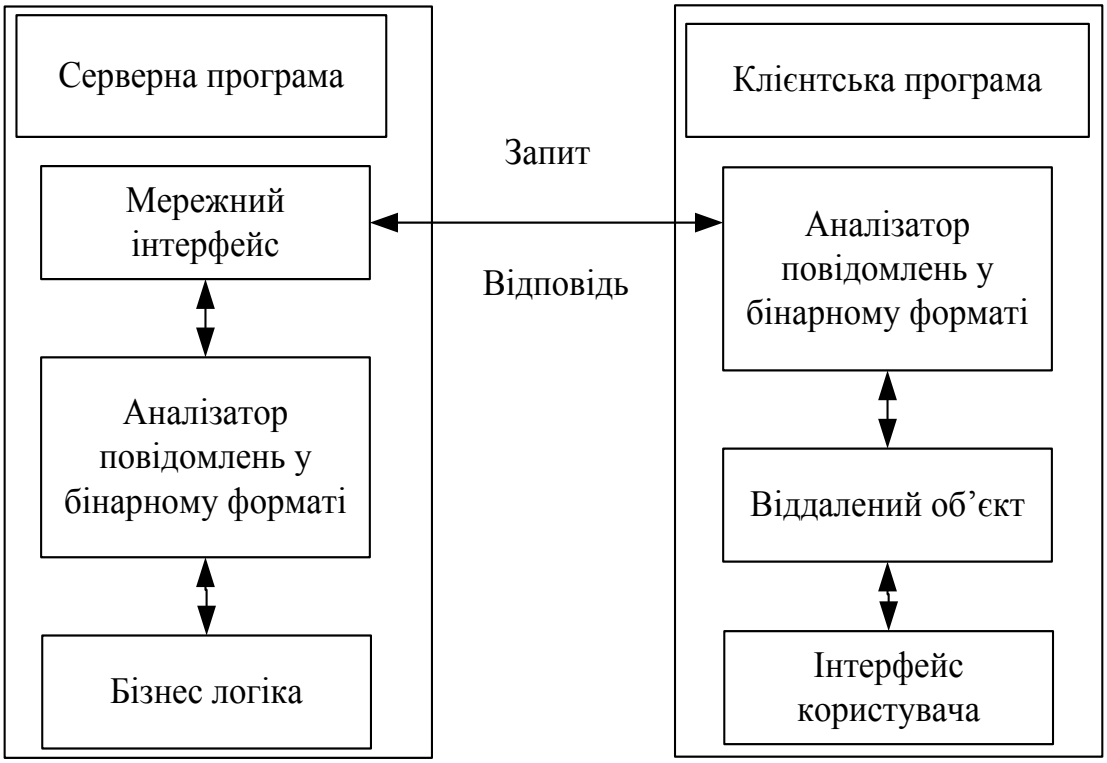


Рисунок 1.9 - Схема архітектури Remote Object

Серверна програма надавала деякі послуги клієнтським програмам. Зв'язок між клієнтом і сервером зазвичай здійснювався за допомогою передачі повідомлень на основі моделі «запит-відповідь», часто через мережу, і при цьому використовувався протокол Remote Object для кодування запитів клієнта і відповідей сервера. Серверні програми були встановлені, як на серверному, так і на клієнтському комп'ютері, щоразу вони забезпечували виконання певних служб (наприклад, сервер баз даних чи ВЕБ-сервер).

Перевагами використання протоколу Remote Object було: 1) він значно швидший засіб зв'язку, ніж будь-які інші засоби зв'язку; 2) можна було надсилати та отримувати суворо типізовані об'єкти із сервера. При реалізації протоколу Remote Object використовувалися серіалізація та десеріалізація, які відбуваються на стороні сервера, який отримував можливість через мережний інтерфейс відправляти прості та складні об'єкти клієнту. Стани віддалених об'єктів були прив'язані до серверного об'єкта, шляхом додавання метаданих, тобто деяких тегів до класу об'єкта. Налаштування комунікацій згідно протоколу Remote Object виявилося складніше, ніж створення зв'язку клієнта з http-сервісом або ВЕБ-сервером. Протокол Remote Object у використанні з мережним інтерфейсом працював більш ефективніше, тому що він створював менше навантаження даними в двійковому форматі в результаті меншого розміру пакетів даних.

1.4.4 Форматування даних у клієнт-серверній взаємодії

Необхідно було перетворити дані у відповідному форматі, перш ніж передати їх по каналу. Дані форматувалися об'єктами, тобто кодувалися і серіалізувалися у відповідний формат. Форматування було реалізоване у мережному інтерфейсі до передачі даних. Види форматування даних:

1. Текстовий формат. Цей протокол є модульним і не залежить від будь-якого конкретного транспортного протоколу, наприклад, HTTP або TCP.

2. Двійковий формат. Цей формат представлення даних використовує повідомлення для передачі інформації. Двійковий формат простий і більш ефективніший, ніж текстовий формат [12]. Однак, тільки додатки можуть прочитати двійковий формат.

1.5 Об'єктна взаємодія мережних інтерфейсів

Модульні об'єктні середовища вважаються, природнім фундаментом для накопичення та використання знань з програмування. Ці середовища

базуються на модульному об’єктному методі [22]. Вони включають в себе готові компоненти, а також інструментальні засоби, які дозволяють вибрати компоненти та налаштувати їх. Найчастіше розподілені об’єкти (компоненти) використовуються в технології клієнт-сервер [21]. Самі сервери є об’єктами, які реагують на запити і надають клієнту послуги або ресурси.

Було досліджено модульні середовища тому, можна сказати, що вони володіють всіма перевагами притаманними об’єктно-орієнтованим технологіям та будуються на основі методу об’єктної взаємодії, основними принципами якої є:

1. Інкапсуляція об’єктних компонентів приховує складність реалізації компонентів роблячи їх доступними у вигляді загальнодоступного інтерфейсу.
2. Наслідування дозволяє розвивати створені раніше компоненти, при цьому не порушуючи цілісність об’єктної частини.
3. Поліморфізм дає можливість групувати компоненти і їх характеристики, які є схожими.

Кожний з цих принципів не новий, але в методі об’єктної взаємодії вони використовуються в сукупності. Ці принципи детальніше буде розглянуто в 3 і 4 розділах дисертаційної роботи. А у цьому розділі розглянемо метод об’єктної взаємодії клієнта з сервером.

Метод об’єктної взаємодії - це найпростіший спосіб зв’язку для передачі станів об’єктів (Рисунок 1.10).

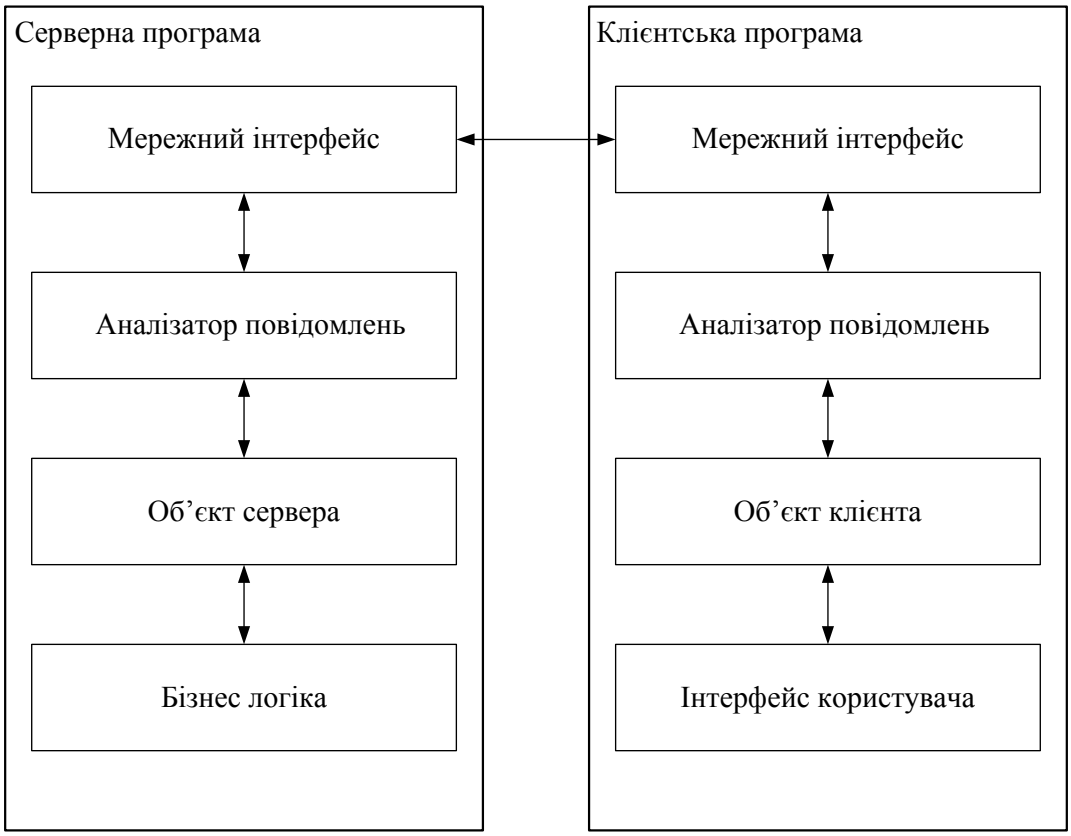


Рисунок 1.10 - Схема об’єктної взаємодії

Представлений метод об’єктної взаємодії використовує протокол HTTP для передачі стану віддаленого об’єкту поданого у бінарному форматі по каналу зв’язку. Клієнт отримує можливість відправки простих і складних об’єктів на сервер. Ці клієнтські об’єкти можуть бути прив’язані до об’єкта сервера, шляхом додавання метаданих, тобто тегів, до класу об’єкта. Налаштування комунікацій в методі об’єктної взаємодії трохи складніше, ніж створення зв’язку клієнта з ВЕБ-сервісом [47]. Метод об’єктної взаємодії у використанні, показав себе ефективнішим ніж інші, тому що навантаження каналу зв’язку даними в двійковому форматі менше в результаті меншого розміру пакета [32].

При дослідженні методу об’єктної взаємодії можна виділити віддалений об’єкт по значенню (ОПЗ). Код методів такого об’єкта виконувався локально. Якщо ОПЗ був отриманий з іншої мережі, то необхідний код повинен або бути наперед відомий обом сторонам, або бути динамічно завантажений. Щоб це

було можливо, запис, що визначає ОПЗ, містить спеціальне поле зі списком адрес, звідки може бути завантажений код. В ОПЗ можуть також бути і віддалені методи. В ОПЗ можуть бути поля, які передаються разом з самим ОПЗ. Ці поля також можуть бути ОПЗ, які формують списки, дерева або довільні графи. ОПЗ можуть мати ієрархію класів, включаючи множинне наслідування і абстрактні класи.

Мережні інтерфейси взаємодіяли один з одним в одній комп'ютерній системі та по різних мережах. Вони обмінювалися даними один з одним, незалежно від їх конкретної реалізації в розподіленій системі, платформи і мови їх реалізації. Семантика об'єктів на цьому рівні не бралася до уваги. Було встановлено завдання мережних інтерфейсів, а саме інтегрувати розподілені системи, що дало можливість програмам, які написані різними мовами та працювати у різних вузлах мережі, взаємодіяти одна з одною так само просто, наче вони знаходилися в адресному просторі одного процесу.

Об'єктна взаємодія надає список функцій, в якому можна виділити програмні модулі, а також надає список служб, які можуть використовувати ці модулі [48,49]. Серед них є служби повідомлень, авторизації, доступу до сховищ і управління транзакціями, що використовуються в НС кіберфізичних систем. Вони найчастіше використовуються розподіленими додатками. Розділяючи реалізацію цих функцій на модулі можна значно понизити складність реалізації мережного інтерфейсу.

Метод об'єктної взаємодії в НС кіберфізичних систем складалася з серверної та клієнтської частин. До неї можна застосувати рівневий підхід (Рисунок 1.11).

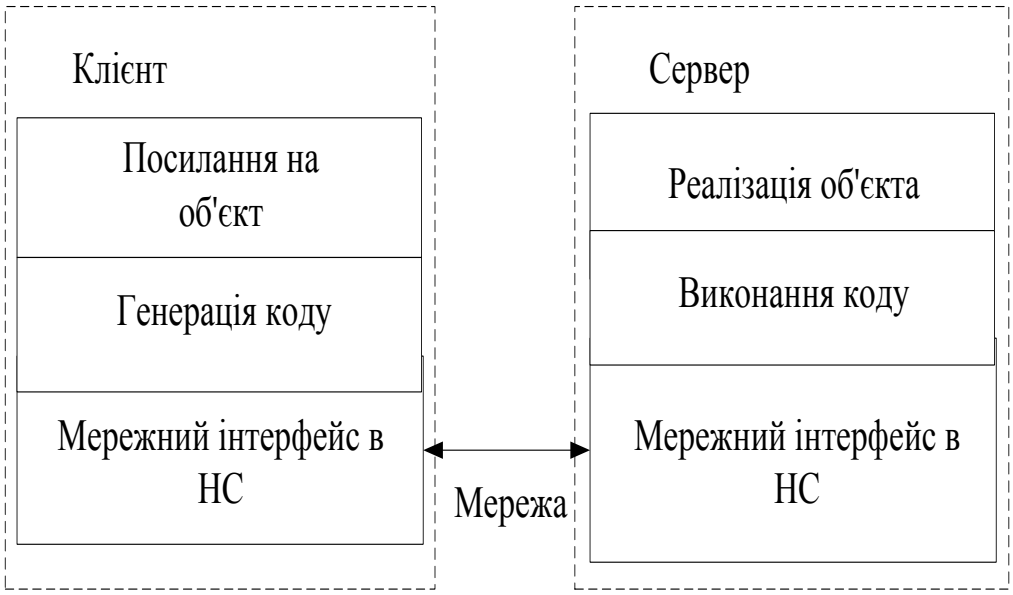


Рисунок 1.11 - Схема об’єктної взаємодії в НС

Кожен рівень клієнтської частини обслуговувався відповідним рівнем серверної частини, при взаємодії через мережу: посилання на об’єкт – реалізація об’єкта, генерація коду - його виконання. З цих частин складався мережний інтерфейс в НС кіберфізичних систем через, який здійснювалася взаємодія між клієнтом і сервером у мережі. При взаємодії цих частин в мережі, за посиланням на об’єкт в клієнтській частині завантажувалася реалізація об’єкта з серверної частини.

1.6 Мережний інтерфейс навігаційного сервісу кіберфізичних систем

Мережний інтерфейс НС кіберфізичних систем розглядався, як параметризований мережний інтерфейс НС кіберфізичних систем, де універсальна частина не параметризувалася, а параметризувалася тільки спеціальна частина. Крім того, для кожного типу клієнтського об’єкта була потрібна специфічна спеціальна частина, а серверна спеціальна частина повинна була містити реалізацію для всіх клієнтських запитів, що не відміняє параметризацію серверного мережного інтерфейсу, а саме його спеціальної частини. Тобто, чи використовувати клієнт-активований об’єкт чи сервер-активований об’єкт, чи в одному потоці керування оброблялися запити, чи

створювався для кожного запиту свій потік керування, тобто для сервер-активований об’єкт це не мало значення, мережний інтерфейс повинен був бути параметризованим. Будь-який мережний інтерфейс, параметризувався типом клієнта, тобто його структурою даних і командами. Цю параметризацію здійснювалося трьома методами. Статично-динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем, як вже це розглянуто вище, а саме розробка на основі принципу наслідування базового класу спеціальної частини мережного інтерфейсу класом спеціалізованої частини для конкретного типу клієнта. Статичний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем використовувався тоді, коли мережний інтерфейс використовує об’єкт спеціальної частини мережного інтерфейсу, тобто реалізувався принцип використання. Динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем базувався на динамічній параметризації на основі шаблонів, тобто реалізувався принцип інстанціювання.

Вдосконалення методів підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем, що були вище згадані, можна провести шляхом реалізації мережного інтерфейсу взявши за основу 3-и принципи, а саме: наслідування, використання та інстанціювання. Розглянемо ці принципи на основі відповідних методів.

1.7 Ефективність мережного інтерфейсу

Ефективність - сукупність властивостей, які визначають ступінь пристосування мережного інтерфейсу до виконання поставлених перед ним завдання. Із трьох мережних інтерфейсів ефективнішим вважався той, який краще відповідає своєму призначенню. Ефективність може бути технічною, економічною, оперативною і т.д. Технічна ефективність - міра пристосування мережного інтерфейсу до виконання експлуатаційного завдання, обумовлена її технічними характеристиками. Якщо поняття надійності використовувався для

оцінки технічного стану складових частин мережного інтерфейсу, то поняття ефективності для оцінки очікуваних або отриманих результатів застосування мережного інтерфейсу [42,55]. Ефективність і надійність – це поняття, які складаються з багатьох окремих взаємопов'язаних властивостей, для модульного мережного інтерфейсу.

Підвищення надійності мережного інтерфейсу не самоціль, а лише один із засобів забезпечення високого рівня ефективності. Надійність мережного інтерфейсу в значній мірі визначає загальні економічні показники її роботи. Між надійністю і технічною ефективністю існує безпосередній зв'язок: чим надійніший об'єкт, тим вища його технічна ефективність. Тому зміна технічної ефективності може бути мірою апаратної надійності, і навпаки [20,23].

Одночасно із створенням технічної бази НС кіберфізичних систем, звертають увагу надають економічності організації виробництва, оскільки впровадження навіть досконалого мережного інтерфейсу у виробництво з поганою організацією може знизити її ефективність.

При виборі складових частин інформаційного забезпечення враховують, що залежність "ефективність – затрати", суттєво залежить від часу між розробкою та використанням цього забезпечення. Це в свою чергу спричиняє виникнення завдання пошуку оптимального об'єму інформації і моменту його отримання, що реалізувалося при розробці інформаційного забезпечення.

Суттєве значення має залежність "ефективність – затрати" технічних засобів, оскільки у вартості створення мережного інтерфейсу, вони займають основний об'єм. Одночасно з вибором відповідної модульного мережного інтерфейсу та типу засобів необхідно вирішувати завдання пошуку їх оптимального об'єму та відповідних цьому об'єму затрат, що можна назвати вибором оптимального рівня автоматизації.

Для кожного конкретного мережного інтерфейсу, мета створення якого полягає в забезпеченні найбільш повного використання можливостей об'єкта управління для вирішення поставлених перед ним завдань, ефективність модульного мережного інтерфейсу визначають порівнянням результатів його

функціонування і затрат всіх видів ресурсів, необхідних для його створення і розвитку [44,45].

1.8 Висновки до розділу

1. Аналіз існуючих принципів покращення ефективності мережного інтерфейсу у навігаційних сервісах кіберфізичних систем дозволяє здійснити вибір базових способів для реалізації мережних інтерфейсів в даних системах. Аналіз показав, що для забезпечення ефективного функціонування навігаційних сервісів кіберфізичних систем необхідне використання засобів формування реакції, які здатні адаптуватися в умовах швидкої зміни структури мережного інтерфейсу.
2. Показано, що існують проблеми щодо створення адаптивних засобів формування реакції мережного інтерфейсу у навігаційних сервісах кіберфізичних систем.
3. Огляд існуючих функцій мережного інтерфейсу у навігаційних сервісах кіберфізичних систем вимагає значних зусиль для реалізації мережного інтерфейсу, що викликає значні витрати часу і низьку ефективність. Проведений аналіз показав необхідність вдосконалення існуючих методів та розроблення нових засобів реалізації мережного інтерфейсу, які б покращила вище згадані показники.
4. Функціональною основою для структурного синтезу мережного інтерфейсу є класифікація його функцій. Особливу увагу звернуто на ефективність її практичного застосування.
5. Огляд літературних джерел обґрунтовує актуальність поставленої задачі.

РОЗДІЛ 2

МОДЕЛЬ ПАРАМЕТРИЗОВАНОГО МЕРЕЖНОГО ІНТЕРФЕЙСУ НАВІГАЦІЙНОГО СЕРВІСУ КІБЕРФІЗИЧНИХ СИСТЕМ

2.1 Модульний мережний інтерфейс навігаційного сервісу кіберфізичних систем

При проектуванні та дослідженні будь-якого виробу, у тому числі й алгоритму, на ранніх стадіях важлива увага приділялася найголовнішій проблемі, і спеціально не беруться до уваги окремі деталі. Тому найбільш загальна підхід у програмування, полягає в розкладанні процесу на окремі дії [23,52]. На кожному такому кроці декомпозиції потрібно впевнитися, що:

- розв’язання окремих завдань призводять до виконання повставленої задачі;
- здійснена декомпозиція дозволяє отримати інструкції, які можна реалізувати ефективніше.

Реалізація принципу структурного програмування здійснювалося з використанням макрокоманд і механізмів виклику підпрограм. Ці ж методи підходять для реалізації програм у модульному програмуванні, яке можна розглядати, як частину структурного підходу.

Необхідно розрізняти використання слова модуль, коли йдеться про одиницю дроблення великої програми на окремі блоки (які можуть бути реалізовані у вигляді процедур і функцій) і коли малося на увазі синтаксична конструкція мов програмування [54].

Важливе значення в концепції модульного програмування надавалося організації керуючих та інформаційних зв'язків між модулями програми, що спільно розв'язують одну або декілька великих задач [56].

При роботі з модулями потрібно пам'ятати їх основну відмінність від процедур і функцій. Традиційні правила сфери дії глобальних і локальних змінних для модулів не працюють. Ця мовна конструкція розроблена так, щоб виключити вплив глобальних змінних, оголошених у головній програмі, на внутрішній опис модуля.

Принцип модульності застосовувався по відношенню до мережного інтерфейсу навігаційного сервісу кіберфізичних систем. Мережний інтерфейс НС кіберфізичних систем розділявся на дві частини: універсальну і спеціальну, таким чином він ставав модульним (Рисунок 2.1).

Рисунок 2.1- Схема модульного мережного інтерфейсу НС кіберфізичних систем

На рис. 2.1 наведені дві частини мережного інтерфейсу НС кіберфізичних систем, а саме спеціальна і універсальна з певним набором функцій, які містилися у кожній із них. Функції універсальної частини були однакові, для

всіх типів мережного інтерфейсу НС кіберфізичних систем, а функції спеціальної частини були спеціалізованими для різних типів мережних інтерфейсів. В універсальну частину виносився уніфікований код, який для різних об’єктів однаковим, тобто він розроблявся і тестувався один раз. А відмінності мережних інтерфейсів НС кіберфізичних систем переносилися в спеціальну частину, яку необхідно розроблялося для кожного мережного інтерфейсу окремо.

На базі вище наведеного поділу мережного інтерфейсу НС кіберфізичних систем на функції, можна реалізувати модель модульної клієнт-серверної взаємодії на основі цього типу інтерфейсу (Рисунок 2.2).

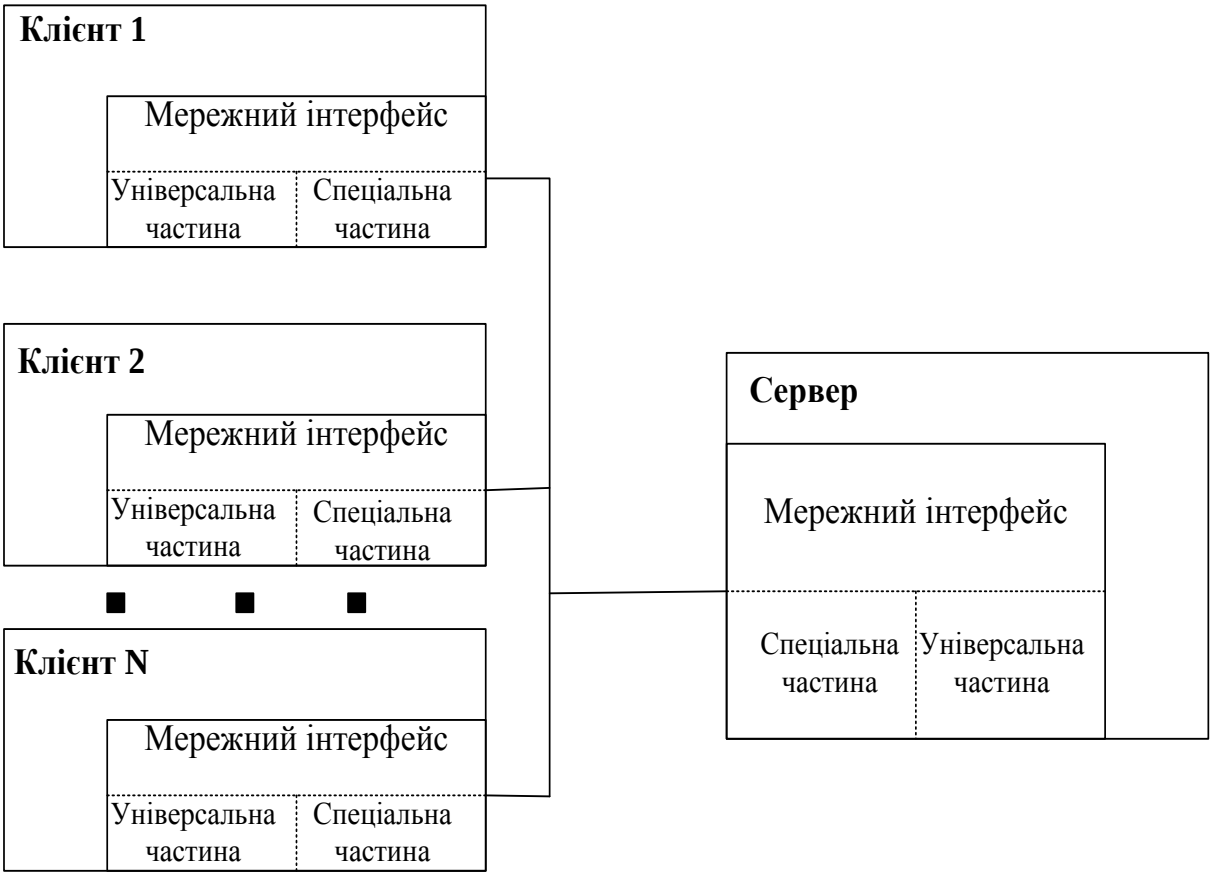


Рисунок 2.2 - Модель модульної взаємодії клієнта з сервером

Модель модульної взаємодії клієнта з сервером складалася з клієнтських та серверних мережних інтерфейсів і асинхронна взаємодія між ними відбувалася через мережу. Тобто, клієнти посилали запити, а сервер

приймав запит і для кожного із запитів створював окремий потік керування. В цьому потоці, сервер відпрацьовував відповідь на запит і відсилав її клієнту. Запити від клієнтів надходили до сервера асинхронно, тобто запити від клієнтів могли надходити одночасно і послідовно. Для того, щоб асинхронна взаємодія між клієнтом і сервером працювала коректно, сервер повинен був постійно очікувати з'єднання з клієнтом, а коли воно починалося то, для цього з'єднання він створював потік керування і відпрацьовувати його. Проблема виникала в наступному, клієнт створював з'єднання для якого на сервері утворювався потік керування. Той самий клієнт по тому ж з'єднанню, не закриваючи його, посилав наступний запит в тому ж потоці керування. Якщо запити могли надходити достатньо часто, наприклад у випадку навігаційних пристроїв, то вони могли створити ситуацію, коли ще нема відповіді а є наступний запит. У цьому випадку, сервер повинен був «вміти» зберігати запити, для того, щоб він міг коли розвантажиться обробити той, що вже збережений запит і наступні, тобто створити чергу запитів. До черги запитів відповідно утворювалася черга відповідей тому, що є час відправки відповіді. Таким чином, відбувалася буферизація. В НС кіберфізичних систем існує багато подібних, але різних мережних інтерфейсів. В такому випадку, сервер при отриманні запита повинен був визначати від якого типу клієнта прийшов запит і виконувати відповідну обробку. Тобто вибір сервера, у свою чергу спричиняє вибір правила обробки для даного потоку керування. В універсальній частині мережного інтерфейсу, при цьому, нічого не змінювалося, тобто сервер нічого не вибирав і не аналізував. В спеціальній частині мережного інтерфейсу, сервер повинен був вибрати правило обробки для потоку керування, відповідно до типу клієнта.

На основі такої загальної моделі модульної взаємодії клієнта з сервером, кожен тип клієнта містив свій набір правил обробки потоку керування, які мали входити в спеціальну частину мережного інтерфейсу клієнта. А серверний мережний інтерфейс систем містив всі типи відповідних функцій всіх типів клієнтів, які були згруповані по типах клієнтів для того, щоб їх можна було

використовувати груповим чином. Відповідно, відмінність між клієнтським і серверним мережними інтерфейсами була в тому, що клієнтський мережний інтерфейс знав тільки один набір правил обробки, а серверний мережний інтерфейс повинен був знати всі набори клієнтських правил обробки і містити диспетчер типу клієнта. Тоді, у відповідь на запит клієнта на сервері створювався потік керування з функціями обробки для конкретного типу клієнта. Ця диспетчеризація виконувалася за допомогою трьох принципів:

1. Наслідуванням загальних функцій мережного інтерфейсу спеціальними функціями клієнтів.
2. Використанням мережним інтерфейсом спеціальних функцій клієнта.
3. Інстанціюванням спеціальними функціями клієнта шаблону взаємодії клієнта з сервером.

Відповідно було отримано три вдосконалених методи розробки модульного мережного інтерфейсу НС кіберфізичних систем, а саме: розробка модульного мережного інтерфейсу на основі принципу наслідування, розробка модульного мережного інтерфейсу на основі принципу використання та розробка модульного мережного інтерфейсу на основі принципу інстанціювання [17].

Модульний підхід до розробки цих методів, полягав в наступному наборі основних принципів:

- Все є об’єктами.
    - Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об’єктами, при якій один об'єкт потребував, щоб інший об’єкт виконував деяку дію. Об'єкти взаємодіяли, надсилаючи і отримуючи повідомлення.

Кожен із запропонованих вдосконалених методів розробки мережних інтерфейсів НС кіберфізичних систем, а саме на основі принципів наслідування, використання та інстанціювання, складалися з наступних складових:

1. **Клієнти** (пристрої), яких була довільна кількість.

3. **Мережний інтерфейс клієнта**, їх кількість залежала від кількості типів клієнтів.

4. **Протоколи**, через які здійснювалася клієнт-серверна взаємодія.

5. **Сервер**, який надавав послуги клієнтам у відповідь на запити від клієнтів, після того як клієнти створювали з ним з'єднання.

6. **Мережний інтерфейс сервера**, очікував з'єднань від клієнтів.

У 4 розділі розглянуто, більш детальніше кожен із запропонованих вдосконалених методів розробки мережних інтерфейсів НС кіберфізичних систем.

2.2 Параметризований мережний інтерфейс навігаційного сервісу кіберфізичних систем

Використання віддалених об'єктів в мережі, при взаємодії клієнта з сервером дуже складне, особливо якщо їх потрібно використовувати повторно. Необхідно підібрати відповідні віддалені об'єкти, віднести їх до різних класів та зберегти, для цього було використано метод розробки мережних інтерейсів. Цей метод полягає у наступному. Коли клієнт посилав запит на віддалений об'єкт на сервері, доступ до нього він отримував через мережний інтерфейс НС кіберфізичних систем. Мережний інтерфейс НС кіберфізичних систем у свою чергу було параметризовано, щоб спростити взаємодію клієнта з сервером через віддалені об'єкти, через велику кількість віддалених об'єктів. А саме, через заміну одного параметра при використанні параметризованого мережного інтерфейсу НС кіберфізичних систем, можна було вирішувати різні поставленні завдання, які потрібно було застосовувати при створенні з'єднання клієнта з сервером в глобальній мережі [23,63].

В іншому випадку, необхідно було для кожного віддаленого об'єкта створити окремий мережний інтерфейс НС кіберфізичних систем. Для параметризації мережного інтерфейсу НС кіберфізичних систем були

використані шаблони. На рисунку 2.3 клас взаємодії і форматування інстанціює шаблони, які переводить у певний формат.

Рисунок 2.3 - Модель параметризованого мережного інтерфейсу НС кіберфізичних систем (КФС)

Для ефективної клієнт-серверної взаємодії в глобальній мережі було використано модель параметризованого мережного інтерфейсу НС кіберфізичних систем на базі, якої пропонується метод розробки мережних інтерфейсів. У цій моделі, мережний інтерфейс параметризувався за допомогою класів з використанням шаблонів (Рисунок 2.3).

При використанні моделі параметризованого мережного інтерфейсу НС кіберфізичних систем важливо знати, що шаблони відрізнялися від класів степенем деталізації і рівнем абстракції і повинні були бути, якимось чином організовані у параметризований мережний інтерфейс. Для того, щоб відрізнити шаблони від класів, класифікуємо їх за двома критеріям (таблиця 2.1). Перший критерій - призначення шаблону. У зв'язку з цим виділялося структурні шаблони і шаблони поведінки. Перші були пов'язані з процесом

створення об’єктів. Другі мали відношення до композиції об’єктів та класів [64,71]. Шаблони поведінки характеризувалися так, як класи чи об’єкти взаємодіють між собою.

Таблиця 2.1. Класифікація шаблонів

Мета\Рівень	Структурні шаблони	Шаблони поведінки
Клас	Адаптер класу	Інтерпретатор
		Шаблонний метод
Віддалений об’єкт	Адаптер об’єкта	Ітератор
	Декоратор	Команда
	Замісник	Спостерігач
	Компоновщик	Відвідувач
	Міст	Посередник
	Пристосованець	Стан
	Фасад	Стратегія
		Зберігач
		Ланцюжок зобов’язань

Другий критерій – рівень, показав те, для чого застосовуються шаблони: до віддалених об’єктів чи до класів. Шаблони рівня класів описували відношення між класами і їх підкласами. Такі відношення відображалися за допомогою наслідування, тому вони були статичні, тобто зафіксовані на етапі компіляції. Шаблони рівня об’єктів описували відношення між об’єктами, котрі могли змінюватися під час виконання і тому більш динамічні. Практично усі шаблони в якийсь мірі використовувалися у принципі наслідування.

Шаблони класів частково відповідади за створення віддалених об’єктів своїми підкласам, у свою чергу породжені шаблони віддалених об’єктів

передавали відповідальність іншому віддаленому об’єкту. Структурні шаблони класів використовували наслідування для вкладеності класів, в той час як структурні шаблони віддалених об’єктів описували способи збору віддалених об’єктів з частин. Поведінкові шаблони класів використовували наслідування для опису алгоритмів і потоку управління, а поведінкові шаблони віддалених об’єктів, як об’єкти, які належать деякій групі, разом функціонували і виконували завдання, які не під силу ні одному окремому віддаленому об’єкту [65,70].

При використанні шаблонів у ПЗ вирішувалися конкретні задачі, внаслідок чого об’єктно-орієнтований вигляд параметризованого мережного інтерфейсу НС кіберфізичних систем став більш простим, ефективним, і його можна було застосовувати повторно.

За допомогою моделі параметризації мережного інтерфейсу НС кіберфізичних систем при клієнт-серверній взаємодії, клієнт з сервером міг обмінюватися віддаленими об’єктами в глобальній мережі. Розглянемо кроки, необхідні, щоб дозволити клієнту отримати доступ до об’єкта на віддаленому сервері:

- Створити TCP або HTTP-з’єднання між клієнтом і сервером через параметризований мережний інтерфейс.
- Вибрати повідомлення, які відправлені від клієнта до сервера та від форматувати їх.
- Зареєструвати типи, які будуть доступні у віддаленому режимі.
- Створити віддалений об’єкт та активувати його з сервера або клієнта.

Клієнт вказуав на об’єкт, який він хоче отримати маючи доступ до сервера і використовувати клієнт-активований об’єкт або сервер-активований об’єкт. А також клієнт вибирав метод реєстрації і передавав кілька параметрів мережного інтерфейсу НС кіберфізичних систем, які визначали адресу сервера і тип (клас), який доступний. Сервер реєстрував типи з’єднання і порти, що він зробить доступними для клієнтів. Коли клієнт намагався викликати метод віддаленого об’єкту, його виклик проходив через кілька шарів на стороні

сервера. Перший з шарів мав вигляд абстрактного класу, який мав такий же параметризований мережний інтерфейс НС кіберфізичних систем, як віддалений об’єкт, який він представляв. Це підтверджувалося тим, що кількість і тип аргументів у виклику вірні, тоді формувася запит в повідомленні, і його передавалося клієнту через канал зв’язку. У свою чергу, канал зв’язку відповідав за запит на віддалений об’єкт. Канал зв’язку складався з сервер-активованого об’єкта, що серіалізував запит в потік передачі і клієнт-активованого об’єкта [8,19, 62].

Сервер-активований об’єкт, з іншого боку, вказував, які об’єкти він хоче зробити доступними для віддалених клієнт-активованих об’єктів. Це реалізовувалося за допомогою механізму, відомого як тип реєстрації. Процес реєстрації відбувався на сервері і клієнті. У своїй найпростішої формі, реєстрація на сервері полягала у створенні екземпляра об’єкту каналу зв’язку та його реєстрації шляхом передачі, як параметра статичного методу відповідного класу [63,72].

2.3 Математичний опис параметризованого мережного інтерфейсу навігаційного сервісу кіберфізичних систем

В багатьох об’єктно-орієнтованих НС кіберфізичних систем зустрічаються шаблони, які повторюються і які складаються з класів і віддалених об’єктів. З врахованням шаблонів було параметризовано мережний інтерфейс.

Запопонований метод розробки мережного інтефейсу НС кіберфізичних систем, шляхом використання принципу параметризації мережного інтерфейсу для коректної розробки цього типу інтерфейсу. Параметризація мережного інтерфейсу НС кіберфізичних систем виконувалася за таким параметрами, як: вихідний алфавіт, відповіді нижчого рівня, множина станів. Параметризований мережний інтерфейс НС кіберфізичних систем, можна описати у вигляді формул за цими параметрами, наступним чином [30].

Параметризований мережний інтерфейс НС кіберфізичних систем опишемо таким впорядкованим набором:

$$IP_{П} = (H_{П}, B_{П}, L, Sp, \lambda, f), \quad (2.1)$$

де $_{П}$ - параметри, які будуть відповідати за коректний запит та відповідь мережного інтерфейсу і завдяки ним параметризувався цей вид інтерфейсу НС кіберфізичних систем.

$H_{П}$ - вхідні запити,

$B_{П}$ - відповіді на запити клієнта,

L - вихідний алфавіт,

Sp - множина станів,

λ, f - функції переходів та виходів.

Множину символів, що складають вхідний алфавіт параметризованого мережного інтерфейсу НС кіберфізичних систем, опишемо у так:

$$H_{П} = \{Hi\}, \quad (2.2)$$

$$Hi = \left\{Id^{(i)}_{H_{П}}, Ph^{(1)}_{1}, \cdots, Ph^{(1)}_{N^{(1)}_{H}}\right\}, \quad (2.3)$$

$$P^{(i)}_{j} \subset D^{(1)}_{H^{(1)}_{j}} \times \cdots \times D^{(l)}_{H^{(1)}_{j}}, l = N^{(j)}_{P_{Hi}}, \quad (2.4)$$

де множина $Id^{(i)}_{H_{П}}$ - множина унікальних ідентифікаторів запитів.

Множина символів, що складає вихідний алфавіт параметризованого мережного інтерфейсу НС кіберфізичних систем, матиме вигляд:

$$B_{П} = \{Bi\}, \quad (2.5)$$

$$Bi = \left\{Id^{(i)}_{B_{П}}, Pb^{(1)}_{1}, \cdots, Pb^{(1)}_{N^{(1)}_{B}}\right\}, \quad (2.6)$$

$$P^{(i)}_{j} \subset D^{(1)}_{B^{(1)}_{j}} \times \cdots \times D^{(l)}_{B^{(1)}_{j}}, l = N^{(j)}_{P_{Bi}}, \quad (2.7)$$

де $Id_{B_{II}}^{(i)}$ - множина унікальних ідентифікаторів відповідей.

Множина символів, що складають відповіді параметризованого мережного інтерфейсу НС кіберфізичних систем на запит клієнта, опишемо так:

$$L = \{Li\}, \tag{2.8}$$

$$Li = \left\{Id_L^{(i)}, Pl^{(1)}{}_1, \cdots, Pl_{N_L^{(1)}}^{(1)}\right\}, \tag{2.9}$$

$$P_j^{(i)} \subset D_{L_j^{(1)}}^{(1)} \times \cdots \times D_{L_j^{(1)}}^{(l)}, l = N_{P_{Li}}^{(j)}, \tag{2.10}$$

де $Id_L^{(i)}$ - множина унікальних ідентифікаторів відповідей параметризованого мережного інтерфейсу НС кіберфізичних систем на запит клієнта[57].

Конкретний стан Sp параметризованого мережного інтерфейсу НС кіберфізичних систем, якщо параметризований мережний інтерфейс НС кіберфізичних систем містить складові, можна описати так:

$$Sp^{(i)} = \left\{Id_{Sp}, Sp^{(i)}{}_1, \cdots, Sp_{N_{Sp}}^{(i)}\right\}, \tag{2.11}$$

Якщо параметризований мережний інтерфейс НС кіберфізичних систем не містить складових:

$$Sp^{(i)} = Sp^{(i)} \subset D_{Sp}^{(1)} \times \cdots \times D_{Sp}^{(l)}, \tag{2.12}$$

де $Sp^{(i)}{}_1, \cdots, Sp_{N_{Sp}}^{(i)}$ - стани компонент параметризованого мережного інтерфейсу НС кіберфізичних систем. Кожен стан $Sp^{(i)}$ містить ідентифікатор стану Id_{Sp} та стани усіх компонент, що належать до параметризованого мережного інтерфейсу НС кіберфізичних систем [60].

Враховуючи формули 2.1, 2.11, 2.12 отримаємо функцію переходу для параметризовананого мережного інтерфейсу НС кіберфізичних систем:

$$\lambda(H_i, Sp_j) = \left\{(Sp_k, Pb\ (Sp_k, H_i, Sp_j)\right\}, \tag{2.13}$$

$$\sum_k Pb\ (Sp_k, H_i, Sp_j) = 1, \tag{2.14}$$

Враховуючи формули 2.1, 2.11, 2.12 отримаємо функцію виходу для параметризованого мережного інтерфейсу НС кіберфізичних систем:

$$f(H_i, Sp_j) = \{(L_k, Pb\ (Sp_j, H_i, B_k)\}, \quad (2.15)$$

$$\sum_k Pb\ (Sp_j, H_i, L_k) = 1, \quad (2.16)$$

У проектах по тестуванню параметризованого мережного інтерфейсу НС кіберфізичних систем використовувалися ті самі етапи, як і при тестуванні мережного інтерфейсу.

Параметризований мережний інтерфейс НС кіберфізичних систем може мати такі ж недоліки як і традиційний мережний інтерфейс, але за рахунок параметрів у параметризованому мережному інтерфейсі НС кіберфізичних систем ці недоліки можна виправити або забезпечити його ефективнішу роботу [26].

Взаємодію клієнта з сервером через параметризований мережний інтерфейс НС кіберфізичних систем на основі шаблонів була реалізована через:

- клієнт-активований об’єкт;
- сервер-активований об’єкт.

Стан віддаленого об’єкту за посиланням використовувався, коли клієнт робить запит до об’єкта, який знаходиться на віддаленому сервері. Активація режиму віддаленого об’єкта відбувалася у двох варіантах: клієнт-активований об’єкт і сервер- активований об’єкт. Вибір варіанту залежав від клієнта. Він обирав, який з варіантів використовувати. Вибір активації режиму мав вплив на загальний вигляд мережного інтерфейсу, продуктивність і масштабованість віддаленого додатку. Він визначав, коли віддалені об’єкти створюються, скільки цих об’єктів створено, як керувати їхнім життєвим циклом, і чи буде підтримувати стан віддалених об’єктів інформацію, яку вони містять в собі. На рисунку 2.6 показана взаємодія клієнта з сервером через параметризований мережний інтерфейс НС кіберфізичних систем за допомогою віддаленого об’єкту.

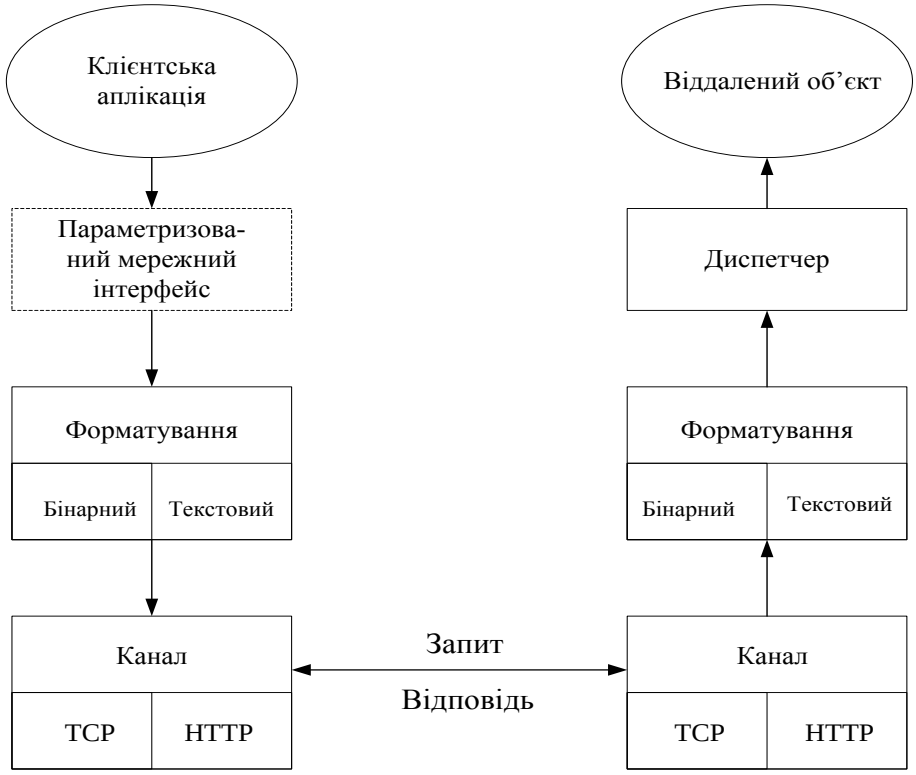


Рисунок 2.4 - Схема взаємодії клієнта з сервером через параметризований мережний інтерфейс НС кіберфізичних систем за допомогою віддаленого об'єкту

Коли клієнт намагався отримати доступ до віддаленого об'єкту через параметризований мережний інтерфейс НС кіберфізичних систем, його посиланням на віддалений об'єкт насправді займався посередник, відомий як сервер-активований об'єкт. При його використанні клієнт приймав запит і пересилав його на віддалений об'єкт. Сервер-активований об'єкт активував параметризований мережний інтерфейс НС кіберфізичних систем для клієнта, який відповідав його класу. Це дозволяло активованим об'єктам переконатися, що всі клієнтські запити відповідають заданому цільовому методу, тобто, тип і кількість параметрів збігаються. Розробник ніс відповідальність за забезпечення метаданими, які визначають клас віддаленого об'єкта доступного під час компіляції та виконання [27].

Найпростіший спосіб полягав в наданні клієнту копії віддаленого об'єкту на сервері. Після того, сервер-активований об'єкт форматував повідомлення, як відповідь на запит клієнт-активованого об'єкта, який реалізував параметризований мережний інтерфейс НС кіберфізичних систем.

Повідомлення клієнт-активованого об’єкта класу передавалося, як параметр для виклику реальних методів сервер-активованого об’єкта, які передавалися по каналу мережного з’єднання. Формат об’єкта, містився у повідомленні і передавався в об’єкт через канал мережного з’єднання.

Поведінка клієнта-активованого об’єкту при використанні відаленого об’єкта нагадувала, що об’єкт створений у ньому. Обидва вони могли бути створені за допомогою нового оператора, який використовував параметризовані конструктори на додаток за умовчанням, і як зберегти інформацію про стан властивостей або полів. Як показано на рисунку 2.5, вони відрізняються тим, що сервер-активованим об’єктом працював на проксі - сервері в окремому домені програми [31,66,68].

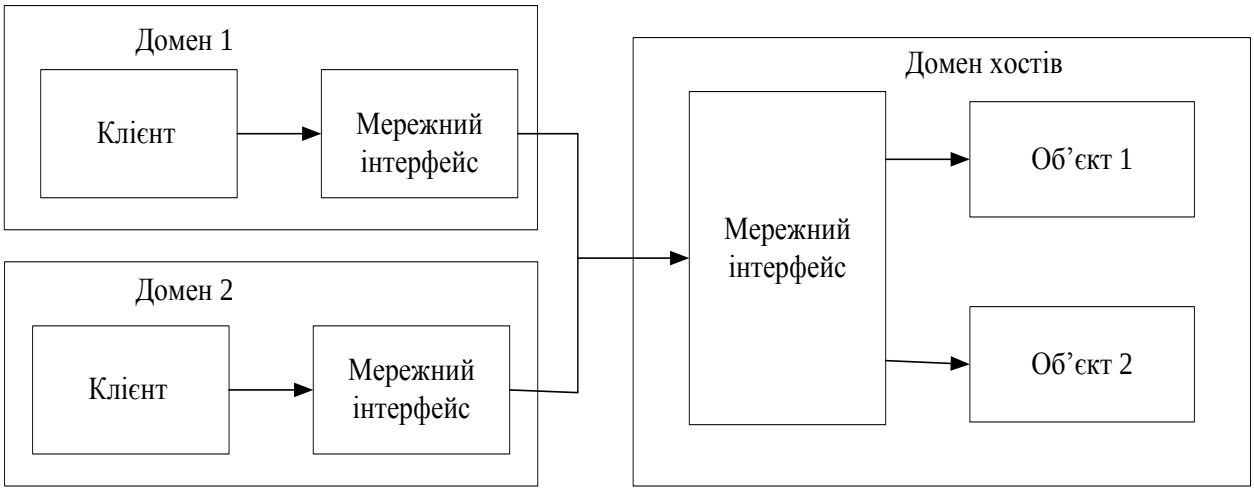


Рисунок 2.5 - Схема взаємодії з клієнт-активованим об’єктом, клієнт зберігає контроль над об’єктом

Віддалений об’єкт знаходився в іншому домені це означало, що він просто існує і може бути знищений навіть якщо клієнт, як і раніше використовує його. Домен обробляв цю потенційну проблему, позначивши кожен віддалений об’єкт, який може бути використаний, щоб зберегти його активним. Сервер-активований об’єкт був реалізований, як з одного виклику віддаленого об’єкту так і один віддалений об’єкт створювався для кожного запиту через використання параметризованого мережного інтерфейсу НС кіберфізичних

систем. У цьому випадку використання віддаленого об’єкта, краще підходить для спільного використання одного ресурсу, або спільних операцій між декількома користувачами.

Клієнтські запити до сервера у глобальній мережі, повинні виконуватися за відносно короткий час декілька операції на сервері, який не вимагав збереження інформації про стан об’єкту від одного виклику до іншого. Такий підхід є найбільш масштабованим рішенням і мав додаткову перевагу, працював добре в середовищі, яке використовувало балансування навантаження клієнтськими запитами об’єкта сервера. Єдиний віддалений об’єкт використовувався для обробки всіх викликів клієнтів через параметризований мережний інтерфейс НС кіберфізичних систем (Рисунок 2.6).

.

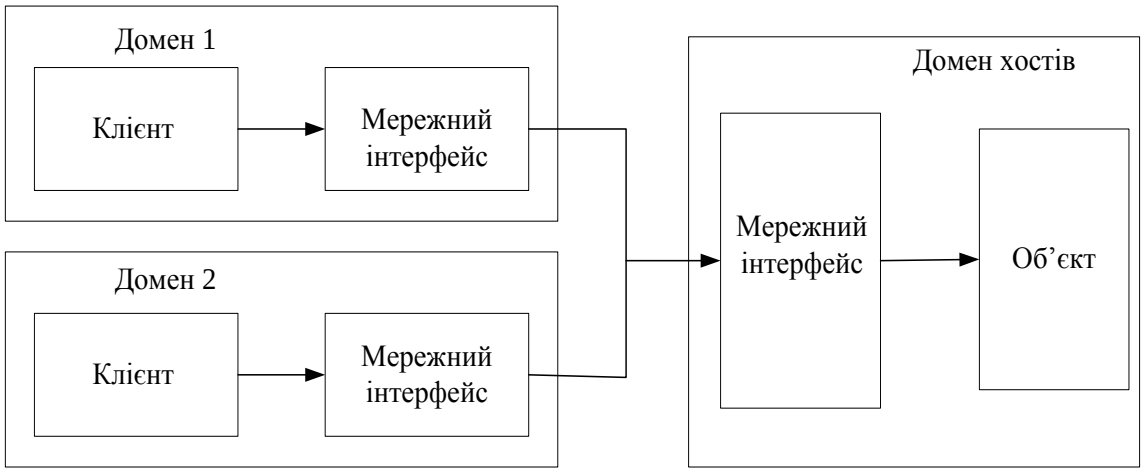


Рисунок 2.6 - Схема взаємодії з сервер-активованим об’єктом: один об’єкт обробляє всі запити

Сервер створював віддалений об’єкт, коли перший клієнт намагався отримати до нього доступ, а не коли він створював його. Тому, що віддалений об’єкт створювався тільки один раз, за допомогою інших клієнтів, щоб створити його екземпляр, замість цього посилання на той же єдиний об’єкт, через параметризований мережний інтерфейс НС кіберфізичних систем. Кожен

раз, коли клієнт викликає віддалений об’єкт виділявся один новий потік з переліку потоків. Це у свою чергу обмежувало масштабованість.

У режимі активації запитами, сервер створював новий віддалений об’єкт кожен раз, коли зроблено запит на нього. Після запиту об’єкт опрацьований, тоді віддалений об’єкт - вимкнено. Кілька запитів оброблялося, коли перший запит був завершений, і віддалений об’єкт, створений для нього буде видалений на момент, тоді клієнт, який зробив цей запит посилав на нульове значення (Рисунок 2.7).

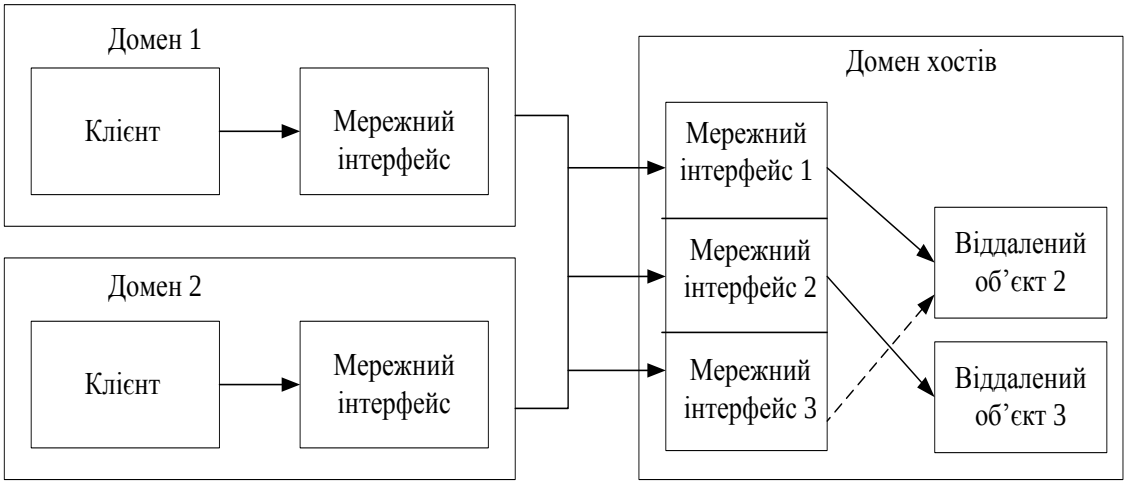


Рисунок 2.7 - Схема взаємодії з сервер-активованим об’єктом: один об’єкт створювався для кожного запиту

Другий виклик виходив від іншого клієнта, і в результаті відбувалося створення другого об’єкту сервера. Коли перший клієнт опрацьовува свій другий запит і третій, тоді був створений віддалений об’єкт [69,73]. Перевагою активації клієнта запитом є те, що ресурси були доступні, як тільки запит був завершений.

2.4 Вибір критеріїв ефективності мережного інтерфейсу навігаційного сервісу кіберфізичних систем

Оцінку ефективності мережного інтерфейсу проводилося за допомогою:

• формуванні вимог, які ставилися до мережного інтерфейсу;

• аналізу створюваних і функціонуючих мережних інтерфейсів на відповідність необхідним вимогам;

• вибору найкращого варіанту створення, функціонування і розвитку мережного інтерфейсу;

• синтезу (формування) найбільш доцільного варіанту розробки мережного інтерфейсу за критерієм “ефективність – затрати”.

Також порівняння мережного інтерфейсу виконувалося через критерії ефективності:

1.Час реакції мережного інтерфейсу, мс;

2.Термін розробки мережного інтерфейсу, люд./год.

Показник ефективності завжди має кількісний зміст, тобто є вимірюванням деякої властивості. З цієї причини використання деякого показника ефективності передбачало наявність способу вимірювання (оцінки) значення цього показника. Для оцінок ефективності мережного інтерфейсу можуть братися, наприклад, такі показники, як продуктивність, вартість, надійність, габарити і т.д. На практиці показник ефективності вибирають так, щоб він був критичним до тих факторів, вплив яких на мережний інтерфейс найбільш суттєвий у певній конкретній ситуації.

Часткові показники ефективності були суперечливими, так як покращення одного з них могло викликати погіршення інших, а це приводило до зниження ефективності мережного інтерфейсу.

У зв’язку з цим при проектуванні та експлуатації технічних компонентів мережного інтерфейсу, необхідно проводити комплексну оцінку цього інтерфейсу за показниками ефективності з врахуванням надійності, прагнучи до максимальної ефективності мережного інтерфейсу.

Ефективності оцінювалася на основі швидкодії конкретної реалізації мережного інтерфейсу. На основі отриманих даних від мережних інтерфейсів на основі різних принципів, дали можливість встановити межі для порівняння мережного інтерфейсу з стандартними засобами розробки, таблиця 2.2.

Таблиця 2.2 Межі для порівняння мережних інтерфейсів НС кіберфізичних систем

Тип інтерфейсу **Прикладний/Ethernet**	Час реакції мереж-ного інтерфейсу, мс	Термін розробки мережного інтерфейсу, люд./год.
Спеціалізований мережний інтерфейс розроблений стандартним засобами	1,0-2,0	7,0-9,0
Модульний мережний інтерфейс розроблений на основі принципу:		
Наслідування	0,1-0,5	2,0-3,0
Використання	0,1-0,5	0,5-1,0
Інстанціювання	0,01-0,05	1,0-2,0
Модульний мережний інтерфейс розроблений стандартним засобами	0,5-0,1	3,0-4,0
Мережний інтерфейс розроблений на основі Jini	1,0-2,5	7,0-8,0

Отримані дані показують, що розроблений модульний мережний інтерфейс дозволяє зекономити час на розробку, що зменшує трудомісткість. А реалізація не завжди, але в більшості випадків ефективніша ніж в універсальному мережному інтерфейсі.

Як видно із таблиці 2.2, модульний мережний інтерфейс має кращі показники за інші види мережних інтерфейсів.

Приймемо за 100% ефективний реалізований мережний інтерфейс НС кіберфізичних систем по сумі 2 критеріїв ефективності. Якщо менше 100% до межі 56% можна вважати реалізований мережний інтерфейсів НС кіберфізичних систем по 2 критеріях ефективним. При межі від 55% - 15% можна вважати реалізований мережний інтерфейс НС кіберфізичних систем по

2 критеріях менш ефективним. Якщо межа від 14% до 5% можна вважати реалізований мережний інтерфейс НС кіберфізичних систем по 2 критеріях не ефективним.

2.5 Висновки до розділу

1. Аналіз традиційної математичної моделі створення мережних інтерфейсів НС кіберфізичних систем традиційними методами не відповідала критеріям покращення ефективності реалізації мережних інтерфейсів НС кіберфізичних систем таким як:
- термін розробки мережного інтерфейсу;
- час реакції мережного інтерфейсу.
2. Запропонована концепція модульного мережного інтерфейсу у навігаційних сервісах кіберфізичних систем яка, окрім зменшення часу його розробки, дозволила спростити процес розробки мережного інтерфейсу.
3. Проаналізовано, що стандартні методи реалізації мережного інтерфейсу у навігаційних сервісах кіберфізичних систем, які не дали можливості формалізувати тестування даного мережного інтерфейсу. Зокрема це стосувалося визначення початкових умов і невідомого наперед, рівня масштабування величини навантаження.
4. Вперше запопоновано метод розробки мережного інтефейсу НС кіберфізичних систем, через використання параметрів мережного інтерфейсу для коректної розробки цього типу інтерфейсу.
5. Для розв’язання вищезгаданих проблем розроблено модель параметризованого мережного інтерфейсу у навігаційних сервісах кіберфізичних систем.
6. Запропоновано межі для порівняння мережних інтерфейсів НС кіберфізичних систем на ефективність реалізованими традиційними і вдосконаленими методами.

РОЗДІЛ 3

ВДОСКОНАЛЕННЯ МЕТОДІВ РОЗРОБКИ МЕРЕЖНИХ ІНТЕРФЕЙСІВ НАВІГАЦІЙНИХ СЕРВІСІВ КІБЕРФІЗИЧНИХ СИСТЕМ

3.1 Вдосконалення статично-динамічного методу розробки мережного інтерфейсу в НС кіберфізичних систем

Важливо, що статично-динамічний метод підвищення ефективності розробки мережного інтерфесу в НС кіберфізичних систем керований даними та командами клієнта. Він містить визначений модульний мережний інтерфейс НС кіберфізичних систем в термінах заданих вхідних даних та отриманих виходів. Специфікація виходів та входів (СВВД) модульного мережного інтерфейсу НС кіберфізичних систем у цьому методі полягає в тому, що базова інформація щодо кожної вхідної та вихідної змінної модульного мережного інтерфейсу, при цьому методі, включає назву змінної, тип та діапазон можливих значень. Ця інформація зазвичай доступна з інструментів управління вимогами [120]. А також, генератор випадкових тестів (ГВТ) для тестування мережного інтерфейсу. У модульному мережному інтерфейсі НС кіберфізичних систем, модуль генерує випадкові комбінації значень з діапазону кожної вхідної змінної. Кількість навчальних тестів для генерації було визначене клієнтом. Генеровані тести в подальшому застосовуються в каналах зв’язку для коректної передачі даних в мережі. Виконання тренувальних випадків було проведене комерційними інструментами автоматизації тестування модульного мережного інтерфейсу. Клієнт отримував від сервера вихідні дані після проходження кожного випадку, які містив модульний мережний інтерфейс. Вхідні дані включаюли в себе тренувальні тести, генеровані ГВТ, та вихідні тести, отримані результати після кожного етапу тестування мережного інтерфейсу.

Статично-динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем повторно виконувався для знаходження підмножини релевантних вхідних змінних до кожного виходу і відповідної множини не надлишкових даних. Актуальні дані генерувалися за автоматично визначеними класами еквівалентності з використанням існуючих команд клієнта. Кожна вихідна змінна представлена відокремленою безкабельною мережею. Переваги та можливості: ГВТ одержує таблицю виходів та входів модульного мережного інтерфейсу разом із їхніми типами із специфікації НС кіберфізичних систем. Не потребувалася жодна інформація щодо функціональних вимог, бо для цього методу автоматично визначалися відносини входу-виходу.

На базі цього методу навчалася кіберфізична система на вхідних даних та командах від клієнта, представлених ГВТ, та вихідних, одержаних від безкабельної мережі за допомогою модульного мережного інтерфейсу НС цієї систем. У свою чергу, відокремлений метод розробки модульного мережного інтерфейсу на основі принципу наслідування використовувався для кожної вихідної змінної. Наступна інформація була отримана із модульного мережного інтерфейсу НС кіберфізичних систем:

1. Множина вхідних атрибутів релевантних до відповідних вихідних.

2. Логічні (якщо ... то) правила, що показували відносини між обраним вхідним атрибутом та відповідним вихідним. Множина правил була сформована для кожного кінцевого клієнта, відображаючи розподіл вихідних значень. Ці правила були використані для визначення прогнозу (найбільш ймовірнісного) значення виходу відповідного тестового випадку.

3. Інтервали дискретизації кожної неперервної вхідної змінної були включені в безкабельну мережу.

4. Множина не надлишкових даних. Кінцеві клієнти мережі перетворювали на тести, кожне відображення не надлишкового зв’язку входу і класів еквівалентності та відповідного розподілу значень виходу. В будь-який час, коли поведінка модульного мережного інтерфейсу НС кіберфізичних систем,

який тестувався, характеризувався деяким стандартним випадком, очікував число запитів, від клієнта до сервера у безкабельні мережі, значно меншим від кількості випадкових запитів, що використовували для навчання безкабельної мережі.

Іншою важливою властивістю даного статично-динамічного методу підвищення ефективності розробки мережного інтерфесу в НС кіберфізичних систем є стійкість по відношенню до тренувальних випадків, навчання проводиться лише на малих та випадково генерованих підмножинах прийнятних вхідних значень.

Вдосконалений статично-динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем направлений на наступні невирішені проблеми, пов’язані із регресійним тестуванням модульних мережних інтерфейсів НС кіберфізичних систем:

• Він автоматично створює множину не надлишкових запитів, що покривають найважливіші функціональні відносини, присутні в навігаційних сервісах.

• Він придатний для тестування комплексних навігаційних сервісів кіберфізичних систем, бо не залежить від аналізу системного коду.

• Використання цього методу не потребує істотної підтримки з боку людини. Існуючі методи та технології проектування модульних мережних інтерфейсів потребують базового аналізу вимог чи коду.

• Пропущені, застарілі або неповні вимоги не є перепоною для запропонованої методології, бо вона вивчає функціональні взаємовідносини автоматично за даними реалізації мережних інтерфейсів. Існуючі методи підвищення ефективності розробки модульних мережних інтерфейсів НС кіберфізичних систем потребують наявність деталізованих вимог, які можуть бути недоступними або неповними в багатьох навігаційних сервісах.

Отже, у випадках коли структура даних і команд клієнтів не змінювалася і повторювалася, тобто вона статично-динамічна, було реалізовано модульний мережний інтерфейс НС кіберфізичних систем на основі принципу

наслідування. Цей принцип дозволив одному класу наслідувати характеристики структуру даних і команд іншого [31]. При цьому похідний клас отримував всі функціональні можливості базового класу, й був вдосконалений за рахунок додавання власних. Тобто, похідний клас являв собою спеціалізовану версію базового класу. Похідний клас наслідував всі члени, визначені в базовому класі, й додавав до них власні унікальні елементи. Базовий клас при наслідуванні залишався незмінним.

Вигляд вдосконаленого методу підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування показано на рисунку 3.1.

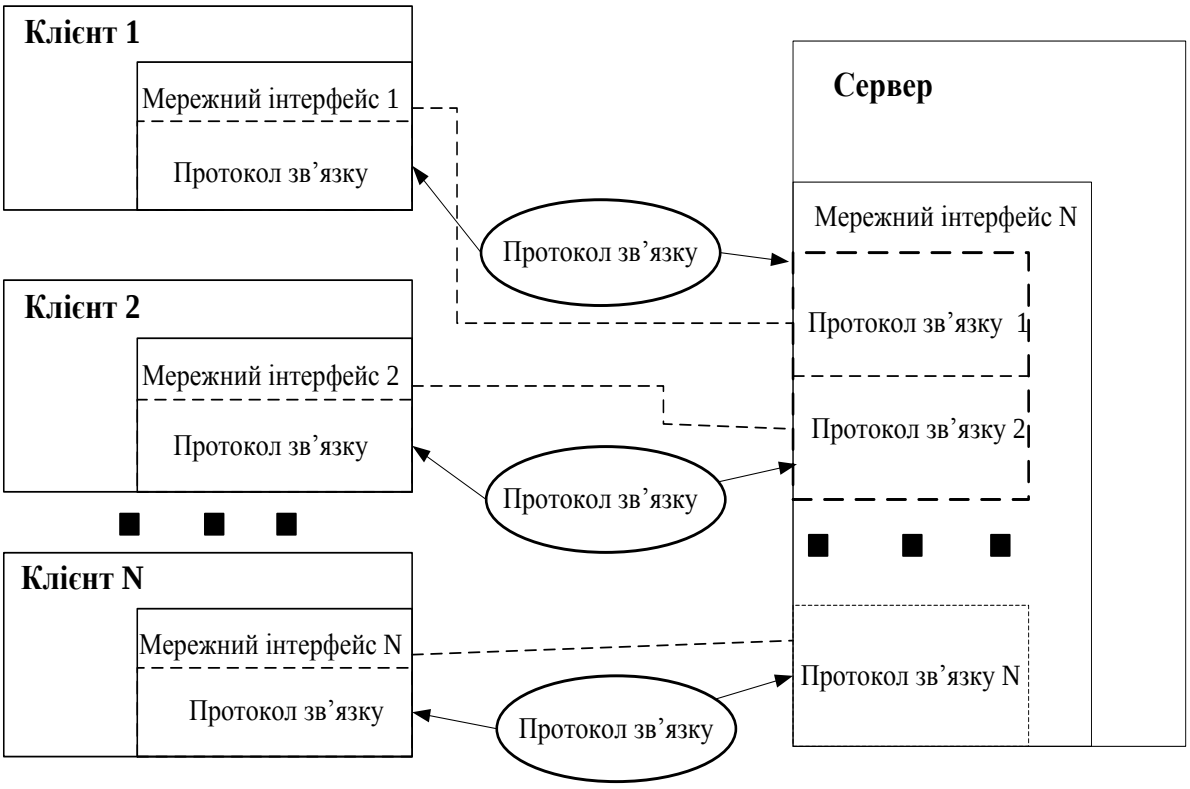


Рисунок 3.1- Схема методу розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування

У вдосконаленому методі підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування для підключення кожного клієнта на сервері використовувався

свій мережний інтерфейс (1,2,..,N), який містив протокол зв’язку (1,2,..,N). Протоколи зв’язку було розбито на універсальну частину, спільну для всіх мережних інтерфейсів і спеціальну частину, характерну для кожного типу клієнта. Універсальну частину мережного інтерфейсу НС кіберфізичних систем описує базовий клас. Для кожного типу протоколу зв’язку створювався свій клас спеціальної частини мережного інтерфейсу НС кіберфізичних систем, шляхом наслідування базового класу з розширенням його набором спеціальних команд. Метод підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування був вдосконалений з використанням параметризації команд та даних (Рисунок 3.2).

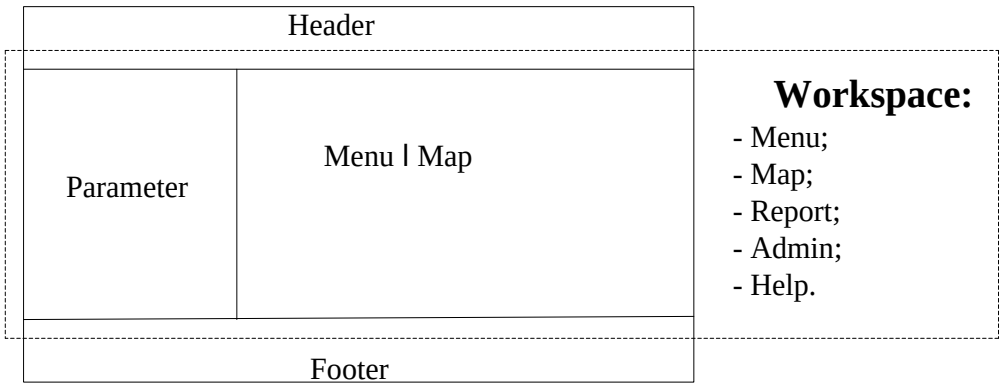


Рисунок 3.2 - Схема методу підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування вдосконалений з використанням параметризації команд та даних.

Цей модульний мережний інтерфейс складався з Header і Footer, які не змінюються. А також Workspace, який має вигляд не заповненого поля, або містить Parameter і Menu l Map. Workspace може містити:

- Menu;
- Map;
- Report;
- Admin;

- Help.

Для того щоб побудувати цей модульний мережний інтерфейс, який був веб-інтерфейсом, існує декілька варіантів:

1.Будуємо Header, Workspace, Footer. Над цим дуже багато надлишкової роботи і потрібно кожен раз змінювати Header, Footer.

2.Створюємо базові класи. Базовий клас Class DocumentWrapper він міститиме Header, Footer і Workspace без наповнення. Його будуть наслідувати наступні класи: Menu, Map, Report, Admin, Help. Програмна реалізація:

Class Menu : public DocumentWrapper;

Class Map : public DocumentWrapper;

Class Report : public DocumentWrapper;

Class Admin : public DocumentWrapper;

Class Help : public DocumentWrapper.

- Також WEB-інтерфейс може містити шаблони:

  Parmetr, MenuContent.

  Parmetr, MapContent.

  ReportContent

  AdminContent

  HelpContent

Також принцип наслідування при розробці модульного мережного інтерфейсу НС кіберфізичних систем виконувався наступним чином:

Class ReportDevice : Report

Class ReportDeviceDetail : ReportDevice

Де Class Report говорить про те, що дані беруться з базового класу EventData і прив’язуються до Class Device. Цей клас також шаблонний він задає вивід. А конкретні дані отримуються з Class ReportDeviceDetail.

У вдосконаленому методі підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування параметризація даних і команд дозволила зменшити об’єм коду. Також перевагою цього методу є те, що принцип наслідування дозволило

використовувати існуючий код батьківського класу у всіх похідних класах багато разів. Використовуючи вдосконалений метод підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування, створюється базовий клас, який визначає риси, притаманні багатьом об’єктам. А потім використовується його для створення будь-якого числа спеціалізованих похідних класів з додаванням у кожен з них своїх унікальних особливостей.

3.2 Вдосконалення динамічного методу розробки мережного інтерфейсу в НС кіберфізичних систем

Динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем базувався на аналізі модульного мережного інтерфейсу безпосередньо при його роботі. Динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кібефізичних систем було розбито на кілька етапів підготовка початкових даних, проведення тестового запуску модульного мережного інтерфейсу НС кіберфізичних систем і збір необхідних параметрів і аналіз отриманих даних. Динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем найбільш важливий в тих навігаційних сервісах, де головним критерієм є надійність цього модульного мережного інтерфейсу, час відклику чи споживані ресурси. Це може бути, наприклад, навігаційні сервіси, що керують моніторингом рухомих об’єктів, або сервер бази даних.

В навігаційних сервісах будь-яка допущена помилка, може виявитися критичною. Переваги та можливості динамічного методу підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем:

1. В більшості реалізацій поява помилкових спрацьовувань була виключена так, як знаходження помилки відбувалася в момент її виникнення в модульному мережному інтерфейсі. Таким чином, знайдена помилка не є передбаченням, зробленим на основі аналізу розробки цього методу, а констатацією факту її виникнення;

2. Зазвичай немає необхідності в вихідному коді. Це дозволило протестувати модульний мережний інтерфейс із закритим кодом. З допомогою динамічного методу отримані наступні результати:

- споживані ресурси час виконання програми в цілому чи її окремих модулів, кількість зовнішніх запитів, об’єм використаної оперативної пам’яті та інших ресурсів;

- програмні помилки такі , як ділення на нуль, витік пам’яті, «стан гонки».

Вдосконалений динамічний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем полягає у:

• виконання розробки модульного мережного інтерфейсу, маючи задані чи випадкові вхідні дані;

• збір символьних обмежень входів модульного мережного інтерфейсу на умовні оператори протягом виконання;

• використання перевірника обмежень модульного мережного інтерфейсу, щоб вивести варіації попередніх вхідних даних, для управління виконанням програми по альтернативній гілці.

У випадках коли структура даних клієнтів у базі даних часто змінювалася, тобто вона була динамічною, але самі команди були не змінні, було реалізовано модульний мережний інтерфейс НС кіберфізичних систем на основі принципу інстанціювання. Існують стандартні команди роботи з базами даних:

- select;
- insert;
- update;
- delete.

Незалежно від того, яку взято таблицю необхідно було б, щоб ці команди працювали коректно. Виникла проблема в тому, що база даних містить багато даних і вони всі різнотипні, а операцій всього 4. Не хотілося б писати в ручну для кожного типу даних конкретну операцію. Тому, що для звіту (Report) був використаний шаблон звіту про пристрої з якої вибиралися всі поля для класу

EventData. Але під конкретний звіт вибиралися ті поля, які потрібні в цьому звіті, наприклад про зупинки потрібні дані про кількість координат, час, місце зупинки все інше не потрібно на зразок входів виходів і так далі. Було реалізовано стандартні операції з базою даних, як «узагальнені», а у узагальненому програмуванні відповідно, застосовано шаблон. Параметром шаблона були назва таблиці та список полів. Таким чином, отримано узагальнений шалон, який пише в базу даних і читає з бази даних узагальнено. А коли, цьому шаблону задавався список полів або назву таблиці, то шаблон не просто узагальнено виконувався у програмі, а конкретно. Таким шляхом, вдосконалювався метод підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання, а саме із застосуванням параметризації команд, як було сказано вище.

Вдосконалений метод підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання передбачав використання принципу узагальненого програмування або програмування на основі шаблонів, в якому тип даних є параметром. Для створення об’єкту з цього класу необхідно було задати тип даних [15]. Цей процес називався інстанціюванням.

Модульний мережний інтерфейс НС кіберфізичних систем було реалізовано, як шаблон з використанням принципу інстанціювання. Універсальну частину цього модульного мережного інтерфейсу було описано класом шаблона, а спеціальну частину класом параметра.

На рисунку 3.3 наведено вигляд вдосконаленого методу підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання.

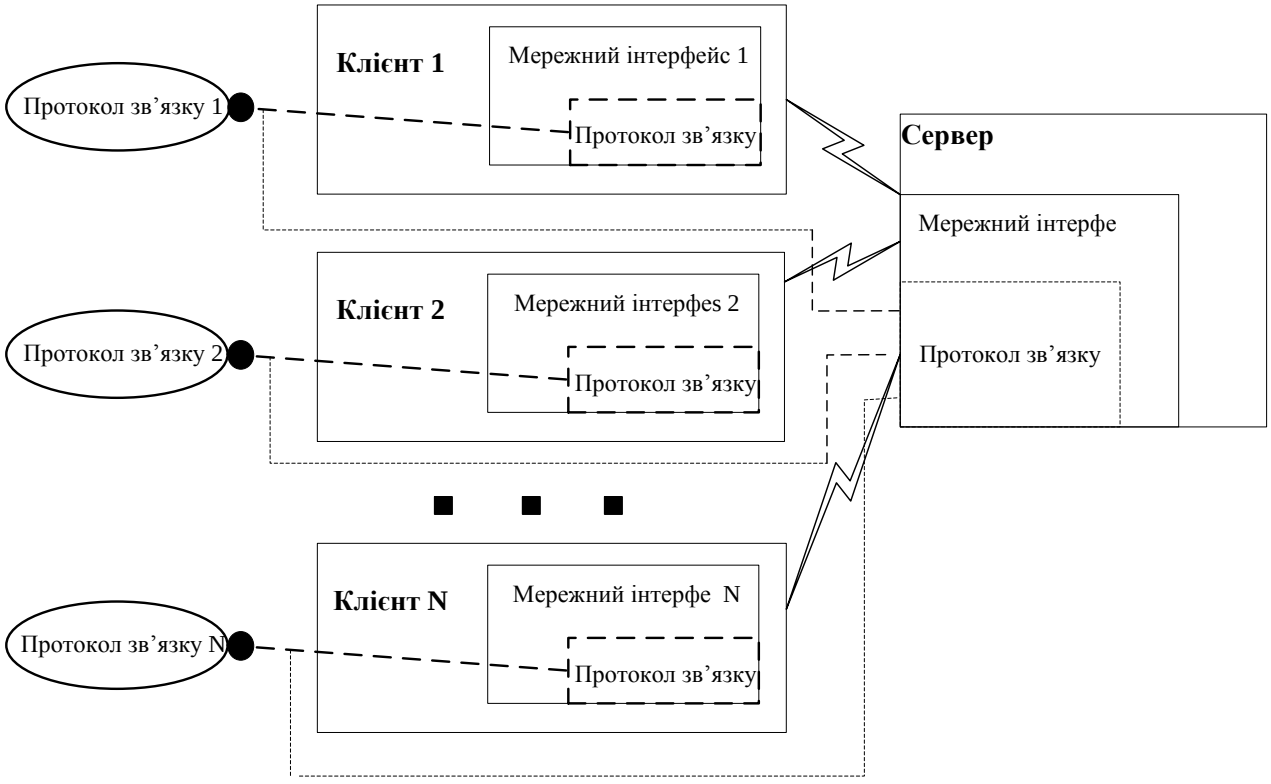


Рисунок 3.3 - Схема методу розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання

Певний мережний інтерфейс НС кіберфізичних систем в даному вдосконаленому методі був об'єктом параметризованого класу шаблона інстанційованого класом спеціальної частини цього мережного інтерфейсу. Інстанціювання шаблону різними класами спеціальної частини мережного інтерфейсу НС кіберфізичних систем проводилося створенням різних мережних інтерфейсів НС кіберфізичних систем. Тобто, для реалізації багатьох мережних інтерфейсів НС кіберфізичних систем з використанням принципу інстанціювання необхідно, було створити один клас шаблону універсальної частини мережного інтерфейсу НС кіберфізичних систем і необхідну кількість класів спеціальних частин мережних інтерфейсів НС кіберфізичних систем. Клас шаблону експортував операції, які можна виконати над його екземпляром. А навпаки, параметризований аргумент класу використовувався для імпорту класів і значень, які були реалізовані як протоколи зв'язку. У програмній реалізації використовувалася мова програмування Java саме тому, що вона під

час компіляції провіряє їх взаємодію, коли фактично проводиться інстанціювання.

3.3 Вдосконалення статичного методу розробки мережного інтерфейсу в НС кіберфізичних систем

Статичний метод підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем часто використовувався, як засіб пошуку у вихідному або двійковому коді програм певних шаблонів або ситуацій (порушень проектних угод, використання бібліотек, помилок стилю кодування, властивостей мови програмування) у реалізації модульного мережного інтерфейсу НС кіберфізичних систем. Головна перевага статичного методу підвищення ефективності розробки мережного інтерфейсу НС кіберфізичних систем в можливості суттєвого зниження вартості усунення дефекту в модульного мережного інтерфейсу НС кіберфізичних систем тому, що ці інструменти дозволили виявити значну кількість помилок ще на етапі проектування. Наступні переваги статичного методу підвищення ефективності розробки мережного інтерфейсу в НС кіберфізичних систем:

1. Повне покриття коду. За допомогою статичного методу підвищення ефективності розробки мережного інтерфейсу перевіряють навіть ті частини коду реалізації модульного мережного інтерфейсу, що досить рідко отримувалися управління, та їх, як правило, не вдавалося протестувати іншими методами. Це дозволяє знаходити дефекти в обробниках помилок чи в системі логування навігаційного сервісу.

2. Цей метод не залежить від безкабельної мережі, яка використовувалася, та навігаційного сервісу, в якому реалізований модульний мережний інтерфейс. Це дозволяє знаходити приховані помилки, що проявляють себе при зміні навігаційного сервісу чи використанні інших ключів для оптимізації його коду реалізації.

У випадках, коли мережний інтерфейс клієнта НС кіберфізичних систем є специфічний для кожного клієнта і містив унікальні команди, але подібну

структуру даних, тобто мережний інтерфейс є статичним і був реалізований модульний мережний інтерфейс НС кіберфізичних систем на основі принципу використання. Під вдосконаленням методу підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу використання (так званому композицією або включенням) малося на увазі методику створення нового класу з вже існуючих класів шляхом включення, званого також делегуванням, вкладеного об’єкту. Вкладені об’єкти нового класу оголошувалися закритими, що робило їх недоступними.

З іншого боку, розробник класу міг змінювати ці об'єкти, не порушуючи при цьому роботи існуючого клієнтського коду [15]. Крім того, заміна вкладених об'єктів на стадії виконання програми дозволила динамічно змінювати її поведінку. Вдосконалений метод підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування такої гнучкості не показав, оскільки для похідних класів встановлювалися обмеження, що перевірялися на стадії компіляції.

На базі вдосконаленого методу підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу використання реалізувалася методика делегування, при якій поставлене зовнішньому об’єкту завдання передоручалося внутрішньому об'єкту, що спеціалізувався на вирішенні задач такого типу. В цьому методі розробки існувало відношення «частина-ціле» між двома рівноправними об'єктами: мережним інтерфейсом і протоколом зв’язку.

При цьому один об’єкт (контейнер) мав посилання на інший об’єкт. Обидва об’єкти могли існувати незалежно: якщо контейнер був знищений, то його вміст - ні.

На рисунку 3.4 наведено вигляд вдосконаленого методу підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу використання.

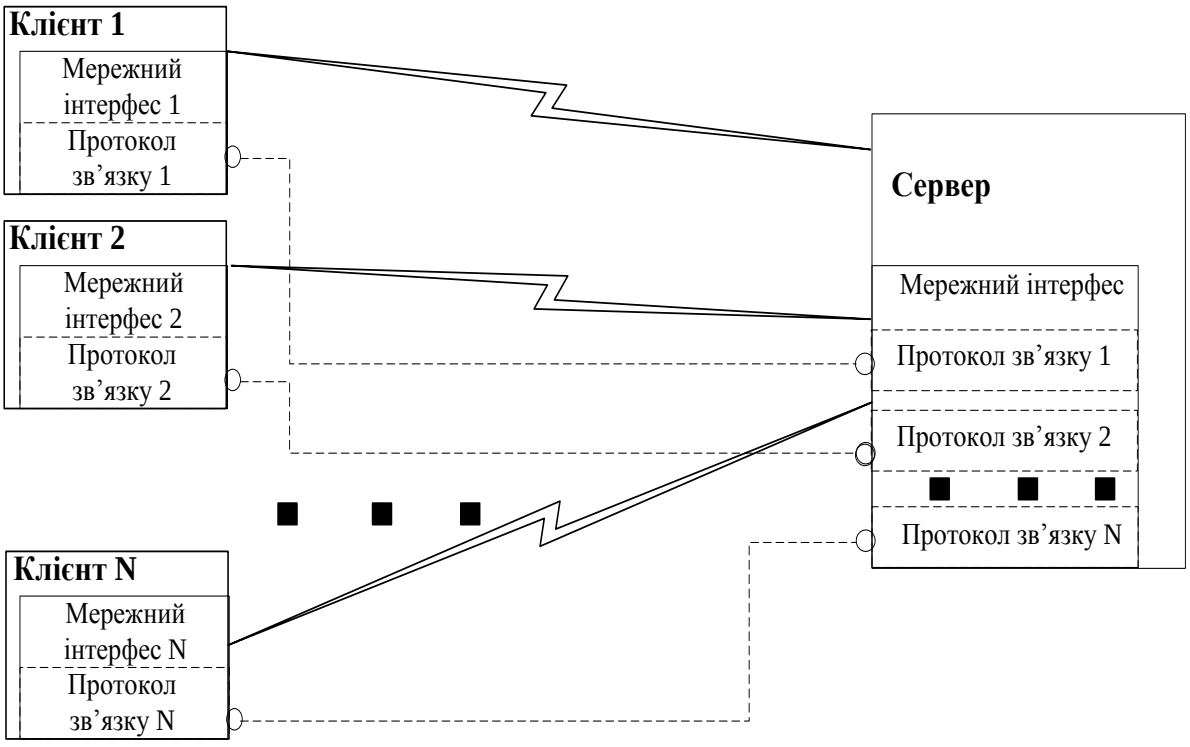


Рисунок 3.4 - Схема методу розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу використання

У вдосконаленому методі підвищення ефективності розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу використання, мережний інтерфейс використовувався, як об'єкт класу контейнера. При застосуванні принципу використання, мережний інтерфейс у сервера один, але він для роботи з клієнтом 1 використовував протокол зв'язку 1, для роботи з клієнтом 2 використовував протокол зв'язку 2, для роботи з клієнтом N використовував протокол зв'язку N.

В загальному випадку можемо сказати, що у кожного клієнта свій тип мережного протоколу і цей тип визначав тип клієнта (пристрою) і відповідно набір команд клієнта. Вигляд команди, яку надсилатиме клієнт (пристрій) через мережний протокол серверу, показано на рисунку 3.5.

Від сервера до пристрою		
Сервер	Команда+параметри. Вигляд: код, N, команда, параметр 1,….,параметр N	Надсилання
Пристрій	ОК Вигляд: код, N, команда, ОК (“0” l ”1”)	Відповідь

Рисунок 3.5 - Вигляд команди у мережному протоколі

Для кожного типу мережного протоколу характерний набір команд, кожна з яких оброблялася відповідною функцією. Вигляд одної з таких команд можна побачити на рисунку 3.6.

Від сервера до пристрою		
Сервер	Код+Команда. Вигляд: Команда1 Команда2…. КомандаN	Надсилання
Пристрій	Код. Вигляд: код N 11 код N 10….. код N 00	Відповідь

Рисунок 3.6 - Вигляд команди для конкретного типу мережного протоколу

Якщо команда була прийнята то 1, якщо не прийнята то 0. Тут існують механізми взаємодії між сервером і пристроєм:

- Безпосереднє з’єднання сервера з клієнтом. Клієнт з’єднувалися з сервером 3 рази для встановлення з’єднання.
- З’єднання “exo”. Під час якого клієнт встановлює з’єднання, а сервер-exo передає команду.

- Через протоколи взаємодії. Зі сторони сервера формувалася смс і відправлялася на пристрій, коли пристрій її отримував невідомо. У свою чергу пристрій відповідав, так само смс. Тобто, виконувалося асинхронне з'єднання. Через те, що "ехо" зєднання розмежовувало дані і команди, тобто було не відомо, коли отримуються дані і команди і як це привести до спільного знаменника. Використовувалися таблиці в базі даних з не відправленими командами, яка матиме наступний вигляд (Рисунок 3.7).

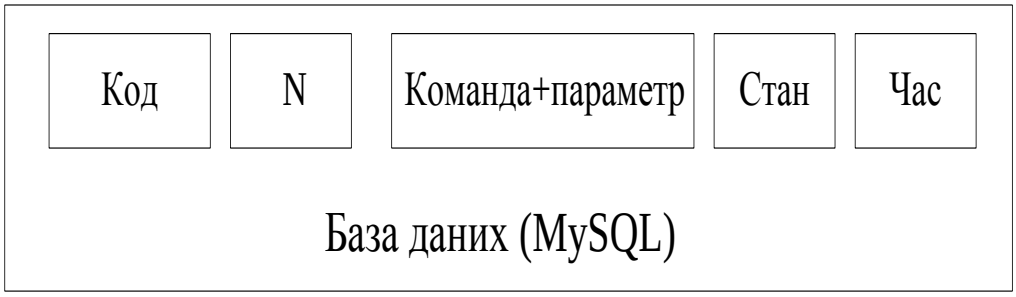


Рисунок 3.7 - Вигляд таблиці в базі даниз із не відправленими командами

Коли накопичувалися команди вони заносилися в базу даних. Коли клієнт відправляв запит до сервера то у цій таблиці змінювався стовпець «Стан» на можливі варіант:

«0»-не відправлено;

«1»-відправлено;

«2»-підтвердження.

Взалежності, який запит приходив від сервера, то у стовпці «Стан» цифра 2, тобто з'єднаня між пристоєм і сервером було встановлене. Якщо цифра 1 то через, якийсь час сервер змушений знову відправити свій запит на пристрій, це могло повторювати 3 рази, якщо жодного разу сервер не отримував відповіді, то пристрій в неактивному стані або бракований.

Набір форматів команд і функцій обробки містився в об'єкті протоколу зв'язку, який використовувався клієнтом і сервером. Лише за однією відмінністю в кожного клієнта були свої мережні інтерфейси, а у сервера він один для них всіх. У свою чергу протоколи зв'язку (1,2,..,N) були включені в

об’єкт мережного інтерфейсу. У цьому методі був один об’єкт мережного інтерфейсу і відповідний йому клас, і багато об’єктів, які відповідали за протоколи зв’язку.

3.4 Оцінка ефективності мережних інтерфейсів навігаційного сервісу кіберфізичних систем

3.4.1 Визначення терміну розробки модульного мережного інтерфейсу НС кіберфізичних систем

Термін розробки модульного мережного інтерфейсу НС кіберфізичних систем залежить від його обсягу, трудомісткості розробки окремих етапів об’єму функцій програмних засобів в залежності від їх типів, а також кваліфікації робітників та запланованих строків, що диктують умови ринку [118,119].

В якості вихідних даних для визначення терміну розробки модульного мережного інтерфейсу НС кіберфізичних систем використовувався типовий склад етапів та збільшення норми часу на розробку ПЗ. Використовуючи міжгалузеві нормативні матеріали по розробці, виготовлені та супроводженню ПЗ обчислювальної техніки та програмуванню задач, для ЕОМ визначалося обсяг роботи над розробкою модульного мережного інтерфейсу НС кіберфізичних систем.

Використовуючи значення ПЗ автоматизованих розрахунків, що містить обсяг функцій V_0 в умовних машинних командах, визначаємо час на розробку модульного мережного інтерфейсу.

Модульний мережний інтерфейс НС кіберфізичних систем, що був розроблений відповідає аналог – програми імітаційного моделювання з V_0 = 8000 умовних машинних команд з часом на розробку $T_р$ = 385 люд./год.

Термін розробки модульного мережного інтерфейсу НС кіберфізичних систем повинна включати розробку наступних етапів:

- технічне завдання – ТЗ;

- технічного проекту – ТП;
- робочого проекту – РП;
- впровадження – ВП.

Термін розробки модульного мережного інтерфейсу НС кіберфізичних систем, що був розроблений визначався по кожному етапі на основі трудомісткості аналога з урахуванням ступені новизни, складності розробки та ступеню використання розробки стандартних модулів цього інтерфейсу на основі формул:

$$T_{тз} = T_р^{а} \times П_{В1} \times К_н, \quad (3.1)$$

$$T_{тп} = T_р^{а} \times П_{В2} \times К_н, \quad (3.2)$$

$$T_{рп} = T_р^{а} \times П_{В3} \times К_н \times К_т, \quad (3.3)$$

$$T_{вп} = T_р^{а} \times П_{В4} \times К_н, \quad (3.4)$$

де $T_р^{а}$ – збільшена норма на розробку аналога ПЗ, люд./год., яка коригувалася поправочним коефіцієнтом, що враховує умови розробки модульного мережного інтерфейсу, тобто в умовах комп'ютера, КК = (0,8);

$П_{В1}$ – питома вага і-го етапу розробки модульного мережного інтерфейсу;

$К_т$ – поправочний коефіцієнт, що враховує ступінь використання в розробці типових модульних мережних інтерфейсів НС кіберфізичних систем;

$К_н$ – поправочний коефіцієнт, що враховує ступінь новизни.

Ступінь новизни – Б. Код ступені новизни – Значення $К_н$ – Б (1). Ступінь обсягу реалізованих функцій – значення $К_т$ – 83% (0,6).

Математичного описуємо трудомісткості розробки ПЗ автоматизованих розрахунків представлено нами у формулах розрахунку трудомісткості розробки ПЗ.

Математичного описуємо термін розробки спеціалізованого мережного інтерфейсу НС кіберфізичних систем, при $1 \le i, j, k \le n$:

$$T_{кк} = 1 \times N_{пз} (люд./год.), \quad (3.5)$$

$$T_{нк} = 0{,}15 \times N_{пз} (люд./год.), \quad (3.6)$$

$$t_{спец.МІ}(j, j) \le t_{спец.МІ\,k}(i, j) \le 0, \quad (3.7)$$

$$t_{\text{спец.МІ}}(j,j)+t_{\text{спец.МІ}}(j,j)-(t_{\text{спец.МІ}}(i,j)+t_{\text{спец.МІ}}(i,j))\geq\frac{\gamma_i}{(1-\gamma_j)\times(n-1)}*V_j\,, \quad (3.8)$$

Математичного описуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем:

$$t_{\text{ММІ}}(j,j)\geq t_{\text{ММІ }k}(i,j)\geq 0\,, \quad (3.9)$$

Математичного описуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем з використанням вдосконаленого методу на основі принципу наслідування:

$$t_{\text{ММІН}}(j,j)+t_{\text{ММІН}}(j,j)-(t_{\text{ММІН}}(i,j)+t_{\text{ММІН}}(i,j))\geq\frac{\delta_i}{(1-\gamma_j)\times(n-1)}*V_j\,, \quad (3.10)$$

Математичного описуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем з використанням вдосконаленого методу на основі принципу інстанціювання:

$$t_{\text{ММІІ}}(j,j)+t_{\text{ММІІ}}(j,j)-(t_{\text{ММІІ}}(i,j)+t_{\text{ММІІ}}(i,j))\geq\delta_i/(n-1)\times V_j\,, \quad (3.11)$$

Математичного описуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем з використанням вдосконаленого методу на основі принципу використання:

$$t_{\text{ММІВ}}(j,j)+t_{\text{ММІВ}}(j,j)-(t_{\text{ММІВ}}(i,j)+t_{\text{ММІВ}}(i,j))\geq\gamma_i/(n-1)\times V_j\,, \quad (3.12)$$

Термін розробки модульного мережного інтерфейсу НС кіберфізичних систем в місяцях розрахуємо по залежності:

$$t_{пл.}=\frac{\sum_{i=1}^{5}t_{ij}}{8\times 360\times 0{,}7/12}\,, \quad (3.13)$$

де $\sum_{i=1}^{5}t_{ij}$ – загальний термін розробки модульного мережного інтерфейсу, годин;

8 - тривалість робочого дня, годин;

0,7 - коефіцієнт переводу в робочі дні;

12 - дванадцять місяців в році.

t_{ij} - трудомісткість j-го виду робіт по і-му етапу.

3.4.2 Визначення часу реакції модульного мережного інтерфейсу НС кіберфізичних систем

Часу реакції модульного мережного інтерфейсу НС кіберфізичних систем було визначено та розраховано, через запити від клієнта до сервера у безпровідній мережі. Клієнти в мережі представлялися, як одно канальні СМО з нескінченною буферною пам'яттю, для визначення середнього числа запитів від клієнта до сервера, що знаходилися в черзі, середнього числа зайнятих каналів, середнього часу очікування запиту в черзі, середнього часу перебування запиту в мережі використаються формули 3.14 - 3.17 відповідно [118,119].

$$\bar{r} = \frac{p^2}{1-p}, \tag{3.14}$$

$$\bar{k} = \frac{p}{1-p}, \tag{3.15}$$

Математичного описуємо час реакції спеціалізованого мережного інтерфейсу НС кіберфізичних систем:

$$t_{оч.} = \frac{p^2}{\lambda(1-p)}, \tag{3.16}$$

$$t_{спец.МІ} = \frac{1}{\mu(1-p)}, \tag{3.17}$$

де $p = \frac{\lambda}{\mu}, \mu = \frac{1}{t_з}$.

Для визначення середнього числа запитів, що знаходяться в черзі, середнього числа зайнятих каналів, середнього часу очікування запиту в черзі, середнього часу перебування запиту в мережі використаються формули 3.18 - 3.22 відповідно.

$$\bar{r} = \frac{p^{n+1} \times P_0}{n! \times n \times \left(1 - \frac{p}{n}\right)^2}, \tag{3.18}$$

$$\bar{k} = \bar{r} + \bar{z}, z = \bar{\mu} = p \Rightarrow \bar{k} = \bar{r} + p, \tag{3.19}$$

Математичного описуємо час реакції модульного мережного інтерфейсу НС кіберфізичних систем:

$$t_{оч.} = \frac{p^n \times P_0}{n \times \mu \times n! \times \left(1 - \frac{p}{n}\right)^2} = \frac{\bar{r}}{\lambda} \ , \tag{3.20}$$

$$P_0 = \left(\sum_{i=0}^{4} \frac{p^i}{i!} + \frac{p^5}{4!(4-p)} \right)^{-1} , \tag{3.21}$$

$$t_{\text{ммі}} = \bar{t}_{\text{оч.}} + \bar{t}_{\text{обсл.}} = \frac{\bar{r}}{\lambda} + \frac{1}{\mu} , \tag{3.22}$$

Необхідно було зробити розрахунок, середнього числа запитів, що перебувають в черзі, середнього числа зайнятих каналів, середнього часу очікування запиту в черзі,часу перебування запиту в мережі. Інтенсивність обробки інформації для відповіді на запити, визначалася за формулою:

$$\mu = \frac{1}{t_{\text{затр.}}} , \tag{3.23}$$

Інтенсивність обробки інформації також однакова для всіх вдосконалених методів розробки модульного мережного інтерфейсу НС кіберфізичних систем.

Математичного описуємо час реакції модульного мережного інтерфейсу НС кіберфізичних систем при реалізації його з використанням вдосконаленого методу розробки на основі принципу наслідування:

$$t_{оч.} = \frac{p^n / P_0}{n \times \mu \times n! \times \left(1 + \frac{p}{n}\right)^2} = \frac{r}{\lambda} , \tag{3.24}$$

$$P_0 = \left(\sum_{i=0}^{2} \frac{p^i}{i!} + \frac{p^5}{4!(4 \times p)} \right) , \tag{3.25}$$

$$t_{\text{ммін}} = t_{\text{оч.}} + \bar{t}_{\text{обсл.}} = \frac{r}{\lambda} + \frac{1}{\mu} , \tag{3.26}$$

Математичного описуємо час реакції модульного мережного інтерфейсу НС кіберфізичних систем при реалізації його з використанням вдосконаленого методу розробки на основі принципу інстанціювання:

$$t_{оч.} = \frac{p^n - P_0}{(n+\mu) - n! \times \left(1 - \frac{p}{2n}\right)^2} = \frac{\bar{r}}{\lambda}\ , \tag{3.27}$$

$$P_0 = \left(\sum_{i=0}^{4} \frac{p^i}{i!} - \frac{p^3}{4!(2-p)} \right)^{-1}\ , \tag{3.28}$$

$$t_{\text{ммії}} = \bar{t}_{\text{оч.}} + \bar{t}_{\text{обсл.}} = \frac{\bar{r}}{2\lambda} + \frac{1}{\bar{\mu}}\ , \tag{3.29}$$

Математичного описуємо час реакції модульного мережного інтерфейсу НС кіберфізичних систем при реалізації його з використанням вдосконаленого методу розробки на основі принципу використання:

$$t_{оч.} = \frac{p^n \times P_0}{(2n \times \mu) \times n \times \left(1 \_ \frac{p}{n}\right)^3} = \frac{\bar{r}}{\lambda}\ , \tag{3.30}$$

$$P_0 = \left(\sum_{i=0}^{6} \frac{p^i}{i!} + \frac{p^2}{4!(7-p)} \right)^{-1}\ , \tag{3.31}$$

$$t_{\text{ммів}} = \bar{t}_{\text{оч.}} + \bar{t}_{\text{обсл.}} = \frac{\bar{r}}{\lambda} + \frac{1}{\bar{\mu}}\ , \tag{3.32}$$

Отже, маючи математичний опис двох критерії ефективності, а саме час реакції та термін розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі методів розробки з використанням трьох принципів, дозволить порівняти реалізацію вдосконалених методів, яка наведена у 4 розділі дисертаційної роботи.

3.5 Висновки до розділу

1. В результаті проведених досліджень виявлено особливості існуючих підходів розробки мережних інтерфейсів у навігаційних сервісах

кіберфізичних систем, які є причиною низьких характеристик мережних інтерфейсів.

2. Під час проведення досліджень було виявлено простіший метод реалізації вимог запропонованої концепції автоматизованого синтезу мережного інтерфейсу у навігаційних сервісах кіберфізичних систем щодо функцій базового рівня. Суть такого методу в тому, щоб синтезувати структуру мережного інтерфейсу спочатку на загальних засадах (без винятків для функцій базового рівня), а потім коригувати синтезовані структури, принципи розробки, яких відрізняються від загальноприйнятих.
3. Вдосконалені методи покращення ефективності мережних інтерфейсів у навігаційних сервісах кіберфізичних систем передбачають оптимізацію структури мережного інтерфейсу за кількістю структурних елементів, що виконують однакові функціональні операції.
4. Розроблено принципи оцінки терміну розробки мережного інтерфейсу у навігаційних сервісах кіберфізичних систем, що дозволяють на етапах синтезу мережних інтерфейсів врахувати багатоваріантність реалізацій і запропонувати різні варіанти структур мережних інтерфейсів разом з прогнозами їх характеристик і рекомендаціями до застосування.
5. На основі синтезованої системи рівнянь запропоновано математичний опис терміну розробки та часу реакції модульних мережних інтерфейсів у навігаційних сервісах кіберфізичних систем.

РОЗДІЛ 4

РЕАЛІЗАЦІЯ ВДОСКОНАЛЕНИХ МЕТОДІВ РОЗРОБКИ МЕРЕЖНИХ ІНТЕРФЕЙСІВ НАВІГАЦІЙНИХ СЕРВІСІВ КІБЕРФІЗИЧНИХ СИСТЕМ

У практичній реалізації функцій мережного інтерфейсу необхідно було ідентифікувати класи і об’єкти спочатку по властивостях, необхідних для реалізації загальної частини модульного мережного інтерфейсу. В іншому випадку необхідно виділити і обробити структури і типи поведінки об’єктів, які були різними і реалізувати їх в спеціальній частині модульного мережного інтерфейсу. З використанням вперше запропонованого методу розробки мережного інтерфейсу НС кіберфізичних систем, були вирішені поставленні завдання, через класи задач та їх специфіку, які у випадку НС кіберфізичних систем зводяться до створення мережних інтерфейсів, а саме:

1. Багато типів пристроїв, які потрібно під’єднати до НС кіберфізичних систем:
   1.1 Мережні інтерфейси зв’язку пристроїв з базою даних (БД) НС кіберфізичних систем;
   1.2 Специфіка - один механізм запису в БД, багато протоколів зв’язку з пристроями.
2. Багато форм інтерфейсу користувача (карти, звіти, адміністрування, допомога, інше):
   2.1 Програмні мережні інтерфейси зв’язку інтерфейсу користувача з навігаційним сервером обробки інформації;
   2.2 Специфіка - один механізм читання з БД багато форм представлення інформації.
3. Багато сутностей в НС кіберфізичних систем і відповідних їм таблиць в БД:

3.1 Програмні мережні інтерфейси зв’язку навігаційного сервера з БД;

3.2 Специфіка - один механізм доступу до БД багато таблиць, для обробки.

З цього вище наведеного переліку бачимо, що класи задач зводяться до створення мережних інтерфейсів, які мають об’єктну структуру. Об’єктна структура формується з таблиць БД. Працюється з нею, а не з таблицею БД, тому що ця структура синхронізувалася з таблицями за допомогою стандартної бібліотеки команд. Маючи класи задач та їх специфікацію мені потрібно було розробити програмну реалізацію мережного інтерфейсу. Для програмної реалізації поставлених класів задач потрібно було розробити алгоритм програмної реалізації. Предметна галузь, а саме мережний інтерфейс НС кіберфізичних систем та специфіка його розробки, до прикладу час на розробку і ефективність цього мережного інтерфейсу залежатиме від об’єму роботи при його розробці.

Пропонується зменшити об’єм роботи при розробці НС кіберфізичних систем. Це зменшення здійснювалося шляхом розділення мережних інтерфейсів НС на класи задач, а саме:

1. Мережний інтерфейс взаємодії з пристроями. Для роботи з пристроями необхідно розробити сервер зв’язку з ними, який реалізувався мережним інтерфейсом. Цей сервер зв’язку працював лише з одним типом пристроїв, тобто для кожного типу пристрою були потрібні свої сервери зв’язку. Оскільки цих типів існує багато, відповідно багато серверів. Отже, трудоємність виконання задачі зростає відповідно до кількості типів пристроїв. Для того, щоб зменшити цю трудоємність було запропоновано розбити мережний інтерфейс на 2 частини:

1) універсальну, яка містить функції, які будуть написані один раз для всіх типів пристроїв;
2) спеціальну, яка містить функцій специфічні для кожного типу пристрою.

2. Мережний інтерфейс для взаємодії між інтерфейсом користувача і НС кіберфізичних систем. Важливу увагу, зверталося на таку форму інтерфейсу користувача, як звіти. Відслідковувалася така закономірність: «Чим коректніше працював НС кіберфізичних систем, тим він продуктивніший та ефективніший. А кожен звіт НС кіберфізичних систем, це нова форма інтерфейсу користувача. Відповідно чим коректніше працював НС кіберфізичних систем, тим більше роботи програмістам. Було запропоновано спростити цю роботу, шляхом розбиття мережного інтерфейсу на 2 частини універсальну та спеціальну. Взаємодію інтерфейсу користувача з навігаційним сервером реалізовувалося через універсальну частину, яка один раз програмувалася, а обробку даних для кожного інтерфейсу користувача треба було реалізовувати для кожного окремо. Елементів інтерфейсу користувача було багато і тродуємності його буде зростати тому, що користувачі вимагали задоволення їх потреб, що призводило до створення нових інтерфейсів користувача. Отже, трудоємність виконання задачі зростала відповідно до кількості елементів інтерфейсу користувача. Для того, щоб зменшити цю трудоємність, було запропоновано розбити мережний інтерфейс на 2 частини.

3. Мережний інтерфейс для взаємодії НС кіберфізичних систем з базою даних. Для того, щоб якісно було будувати звіти необхідні дані, які зберігалися в БД. Чим більша кількість сутностей в НС кіберфізичних систем, тим більше було таблиць в БД. Наприклад, робота з водіями. Наприклад, була потрібна таблиця в БД про водіїв транспортних засобів. Крім того, робота з водіями передбачала графік робочого часу, записи про це були розміщені ще в одній таблиці в БД. Також, необхідна таблиця в БД, яка зв’язує графік робочого часу водія з конкретним транспортним засобом. Тобто, для обробки інформації про водіїв необхідно було 3 таблиці в БД, а для повної обробки в завершеній системі потрібно було 6 таких таблиць. І тут виникала проблема написати механізми взаємодії з цими всіма сутностями, які є в БД. Мережний інтерфейс для зв’язку з БД був також зміст розбити на універсальну та спеціальну частини, тоді зменшувався об’єм написання коду і спеціальна частина, яка була

зв’язана безпосереднього з даними, які є в конкретній таблиці і цю частину не можна «універсалізувати» і її доводилося робити для кожної табиці в БД.

У системі, яка використовувалася, таблиць в БД було багато для кожної потрібно створити сутність і зв’язати її з інтерфейсом до БД. Але всі сутності виконували одне і теж, записували, читали, змінювали і видаляли відповідні таблиці. Тобто, була загальна частину яка для всіх однакова, тобто вона ставала універсальною, а саме: зв'язок з БД, посилання запиту, синтаксичний аналіз відповіді і т.д. А спеціальна частина це структура таблиці БД, вона для кожної таблиці своя, тому засоби було добре універсалізувати, а дані специфікувати.

4.1 Реалізація модульного мережного інтерфейсу на основі принципу наслідування

Мережний інтерфейс зв’язку НС кіберфізичних систем з інтерфейсом користувача - це реакція сервера на дії користувача, тобто це механізм за допомогою, якого дії користувача переносяться на сервер. Відповідно до того, які сервер надає відповіді - перебудовувався інтерфейс користувача. В НС кіберфізичних систем існувало багато форм представлення інтерфейсу користувача. Для зменшення об’єму роботи над розробкою цього інтерфейсу необхідно було мати один мережний інтерфейс, для всіх форм представлення інтерфейсу користувача. Його специфіка така, що форм представлення інформації багато: з відображенням карт; таблиць або звітів; редагування налаштувань; відображення тексту допомоги, тощо. Вся робота з інтерфейсом користувача зводилася до того, що інтерфейс користувача надсилає запит на отримання даних, а сервер отримував дані з БД, обробляв їх і передавав інтерфейсу користувача. Кожна форма інтерфейсу користувача, вимагала своїх наборів даних. Відповідно, виникало завдання надіслати запит і отримати відповідь, яка була загальною для всіх форм інтерфейсу користувача з активним вмістом. В загальну частину на першому етапі розробки мережного інтерфейсу попадала взаємодія «запит-відповідь», а на другому етапі крім неї, ще формат виводу інформації. Набір даних і форма представлення інформації,

які були потрібні для розробки інтерфейсу користувача, були спеціальною частиною.

Взаємодія «запит-відповідь» ініціювалася користувачем в своєму інтерфейсі, але вона не приводила до завантаження інтерфейсу користувача, він вже знаходився на клієнтському комп'ютері і перебудовувався відповідно до отриманих даних. Тобто, інтерфейс користувача завантажувався тільки один раз. Дії користувача призводили до переформатування інтерфейсу користувача. Це переформатування відбувалося за рахунок даних які надходили з сервера, відповідно мережний інтерфейс повинен був, в найпростішому випадку, передати запит інтерфейсу користувача на сервер, отримати з сервера відповідь і перебудувати зовнішній вигляд інтерфейсу користувача. Специфіка полягала в тому, що є різні форми представлення одної і тої ж інформації в інтерфейсі користувача. Тому для кожної форми представлення інформації разом з даними передавалося форматування виводу. Тобто, яким чином були відформатовані ті дані, які передавалися з сервера. Наприклад, у звіті у вигляді таблиці або графіка, на карті у вигляді лінії, в адміністративній частині у вигляді параметрів. Тобто, варіанти представлення одної і тої інформації різні, але інтерфейс користувача один і той самий. Тому доцільно було мати 3-ьох рівневий підхід на основі принципу наслідування, де: 1-им рівнем є взаємодія «запит-відповідь»; 2-им рівнем взаємодія при якій, ще виконувалося форматування виводу; 3-ім рівнем набір даних для відповідного звіту, які передавали із взаємодією і форматуванням виводу, тобто не сам інтерфейс користувача, а дані для нього.

Отримувалася ієрархія наслідування, коли форма представлення інтерфейсу відносилася до типу інтерфейсу користувача. Цей принцип полягав в наступному, базовий клас інтерфейсу користувача наслідують класи інтерфейсів карти, звітів та інші. Їх у свою чергу наслідували інтерфейси адміністрування водіїв, тощо. Наприклад, інтерфейс звітів про пристрій і інтерфейс звітів про групу пристроїв наслідували класи звіту по витраті палива конкретним пристроєм або звіту про витрати палива групою пристроїв. Тобто,

утворювалася ієрархія наслідування, де кожен наступний клас наслідував попередній з додаванням спеціальної частини мережного інтерфейсу, відповідно до типу інтерфейсу користувача та засобів взаємодії з ним, та розширював собою універсальну частину мережного інтерфейсу. Адже, для тих інтерфейсів, що його наслідували, це вже універсальна частина мережного інтерфейсу, а наступний клас розширював спеціальну частину мережного інтерфейсу і це розширення в інших класах лише поглиблювалося.

В даному випадку, універсальна частина мережного інтерфейсу генерувала форму представлення інтерфейсу користувача і механізм його взаємодії з сервером. Спеціальна частина мережного інтерфейсу, формувала безпосередньо набір даних для інтерфейсу користувача. Крім того, в кожного з видів інтерфейсу користувача була загальна форма представлення інформації та форма представлення необхідної інформації (карта, звіт, тощо), тобто поглиблювалися спеціалізації при переході від загальної форми представлення інтерфейсу користувача до необхідного. У цьому випадку це аналог ієрархії наслідування, тобто для створення даного мережного інтерфейсу був розроблений модульного мережного інтерфейсу на основі принципу наслідування. Розроблений мережний інтерфейс на основі принципу наслідування містив: вхідні дані - список функцій універсальної і спеціальної частин; вихідні дані - абстрактна реалізація мережного інтерфейсу. Було проведене групування функцій універсальної частини для утворення модулів, які були представлені класами (Рисунок 4.1).

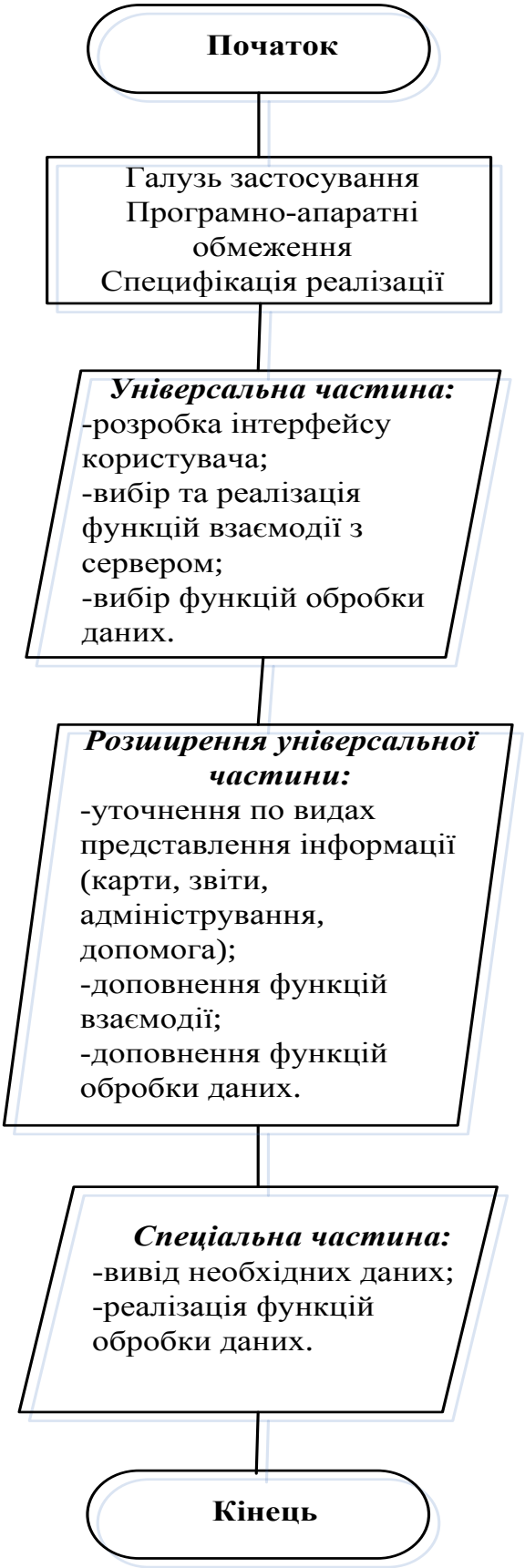


Рисунок 4.1- Графічне подання модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування

Формувався список функцій універсальної частини, які входили у кожний із модулів і створювалися абстрактні функції. Потім проводилося групування функцій спеціальної частини для утворення модулів, які були представлені класами та формувався список функцій спеціальної частини, які входили у кожен із модулів і створювалися абстрактні функції. Наступним кроком було створення нащадків з абстрактних похідних класів спеціальних частин, для кожного конкретного мережного інтерфейсу. Далі перевірялася працездатність абстрактної системи. Як наслідок було отримано, абстрактну програмну реалізацію мережного інтерфейсу, яку дальше булор реалізовано і відповідно функції ставали не абстрактними, а практичними.

Розроблений модульний мережний інтерфейс на основі принципу наслідування використовувався в НС «ZITtrack» кіберфізичних систем. Користувач здійснював доступ до цього НС кіберфізичних систем через ВЕБ-інтерфейс, який інтерактивно формувався на основі дій користувача та інформації з бази даних НС кіберфізичних систем. Інтерфейс користувача був реалізований на мовах програмування Java, HTML, JavaScript, CSS. При цьому інтерфейс користувача складався з шаблону оформлення виконаного за технологією Java Server Page (JSP) і вмісту, який був сформований інтерактивно. Форми представлення інформації, що складали інтерфейс користувача, було розділено на декілька категорій. Форми представлення інформації кожної категорії були побудовані згідно однакового принципу. Наприклад, всі звіти виглядали подібно, але містии різну інформацію.

Інтерфейс користувача завантажувався в браузер користувача у вигляді ВЕБ-сторінки, яка інтерактивно взаємодіяв з сервером НС кіберфізичних систем за допомогою технології Ajax, згідно якої розроблений мережний інтерфейс взаємодії сервера з інтерфейсом користувача, тобто використовувалася стандартна клієнт-серверна взаємодія. Мережні інтерфейси одного класу були розділені на універсальну частину, яка включала мережну взаємодію за допомогою JavaScript і форматування виводу за допомогою CSS, та спеціальну частину, яка включала дані для відповідного інтерфейсу

користувача. Структура класів мережного інтерфейсу користувача в НС «ZITtrack» був розроблений з використанням принципу наслідування наведений на рисунку 4.2.

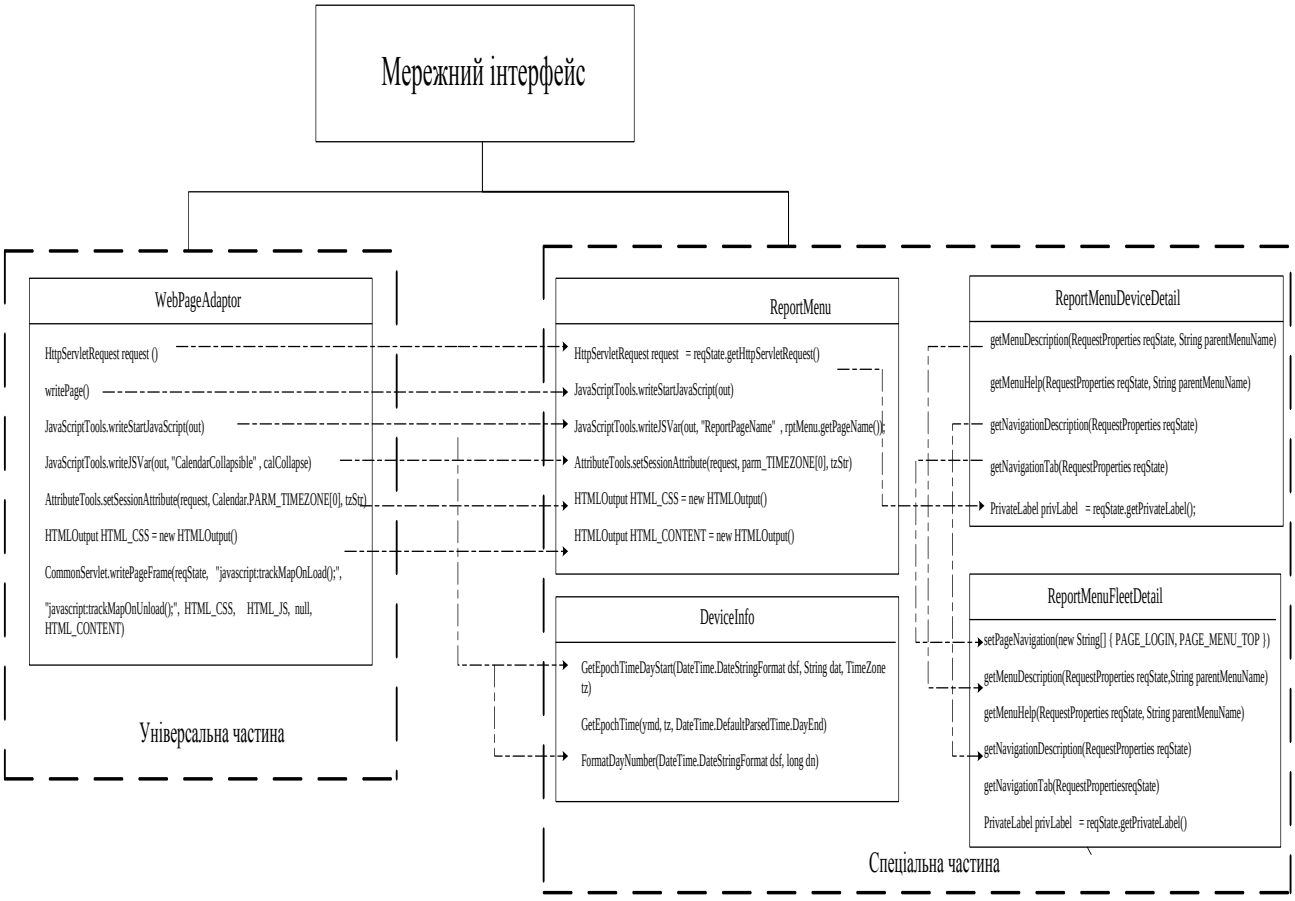


Рисунок 4.2 - Схема структури класів мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування

Універсальна частина мережного інтерфейсу реалізувалася в класі WebPageAdaptor, а спеціальні частини модульного мережного інтерфейсу для кожної інформаційної форми були описані у відповідних похідних класах. Наприклад, інформаційна форма з інформацією про навігаційні пристрої формувалася класом DeviceInfo, який наслідував клас WebPageAdaptor. Всі інші класи мережного інтерфейсу користувача, теж наслідували базовий клас WebPageAdaptor. Враховуючи це визначався базовий клас, знаючи функції та змінні, які наслідвалися його підкласами.

Універсальна частина була реалізована у вигляді класу WebPageAdaptor класами (Рисунок 4.3).

```
class WebPageAdaptor {
isValidID(PrivateLabel privLabel, String id);
EncodeURL(RequestProperties reqState, URIArg url);
URIArg MakeURL(String baseURI);
FormRow_ComboBox(String key, boolean editable, String desc, String selKey,
ComboMap map, String oncha);
Property(PrivateLabel privLabel, String key, String dft);
Form_TextField(String name, boolean editable,  String value, int size, int maxLen);
   }
```

Рисунок 4.3 - Приклад програмної реалізації класу WebPageAdaptor

Спеціальна частина мережного інтерфейсу була реалізована, наприклад в класі DeviceInfo (Рисунок 4.4).

```
class DeviceInfo extends WebPageAdaptor {
editUniqID = privLabel.hasWriteAccess(currUser,
getAclName(_ACL_UNIQUEID));
editServID = privLabel.getBooleanProperty
(PrivateLabel.PROP_DeviceInfo_allowEditServerID,EDIT_SERVER_ID) &
selectURL  = DeviceInfo.this.encodePageURL(reqState);
newURL     = DeviceInfo.this.encodePageURL(reqState);
ComboMap ppList = new ComboMap(reqState.getMapProviderPushpinIDs());
showAssgnUsr = privLabel.getBooleanProperty
(PrivateLabel.PROP_DeviceInfo_showAssignedUserID,SHOW_ASSIGNED_USE);
Map < String,DeviceCmdHandler > DCMap = new HashMap <
String,DeviceCmdHandler >();
devCmdPackage = DeviceCmd_gtsdmtp.class.getPackage().getName();
}
```

Рисунок 4.4 - Приклад програмної реалізації класу DeviceInfo

Модульний мережний інтерфейс був побудований на основі принципу наслідування, а саме спеціальна частина наслідує універсальну частину модульного мережного інтерфейсу.

4.2 Реалізація модульного мережного інтерфейсу на основі принципу інстанціювання

У навігаційному сервісі кіберфізичних систем виконувалося багато паралельних операцій доступу до БД одночасно, а саме:

1. Мережні інтерфейси зв’язку з пристроями надсилали дані в БД.
2. Мережні інтерфейси зв’язку з інтерфейсами користувачів отримували інформацію з БД.

Для зв’язку з кожним пристроєм створювався потік керування і відповідно цей потік надсилав запис в БД. Кожен користувач викликав, якусь форму на екран таким чином створювався потік керування для отримання інформації з БД. В результаті в НС кіберфізичних систем існували тисячі потоків керування для роботи з БД. Для того, щоб сервер БД не припинив свою коректну роботу був необхідний єдиний мережний інтерфейс до сервера БД. Було запропоновано синхронізований мережний інтерфейс. Тобто, коли існувало багато потоків керування, для того щоб вони між собою не конфліктували потрібно було їх синхронізовувати. СУБД керувала цими потоками сама, при цьому вона створювала чергу в яку записувала запити від користувачів відповідно, над якими виконувалися маніпуляції з потоками даних. Але виникала проблема в тому, що ця черга мала обмежений розмір, тобто, як тільки розмір переповнювався, робота навігаційного сервісу кіберфізичних систем призупинялася. Щоб не виникало подібних ситуацій, потрібно було організувати синхронізований доступ до потоків керування, коли вони виконувалися чітко один за одним.

Реалізувався цей доступ наступним чином. Реалізувався об’єкт в пам’яті, який відображав таблицю БД і працював з ним, а вміст БД не мав значення. Це відбувалося наступним чином, один раз створювався об’єкт в пам’яті, а далі

здійснювалася його модифікація. У свою чергу, об’єкт синхронізувався з таблицею БД шляхом відкладеного доступу для того, щоб зменшити навантаження на сервер БД, тобто коли ця структура могла отримати доступ до БД чи, тоді коли в БД накопичилася критична кількість змін. Узагальнення цієї взаємодії з БД було зроблено за допомогою шаблонів, які вміють працювати з узагальненими даними, тобто дані не були конкретизовані. Коли створювався об’єкт в пам’яті - описувався перелік даних, які він мав містити і таким чином, здійснювалося інстанціювання цього шаблону самими даними. В результаті об’єкт на сервері вмів ці дані записувати, читати і так далі. Тому загальна частина мережного інтерфейсу була побудована на основі шаблона, а прикладні класи інстанціювали цей шаблон структурою об’єктів.

Об’єкт, який відображав одну із сутностей, наприклад користувача, взаємодіяв з таблицями БД. Тому таким чином, було розроблено мережний інтерфейс навігаційного сервісу кіберфізичних систем, для того щоб об’єкт типу «user» синхронізовано з іншими об’єктами взаємодіяв з таблицею «user» БД. Практично це реалізовувалося наступним чином, один клас взаємодії і інстанціювався його конкретними структурами заданого об’єкту. Наприклад, об’єкт «user» мав атрибути: account_id, user_id, name, login, password. Ці специфікації (атрибути) і були структурою таблиці БД. Таким чином, інстанціювавши об’єкт мережного інтерфейсу об’єктом «user» отримувалося доступ до таблиці БД. Програмно це реалізовувалося класом шаблону мережного інтерфейсу для роботи з таблицями БД, який інстанціювався структурою конкретного об’єкта в НС кіберфізичних систем і навпаки, структура БД повинна відображатися в об’єкті в пам’яті. Реалізувався механізм синхронізованих операцій з БД, тобто запити оброблялися не одночасно. Цю задачу було вирішено створивши клас, який здійснював узагальнені операції доступу до БД, а саме операції: вставки нових даних, читання тих даних що є, модифікація і видалення даних, які виконувалися незалежно від того, які дані в даний момент оброблялися.

Універсальна частина мережного інтерфейсу представлялася класом шаблона мережного інтерфейсу, а спеціальна частина відповідно, представлялася класом структури таблиці БД (Рисунок 4.5).

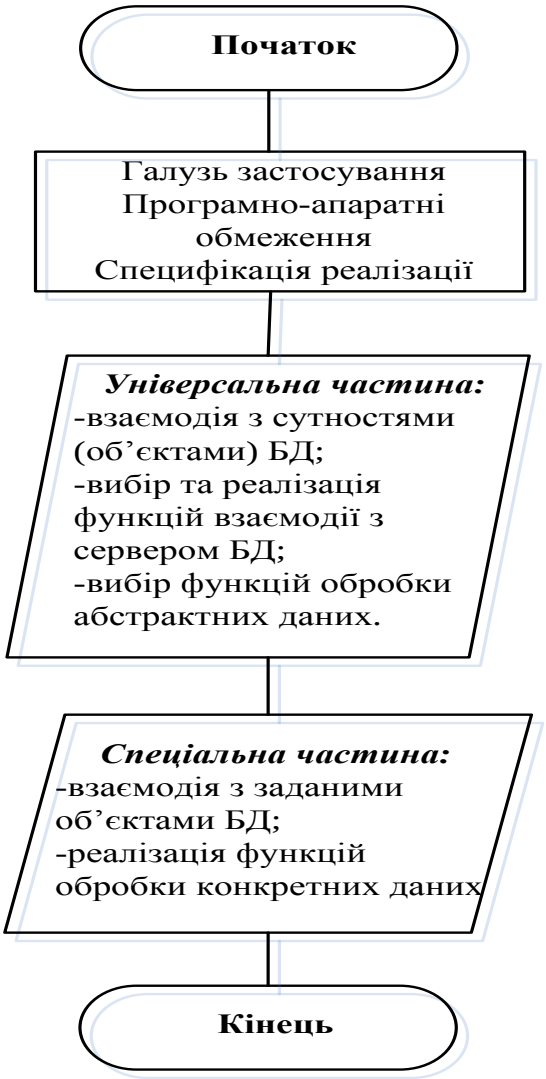


Рисунок 4.5 - Графічне подання модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання

За допомогою цього вдосконаленого методу створювалися шаблони модулів спеціальної частини мережного інтерфейсу і потім використовуючи принцип інстанціювання створювалися об'єкти спеціальної частини мережного інтерфейсу. Шаблон опирував типами даних. Суть шаблонів в тому, що всі функції і структури в ньому описані таким чином, що вони працювали з даними. А коли використовувалося безпосередньо цей шаблон, тоді

інстанціювалося конкретним типом даних і він автоматично працював з узагальненим типом даних. З точки зору, взаємодії з БД це означало що працювалося з інформацією в БД безвідносно, що це за інформація. Окремо, описувався конкретний об'єкт БД, наприклад таблицю, в якій містилася відповідна інформація і інстанціювалося цією інформацією через об'єкт БД мережний інтерфейс. В результаті мережний інтерфейс автоматично міг реалізовувати загальні операції в БД. Ці операції виконувалися безвідносно до інформації, яка модифікувалася чи шукалася. А інстанціювання було приведенням до типу даних конкретним об'єктом БД. Наприклад, таблицею БД з інформацією про пристрої, тобто автоматично модифікувалася інформація про пристрій, додавати нові пристрої, видаляти пристрої, які не використовувалися і т.д. Специфіка роботи з БД полягала в тому, що наперед невідомо з якими типами даних працюватиметься, але є стандартизовані операції для роботи з цими типами даних. Така специфіка в програмуванні описувалася так званим шаблонами. Шаблон – параметризований клас, в якому у якості параметра - тип даних. В даному випадку, типами даних були об'єкти БД, які описувалися як спеціальна частина. В результаті модульний мережний інтерфейс на основі принципу інстанціювання був реалізований так, шаблон який описував універсальну частину даного мережного інтерфейсу і класи, які описували об'єкти БД, які при використанні інстанціювали тип даних.

Модульний мережний інтерфейс на основі принципу інстанціювання було практично реалізовано через використання цього принципу. Принцип інстанціювання полягав у наступному. Сервер НС «ZITtrack» під'єднувався як клієнт до сервера баз даних. Інформація в базі даних розділена на таблиці згідно функціонального призначення. Доступ до будь-якої таблиці здійснювалося за допомогою запитів на мові SQL. При цьому запити до різних таблиць були подібними, відрізнялися лише назви таблиць і списки атрибутів, а також умови відбору результатів. Взаємодію сервера із сервером баз даних описувалося набором команд, які мали подібний вигляд, але різний зміст. Вигляд команд доступу до БД можна було вважати універсальною частиною

мережного інтерфейсу, а специфікацію таблиць, атрибутів і умов спеціальною частиною мережного інтерфейсу.

Будь-який клас шаблона інстанціювався перед використанням. Клас шаблона модульного мережного інтерфейсу на основі принципу інстанціювання створювався параметризованим. Суть реалізації модульного мережного інтерфейсу на основі принципу інстанціювання полягав в створенні шаблону доступу до БД параметром якого була відповідна таблиця БД. Мережний інтерфейс реалізувався у вигляді сервера DB, який здійснював доступ до БД. У цьому сервері було багато класів і вони всі інстанційовані. Отже, вся робота з БД інстанційована. Класами сервера будуть: DBRecord, DBFactory та інші допоміжні класи, наприклад DeviceRecord.

Практична реалізація модульного мережного інтерфейсу на основі принципу інстанціювання (Рисунок 4.6).

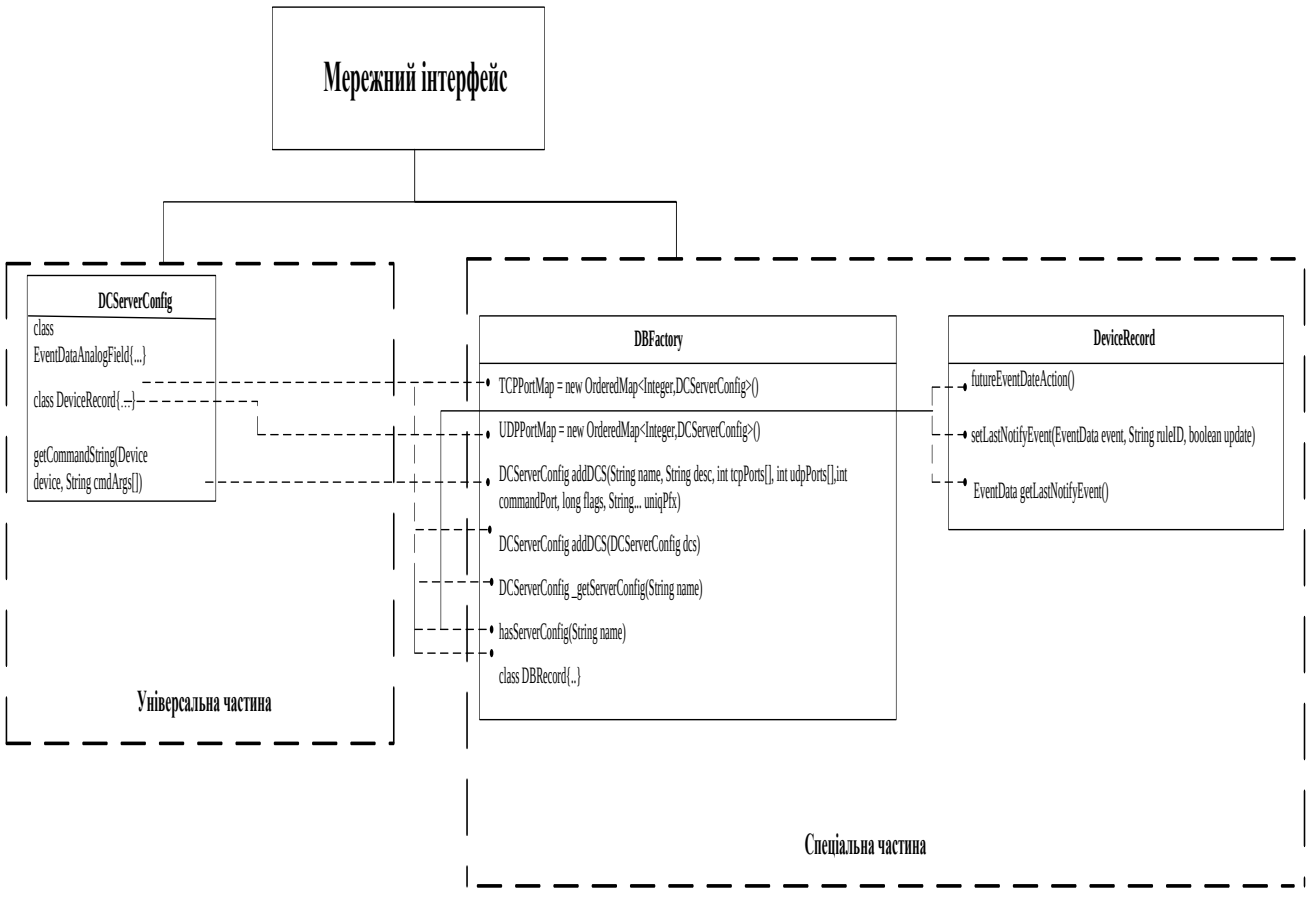


Рисунок 4.6 - Схема структури класів мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання

Як написати універсальні засоби, коли всі таблиці різні і у них різні аргументи? Для цього використовувався шаблон доступу до таблиці, який називався DBRecord. Для роботи з цим шаблоном використовувався клас DBFactory. Крім того, був набір класів кожен з яких описував конкретну таблицю і роботу з нею. Кожен засіб роботи з цими таблицями використовував об'єкт, який інстанціювався класом DBRecord, а об'єктом є об'єкт таблиці, тобто він міг сам себе викликати через той же шаблон. Тобто, використовувався шаблон таблиці та універсальні засоби, які працювали з таблицями, а коли потрібно звернутися до визначеної таблиці, то цей шаблон інстанціювався об'єктом цієї таблиці, який вже містив механізми роботи саме з цією таблицею, а не з іншою. Таким чином були узагальнені засоби і реалізації під кожну таблицю. Тобто, використовувався модульний мережний інтерфейс на основі принципу інстанціювання.

У практичній реалізації модульного мережного інтерфейсу на основі принципу інстанціювання клас DeviceRecord інстанціював клас DBRecord, Шаблон класу універсальної частини мережного інтерфейсу представлений в класі DBRecord (Рисунок 4.7).

```
class DBRecord < gDBR extends DBRecord >{
DBRecordKey < gDBR > recordKey = null;
DBRecord(DBRecordKey < gDBR > key);
DBRecordKey < gDBR > getRecordKey();
DBFactory < gDBR >
getFactory(boolean required);
DBFactory < gDBR >fact= DBFactory < gDBR >)
methGetFactory.invoke();
DBRecordKey < gDBR > recKey = this.getRecordKey();
DBSelect < gDBR > dsel = new DBSelect < gDBR >(recKey.getFactory());
}
```

Рисунок 4.7 - Приклад програмної реалізації класу DBRecord

Спеціальна частина модульного мережного інтерфейсу представлена в класі DBFactory, який інстанціювався DBRecord (Рисунок 4.8).

```
class DBFactory < gDBR extends DBRecord >implements DBRecordListener < gDBR >{
Class < gDBR > rcdClass = null;
Class < ? extends DBRecordKey < gDBR >> keyClass = null;
DBRecordListener < gDBR > recordListener = null;
setRecordListener(DBRecordListener < gDBR > rcdListener);
recordWillInsert(gDBR rcd);
}
```

Рисунок 4.8 - Приклад програмної реалізації класу DBRecord.

Специфіка даного підходу до розробки мережного інтерфейсу полягала в принципі інстанціювання. Модульний мережний інтерфейс будувався на основі принципу інстанціювання, а саме спеціальна частина інстанціювалася універсальною частиною мережного інтерфейсу.

4.3 Реалізація модульного мережного інтерфейсу на основі принципу використання

Всі навігаційні пристрої передавалися дані, які повинні були занесені в БД, але в кожного свій набір команд і даних. Як зазначено вище, був необхідний один мережний інтерфейс зв’язку з пристроями багатьох типів, для всіх цих типів пристроїв. Для цього, який дозволяв зменшити об’єм роботи по написанню коду розроблено мережного інтерфейс для зв’язку з пристроями. Для цього запропонований модульний мережний інтерфейс на основі принципу використання. Для практичної реалізації цього мережного інтерфейсу був потрібний поділу мережного інтерфейсу НС кіберфізичних систем для зв’язку з пристроями на дві частини універсальну і спеціальну. Мережний інтерфейс

для зв’язку з пристроями, оскільки пристрої апаратні і на них не можна було впливати, вони працювали і мали свою специфіку. Ця специфіка полягала в наступному, що в НС кіберфізичних систем є таблиці БД, які було потрібно заповнити даними отриманими з різнотипних пристроїв з їхніми протоколами зв’язку, адже ці пристрої передавали різнотипні дані в довільному порядку. Специфіка задачі вимагала реалізувати стандартний клас, який записував дані в таблицю БД. При цьому виконувалися функції: перевірки даних на коректність, визначення і врахування обмежень. Виходячи з цього об’єкт, який записував дані в таблицю БД, повинен був отримувати інформацію від об’єкта, який взаємодіяв і безпосередньо приймав дані від пристрою по заданому пристроєм протоколу. Отже, об’єкт який взаємодіяв з БД використовував об’єкт, який відповідав за протоколи зв’язку з пристроями, тобто об’єкт, який записував дані в БД і взаємодіяв з пристроями, але не напряму, а через об’єкт, який відповідав за протокол зв’язку з пристроями. Для реалізації цього використовувався модульний мережний інтерфейс на основі принципу використання. А також, програмно реалізований мережний інтерфейс до БД для запису даних з цих пристроїв, який не залежав ні від пристроїв ні від їх протоколів зв’язку, а залежав від того, які дані потрібно було отримати з цих пристроїв. А для того, щоб отримати дані які мережний інтерфейс поміщає в таблицю БД, він для цього використовував об’єкт з протоколом зв’язку з навігаційним пристроєм. Оскільки він використовував об’єкт, було запропоновано модульний мережний інтерфейс на основі принципу використання.

Практична реалізація модульного мережного інтерфейсу на основі принципу (Рисунок 4.9).

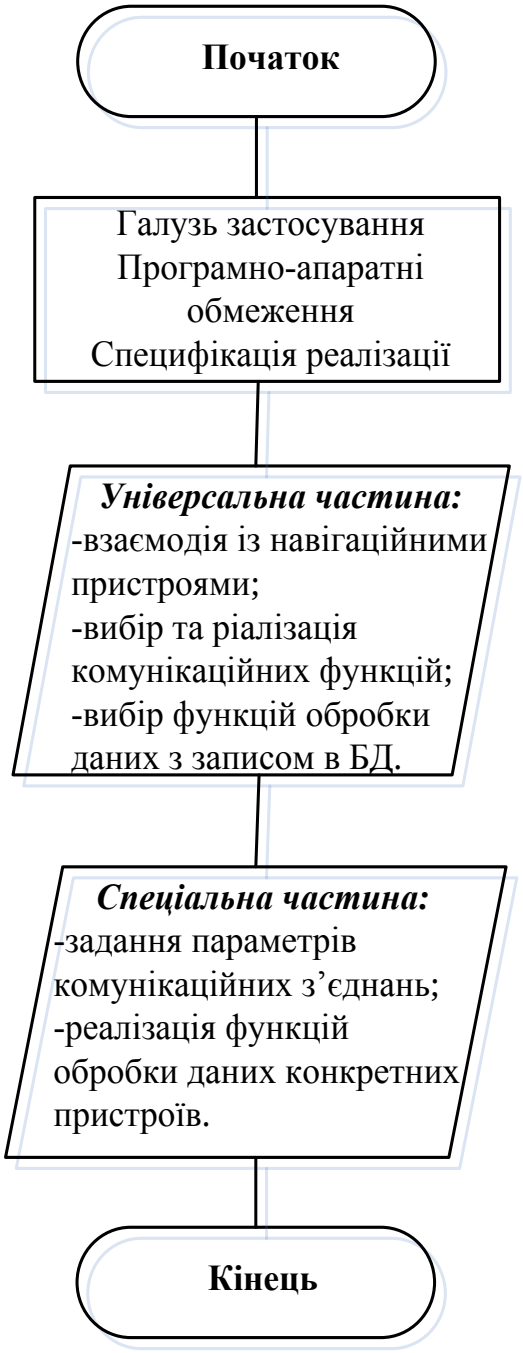


Рисунок 4.9 - Графічне подання модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу використання

Універсальна частина модульного мережного інтерфейсу на основі принципу використання є такою ж, як і у модульних мережних інтерфейсів на основі принципів наслідування і інстанціювання. А спеціальна відрізнялася тим, що у ній створювалися абстрактні модулі та об'єкти для кожного конкретного мережного інтерфейсу. Модульний мережний інтерфейс був

побудований на основі принципу використання реалізувався в НС «ZITtrack». В цьому НС кіберфізичних систем навігаційні пристрої передавали свою інформацію на сервер. Причому кожен тип навігаційного пристрою мав свій протокол зв'язку. Також пристрої перебували в різних режимах роботи. Інформація з пристроїв накопичувалася в одній таблиці БД. Це означає, що всі дані повинні були приведені до однакового вигляду. В протоколах зв'язку навігаційних пристроїв виділялося універсальну частину, яка була однакова для всіх пристроїв, а саме встановлення та розірвання з'єднання, логування та інше. Тим часом в кожного пристрою використовувався свій формат повідомлення, а також були свої параметри з'єднання.

Спеціальна частина модульного мережного інтерфейсу, містила константи конфігурування універсальної частини та функції обробки даних протоколу зв'язку. При практичній реалізації модульного мережного інтерфейсу на основі принципу використання стикалася із тим, як передавати інформацію. Вся сукупність інформації є телеметричними даними. Як правило, вони передаються по мережі у вигляді потоків даних, які складаються із заголовку та кількох пакетів даних отриманих від підсистеми датчиків. В програмній реалізації це все виглядало, як простий набір різнотипних даних.

Мережний інтерфейс розроблений на основі принципу використання в НС «ZITtrack» використовувався для зв'язку з навігаційними пристроями. В практичній реалізації модульного мережного інтерфейсу на основі принципу використання існував зв'язок об'єктів у вигляді методів об'єкта універсальної частини мережного інтерфейсу, які в якості аргумента були об'єкти спеціальної частини мережного інтерфейсу. Універсальна частина мережного інтерфейсу для зв'язку з навігаційними пристроями складалася з об'єкта сервера зв'язку (типу GPSEvent) та об'єкта встановлення зв'язку. Спеціальна частина мережного інтерфейсу була реалізована у вигляді об'єкта констант і об'єкта протоколу зв'язку, типу TrackClientPacket Handler (Рисунок 4.10).

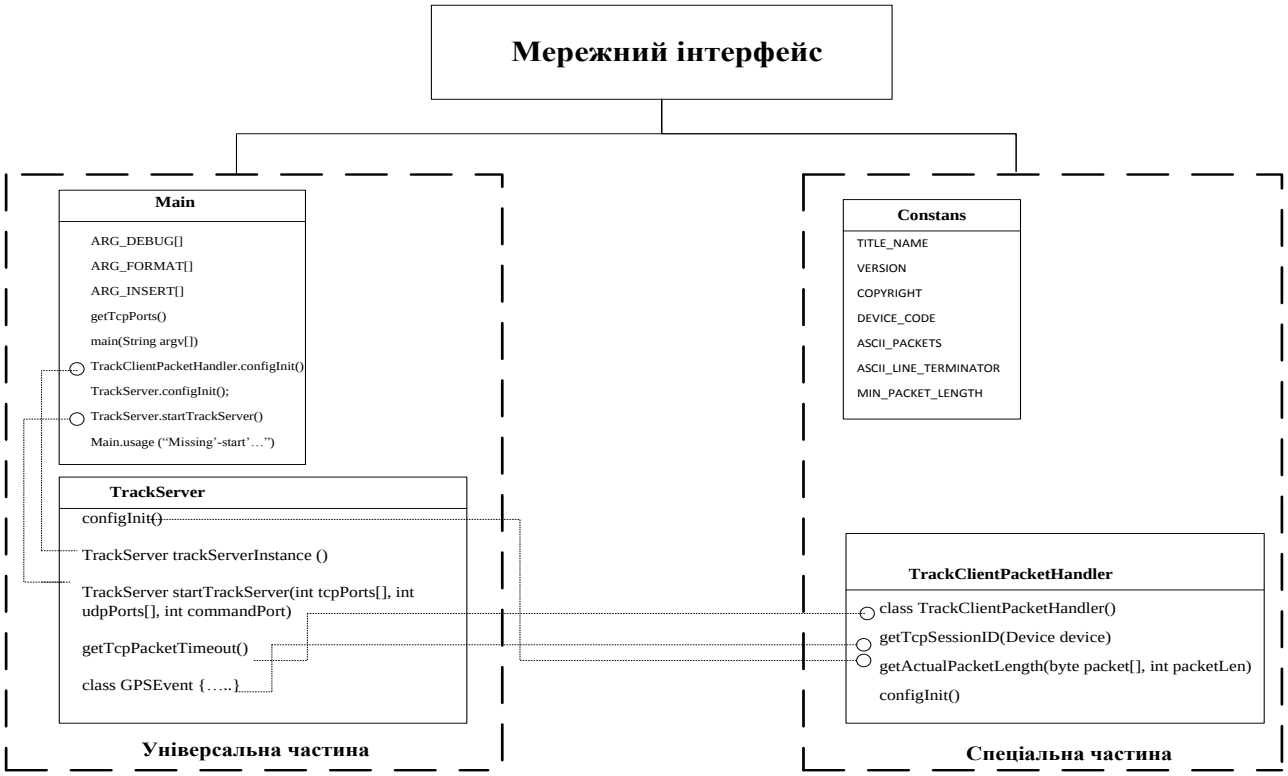


Рисунок 4.10 - Схема структури класів мережного інтерфейсу НС кіберфізичних систем на основі принципу використання

Універсальна частина мережного інтерфейсу на основі принципу використання містила клас GPSEvent (Рисунок 4.11).

```
class GPSEvent {
GPSEvent(DCServerConfig server, String ipAddress, int clientPort, String
modemID);
 }
```

Рисунок 4.11 - Приклад програмної реалізація класу GPSEvent

Універсальна частина модульного мережного інтерфейсу на основі принципу використання, яку використовували всі сервери НС кіберфізичних систем і вона для всіх них була однакова, скільки б тих серверів не було, містила файли GPSEvent, TrackClientPacketHandler. У першому файлі були

програмно оголошено порти через, які обмінювалися клієнт з сервером у безкабельній мережі. У другому файлі, описано конфігурацію сервера.

Спеціальна частина модульного мережного інтерфейсу на основі принципу використання міститься у класі TrackClientPacketHandler (Рисунок 4.12).

```
class TrackClientPacketHandler {
GPSEvent gpsEvent = null;
parseInsertRecord_Common(GPSEvent gpsEv);
gpsEvent = new GPSEvent(Main.getServerConfig(null),
ipAddress, this.clientPort, modemID);
Device device = this.gpsEvent.getDevice(); this.gpsEvent = new
GPSEvent(Main.getServerConfig(null),
ipAddress, this.clientPort, accountID, deviceID);
    }
```

Рисунок 4.12 - Приклад програмної реалізації класу TrackClientPacketHandler

На рис. 4.12, бачимо спрощений вигляд спеціальної частини модульного мережного інтерфейсу на основі принципу використання, яка містила файл TrackClientPacketHandler. Файл TrackClientPacketHandler містив інформацію про створені з'єднання та транзакції, які відбулися і відбуваються при клієнт-серверній взаємодії у безкабельній мережі. Чому саме модульний мережний інтерфейс на основі принципу використання, тому що термін використання передбачав функцію getHandlePacket, яка викликала дані. При модульному мережному інтерфейсі на основі принципу використання, клієнт мав доступ до універсальної частини мережного інтерфейсу з'язку з пристроями, інформація про яких, знаходилася на сервері.

Специфіка даного підходу до розробки мережного інтерфейсу полягала в принципі використання. Модульний мережний інтерфейс будувався на основі принципу використання, а саме спеціальна частина використовувала універсальну частину мережного інтерфейсу.

4.4 Спосіб порівняння мережних інтерфейсів навігаційного сервісу кіберфізичних систем на ефективність

Спосіб порівняння розроблених мережних інтерфейсів і традиційних підходів полягав в наступному. За допомогою архітектури Jini мережний інтерфейс може надавати клієнтам доступ до протоколів зв'язку у багатьох відношеннях. Мережний інтерфейс складався з декількох варіантів протоколів зв'язку, який завантажувався на клієнт під час пошуку сервера, а потім виконувався локально. Альтернативно, модульний мережний інтерфейс міг використовуватися лише, як проксі до віддаленого сервера. Коли клієнт викликав протокол зв'язку, він надсилав запити через мережу на сервер, який створював з'єднання.

Одним з важливих принципів архітектури Jini є те, що протокол зв'язку, який використовувався для зв'язку між об'єктом проксі - сервісу і віддаленим сервером не повинен бути відомий клієнту, адже протокол зв'язку є частиною реалізації мережного інтерфейсу. Мережний інтерфейс може використовувати RMI, CORBA, DCOM. Клієнту не потрібно було «думати» про протокол зв'язку, тому що він міг використовувати вже відомий мережний інтерфейс. Різні реалізації одного і того ж мережного інтерфейсу було використано у зовсім різних підходах щодо впровадження і абсолютно різні протоколи зв'язку. Мережний інтерфейс використовував спеціалізоване обладнання для виконання клієнтських запитів. Підхід до реалізація одного з модульних мережних інтерфейсів змінювався з плином часу. Порівняємо Jini з запропонованим модульним мережним інтерфейсом наступним чином.

Критерієм оцінки ефективності міг виступати час виконання сервером клієнтського запиту. На стороні клієнта виміряти час на обробку запиту було не коректно тому, що є затримки мережі, тобто в із сторони з'єднання це впливало на час проходження команди з'єднання. А на стороні сервера виміряти час на обробку одного запиту простіше, адже це час від отримання запиту до відправки відповіді. Постановка експерименту: клієнтом виступає ноутбук Asus K73E-TY272D з наступними характеристикам, таблиця 4.1.

Таблиця 4.1 Характеристики ноутбука Asus K73E-TY272D

Тип процесора	Intel Core i3-2330M
Частота, GHz	2,2
Вінчестер, GB	750
Об’єм оперативної пам’яті, GB	4
Графічний адаптер, об’єм пам’яті	Intel HD3000
ОС	Windows XP
Чіпсет	Intel HM65 Express
Оптичний привід	DVD-RW Super Multi
Зовнішні порти	4xUSB 2.0, VGA, HDMI, mic-In, line-out
Формати карт пам’яті	SD, MMC, MS
Веб камера, Мп	0,3
Діагональ екрану, дюймів	17,3
Розширення екрану	1600x900
Безпровідні комунікації:	
Wi-Fi	+
Bluetooth	+
Ємкість батареї, мА	5200
Провідні комунікації:	
Мережний адаптер	10Base-T/100Base-TX/1000Base-T

Із використанням методу розробки мережного інтерфейсу було встановлено, що на стороні сервера при модульному мережному інтерфейсі на основі принципу використання час роботи процесора при отриманні запиту від клієнта становитиме ~ 0,2 мс, також маємо змогу побачити при 1,12,100 запитах (див. додаток А 2). І ця обробка тривала поки не вичерпався прямий час процесора то затримки на обробку команди. Модульний мережний інтерфейс на основі принципу наслідування час роботи процесора при отриманні запиту від клієнта становитиме ~ 0,3 мс, також маємо змогу побачити при 1,12,100 запитах. Модульний мережний інтерфейс на основі принципу інстанціювання, тобто кожного разу новий під специфічний запит від клієнта, становитиме ~ 0,04 мс часу процесора на обробку одного запиту, а також маємо змогу побачити при 1,12,100 запитах. Тобто із зростанням запитів від клієнта час роботи процесора збільшувався, адже його час ще додатково йде на обробку

запитів до бази даних на якій і будувався модульний мережний інтерфейс на основі принципу інстанціювання.

Отже, на основі вище сказаного можна сказати що модульні мережні інтерфейси на основі цього принципу наслідування та інстанціювання надійніші за модульний мережний інтерфейс на основі цього принципу використання, але у свою чергу модульний мережний інтерфейс на основі цього принципу використання ефективніший за вище згадані два модульних мережних інтерфейси.

Було використано спосіб порівняння запропонованих мережних інтерфейсів з традиційними, який базувався на критеріях ефективності. Отримавши дані для порівнянні модульних мережних інтерфейси за допомогою цього способу, на основі різних принципів, було проведено порівняти модульний мережний інтерфейс з стандартними засобами розробки, таблиця 4.2.

Таблиця 4.2 Порівняння мережних інтерфейсів НС кіберфізичних систем

Тип інтерфейсу **Прикладний/Ethernet**	Час реакції мереж-ного інтерфейсу, мс	Термін розробки мережного інтерфейсу, люд./год.
Спеціалізований мережний інтерфейс розроблений стандартним засобами	~ 1,6	~ 8,7
Модульний мережний інтерфейс розроблений на основі принципу:		
Наслідування	~ 0,3	~ 2,7
Використання	~ 0,2	~ 0,7
Інстанціювання	~ 0,04	~ 1,8
Модульний мережний інтерфейс розроблений стандартним засобами	~ 0,5	~ 3,64
Мережний інтерфейс розроблений на основі Jini	1,2-2,0	~ 7,4

Примітка. Дані за 2015 рік.

Отримані дані показували, що розроблений модульний мережний інтерфейс дозволив зекономити час на розробку, що зменшує трудомісткість. А реалізація не завжди, але в більшості випадків ефективніша ніж в універсальному мережному інтерфейсі. Як видно із таблиці 4.1, модульний мережний інтерфейс має кращі показники за інші види мережних інтерфейсів.

Також було проведено порівняння мережних інтерфейсів НС кіберфізичних систем по критеріям ефективності, таблиця 4.2.

Таблиця 4.3 Порівняння мережних інтерфейсів НС кіберфізичних систем по критеріях ефективності

Мережні інтерфейси	Критерії ї ефективності		Ефективність
Тип інтерфейсу **Прикладний/Ethernet**	Час реакції мереж-ного інтерфейсу	Термін розробки мережного інтерфейсу	Сума по критеріях
Спеціалізований мережний інтерфейс розроблений стандартним засобами	10%	20%	30%
Модульний мережний інтерфейс розроблений на основі принципу:			
Наслідування	30%	25%	55%
Використання	20%	30%	50%
Інстанціювання	50%	45%	95%
Модульний мережний інтерфейс розроблений стандартним засобами	25%	22%	47%
Мережний інтерфейс розроблений на основі Jini	21%	25%	46%

Примітка. Дані за 2015 рік.

Модульний мережний інтерфейс на основі принципу наслідування (55%) менш ефективним, але надійний. Модульний мережний інтерфейс на основі принципу використання (50%) не ефективний, а модульний мережний інтерфейс на основі принципу інстанціонування (95%) ефективний, але у свою

чергу не достатньо надійний, таблиця 4.3. Тобто модульні мережні інтерфейси на основі принципів наслідування і використання не є пристосовані до виконання експлуатаційного завдання, це обумовлено їх технічними характеристиками. А модульний мережний інтерфейс на основі принципу інстанціювання достатньо надійний, адже в значній мірі він показував кращі загальні економічні показники роботи. Отже, запропоновані мережні інтерфейси на основі принципу наслідування, використання, інстанціювання на 19% ефективніше працюють у порівнянні з спеціалізованими мережними інтерфейсами розробленим стандартним засобами та аналогом мережним інтерфейсом розробленим на основі Jini, адже навіть у сумі вони не дають ефективність більшу, ніж 95%. Проведене порівняння мережних інтерфейсів побудованих з застосуванням вдосконалених методів та традиційних підходів до побудови мережних інтерфейсів, яке дало можливість скоротити у 2 рази час розробки мережних інтерфейсів навігаційного сервісу «ZitTrack».

За допомогою методу розробки мережного інтерфейсу для НС кіберфізичних систем шляхом параметризації системи даних і команд, за якою для кожного класу задач були визначені підходи для розробки модульних мережних інтерфейсів, він дав можливість зменшити час реакції мережного інтерфейсу на запит від клієнта.

4.5 Навігаційний сервіс «ZITtrack»

Навігаційний сервіс «ZITtrack», далі НС «ZITtrack», використовувався для моніторингу рухомих об’єктів . Цей навігаційний сервіс працював не залежно від якого небудь конкретного пристрою. НС «ZITtrack» спеціально розроблений для використання в малих і середніх комерційних підприємствах, які мали намір скористатися GPS-моніторингом транспортних засобів. Тим не менше, НС «ZITtrack» міг конфігуруватися і масштабуватися для великих підприємств, як це видно на рисунку 4.13.

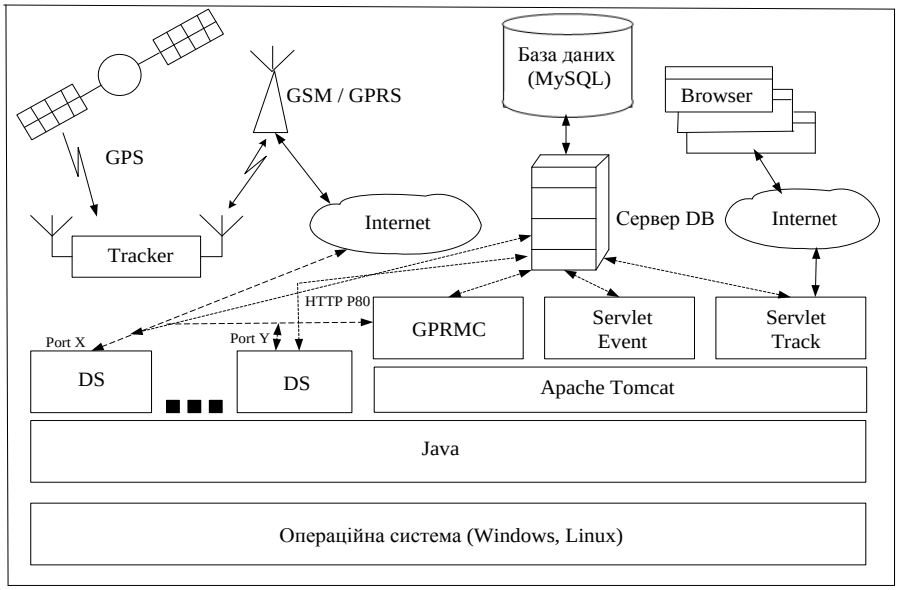


Рисунок 4.13 - Схема архітектури НС «ZITtrack»

На стороні сервера, НС «ZITtrack» був призначений для пристрою і злежить від використовуваних протоколів (GPS, GSM / GPRS). Для того, щоб використовувати повинен був визначений сервер DB, пристрій зв'язку, протоколи зв'язку, які взаємодіяли з віддаленими пристроями і зберігати дані в базі даних MySQL. Servlet Event, Servlet Track працювали з використанням протоколу HTTP. В Servlet-контейнері працював Apache Tomcat.

На рисунку 4.13 бачимо а рхітектуру НС «ZITtrack» різні протоколи зв'язку, сервер отримували і аналізували дані від віддалених пристроїв і працювали, як окремі процеси на платформі Java.

Інтерфейс користувача, мав вигляд веб-сторінки, реалізований на стороні клієнта, який налаштовувався відповідно до потреб користувача, як наприклад налаштування опцій меню веб-сторінки. НС «ZITtrack» був протестований на таких операційних системах, як: Windows XP/ Vista / 7 / 8, Linux. Можливості НС «ZITtrack» можна розширити і вони завжди доступні для користування.

З використанням модульних мережних інтерфейсів, а саме на основі принципів використання, інстанціювання, наслідування було розроблено навігаційний сервіс «ZITtrack». Цей сервіс (інформаційний, навігаційний та диспетчерський) був призначений для відображення в реальному масштабі часу на карті місцевості розташування підконтрольних рухомих об'єктів, відображення їх поточного стану, а також перегляду історії їх переміщень і

зміни їх стану. Основні області застосування: пасажирський транспорт і маршрутні таксі, міжміські та міжнародні перевезення, будівельна та комунальна техніка, вантажний транспорт, екстрені служби, особистий транспорт.

Наприклад, керівник будь-якої компанії бажає, щоб його компанія отримувала максимальний прибуток. Реалізувати це можна різними способами. Вдалим способом збільшити прибутковість є зниження власних витрат. Всі підприємства, які користуються транспортом, несуть значні витрати на його експлуатацію. Частка транспортних витрат у структурі собівартості продукту становить для більшості підприємств від 20 до 80%. І ці витрати постійно зростають. Вартість палива (бензину та дизельного) за останні три роки значно виросла. Збільшувалася собівартість продукції, зростає ціна, зменшувався прибуток, знижувалася конкурентоспроможність, сповільнювався ріст і розвиток підприємства, і так далі.

Відхилення від маршруту, зливання палива стали повсякденною частиною виробничого життя. Цей список можна продовжувати, але все це не мало великого значення, якщо цим процесом керувати. Висока вартість нової техніки повинна компенсуватися збільшенням терміну її служби, зниженням експлуатаційних витрат, підвищенням надійності.

В якості рухомих об’єктів можуть виступати транспортні засоби (автомобілі, мотоцикли, мотороллери ...), маломірні судна, катери, човни, мала авіаційна техніка і т.п. Всі підконтрольні транспортні засоби повинні бути обладнані спеціальною апаратурою, яка збирає інформацію про переміщення машин, їх стан та стан їх вантажів.

Навігаційний сервіс «ZITtrack» дає повний контроль за транспортом (облік пробігу, витрати палива та інших параметрів) і завдяки цьому збільшувалася економія палива і дисципліна водіїв. Цей НС «ZITtrack» - це недорогий і ефективний інструмент для обліку використання автотранспорту.

Таким чином, програмна реалізація мережних інтерфейсів на основі принципів наслідування, використання та інстанціювання, міститиме в собі

набір об’єктів, що мають стан та поведінку. Об’єкти взаємодіють використовуючи повідомлення. Будувалася ієрархія об’єктів: програма в цілому — це об’єкт, для виконання своїх функцій вона зверталася до об’єктів, що містяться у ньому, які у свою чергу виконують запит шляхом звернення до інших об’єктів програми. Звісно, щоб уникнути безкінечної рекурсії у зверненнях, на якомусь етапі об’єкт трансформує запит у повідомлення до стандартних системних об'єктів, що даються мовою та середовищем програмування. Стійкість та керованість навігаційного сервісу кіберфізичних систем забезпечуються за рахунок чіткого розподілення відповідальності об’єктів (за кожну дію відповідає певний об’єкт), однозначного означення інтерфейсів між об’єктної взаємодії та повної ізольованості внутрішньої структури об’єкта від зовнішнього середовища, що забезпечує надійність НС кіберфізичних систем.

Навігаційний сервіс «ZITtrack» надає можливість повного контролю над транспортом (облік пробігу, витрат пального та інших параметрів) і таким чином збільшує економію палива і дисципліна водіїв. Цей НС «ZITtrack» - недорогий і ефективний інструмент для обліку використання транспортних засобів.

Для того, щоб користувач мав можливість використовувати цей НС кіберфізичних систем необхідно зареєструватися (Рисунок 4.14). Ця реєстрація реалізована у вигляді веб-сторінки в браузері.

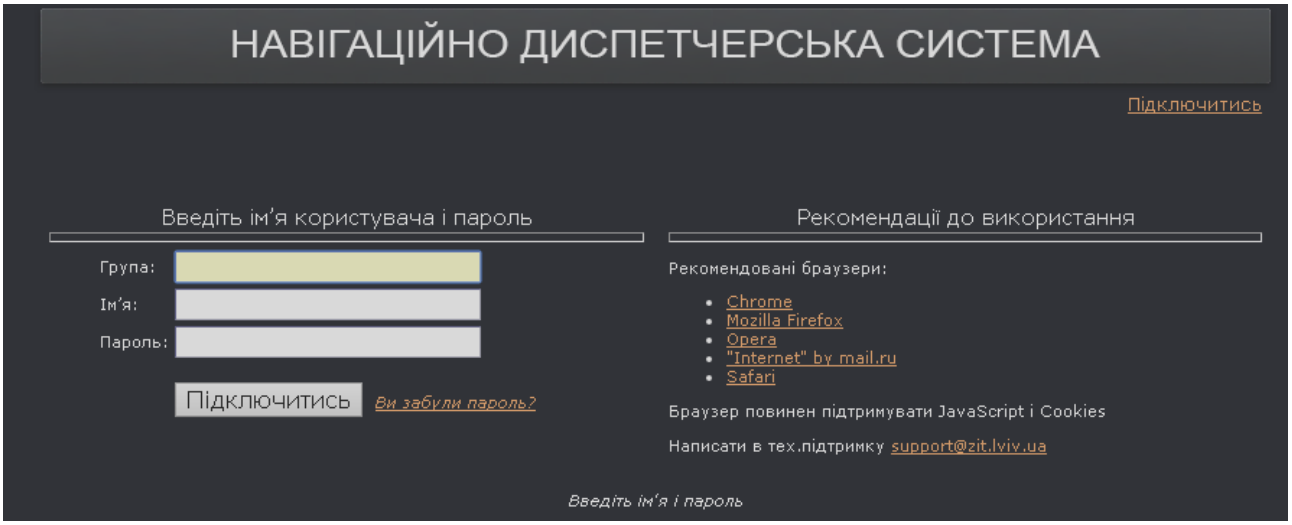


Рисунок 4.14 - Спрощений вигляд вікна реєстрації нового користувача

Для правильної роботи і правильного використання навігаційного сервісу користувачі повинні використовувати наступні браузери: Chrome, Mozilla Firefox, Opera. Крім того, ваш браузер має підтримувати мову програмування JavaScript і «Cookies».

Познайомимося з НС кіберфізичних систем в демонстраційному режимі. А саме, коли користувач входить в НС кіберфізичних систем, це реалізовано у вигляді веб-сторінки у браузері (Рисунок 4.15).

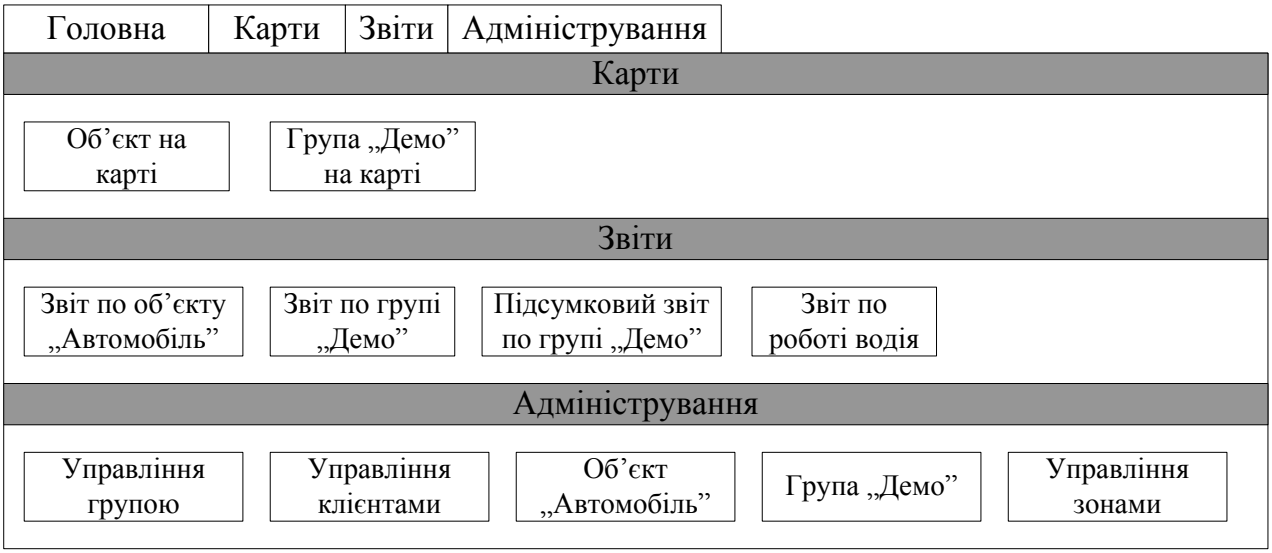


Рисунок 4.15 - Спрощений вигляд навігаційного сервісу «ZITtrack»

Вкладка об'єкта на карті дозволяє бачити весь шлях руху і місця зупинки даного об'єкта на карті області.

У вкладках групи "Демо", на карті, бачимо, всі об'єкти, які мають вбудований трекер.

Зведені звіти: об'єктів, груп об'єктів, водій і підсумковий звіт, можна переглянути інформацію: рух об'єкта за певний період часу, обраний, місце його знаходження, швидкості, зупинки, парковки.

На сьогоднішній день навігаційний сервіс «ZITtrack» є функціональним і швидко розвивається, як сервіс GPS моніторингу автотранспорту для приватних і комерційних цілей. Основною відмінністю цього НС «ZITtrack» кіберфізичних систем від аналогічних, представлених на ринку, є те, що він не вимагає спеціалізованого робочого місця та спеціалізованого ПЗ. Доступ до НС кіберфізичних систем отримується з будь-якого комп'ютера або мобільного пристрою, за допомогою веб-браузера. Цей НС кіберфізичних систем підтримуємо роботу з усіма відомими веб-браузерами [2]. Також НС кіберфізичних систем підтримує роботу з датчиками рівня палива, сигналами від датчиків витрат палива, дискретними датчиками, датчиками наявності об'єкта, пасажира і т.д.

Навігаційний сервіс «ZITtrack» не тільки підтримує збір даних і зберігання GPS стеження і телеметрії даних з віддалених пристроїв, але і включає в себе наступні функцій:

- Web-аутентифікація. Кожен акаунт може підтримувати кілька користувачів, і кожен користувач має свій власний пароль, логін і здатен самостійно контрольований доступ до розділів в межах дозволених адміністратором сайту.
- GPS пристрої стеження. Пристрої від різних виробників можуть бути відстежені одночасно. Підтримка наступних GPS пристроїв стеження, які входить в НС «ZITtrack» кіберфізичних систем:
    - Aspicore GSM Tracker (Nokia, Samsung, Sony Ericsson телефони);
    - Sanav GC-101, MT-101 і CT-24- персональний трекер (HTTP-протоколу), Sanav GX-101 Tracker автомобіля (HTTP-протоколу);
    - V-Нд 3338- -персональний трекер;
    - GPSReader - GPS реєстратор даних з автоматичним Wi-Fi для завантаження даних;
    - "GPSMapper", для телефонів, які підтримують цей сервіс;
    - "NetGPS", для пристроїв, які мають відповідне апаратне забезпечення [4,5].

На кожному рухомому об’єкті встановлювався навігаційний прилад "трекер", який здійснює визначення географічних координат, напрямок і швидкість його руху за допомогою супутникової системи GPS і передачі її по каналу GSM (GPRS) диспетчеру НС кіберфізичних систем. Можливості навігаційного приладу "Tracker":

- контроль переміщення транспортного засобу та його місцезнаходження;
- контроль зовнішніх подій за допомогою вхідних ліній;
- спільна робота зі штатною системою сигналізації автомобіля (використовувалася лінія пейджер);
- можливість роботи з датчиками рівня палива;
- тривожна кнопка з можливістю оповіщення за вказаним номером;
- можливість управління зовнішнім виконавчим пристроєм;
- дистанційне зміна налаштувань приладу, використовуючи канал передачі даних в мережах GSM (CSD);
- накопичення даних в "чорному ящику" при знаходженні транспортного засобу поза зоною дії GSM оператора;
- автоматична вивантаження вмісту в "чорному ящику" при входженні транспортного засобу в зону дії GSM оператора;
- передача даних через Інтернет, використовуючи технологію пакетної передачі даних GPRS;
- використання зовнішніх роздільних або зовнішніх суміщених антен;
- підтримка різних протоколів, як цифрового, так і текст-орієнтованого.

Бортова апаратура також виконує контроль стану різних датчиків, встановлених на об’єкті. Отриману інформацію прилад передає на сервер моніторингу. Після того, як інформація розміщена на сервері, вона стає доступна клієнтам (диспетчерам) НС кіберфізичних систем. Клієнт (диспетчер) через мережу Internet отримує оброблену інформацію про місцезнаходження і стан транспортних засобів (дані про перевищення швидкості, дотримання маршруту, тривалість і місце зупинок, витрати палива і т.д.). Бачимо у вигляді НС «ZITtrack» для доріг Львівської області, використаний комунальним

підприємством «Львівавтодор» (Рисунок 4.16). Усі зміни початково розробленого навігаційного сервісу «ZITtrack» зроблено у комерційних цілях цим комунальним підприємством.

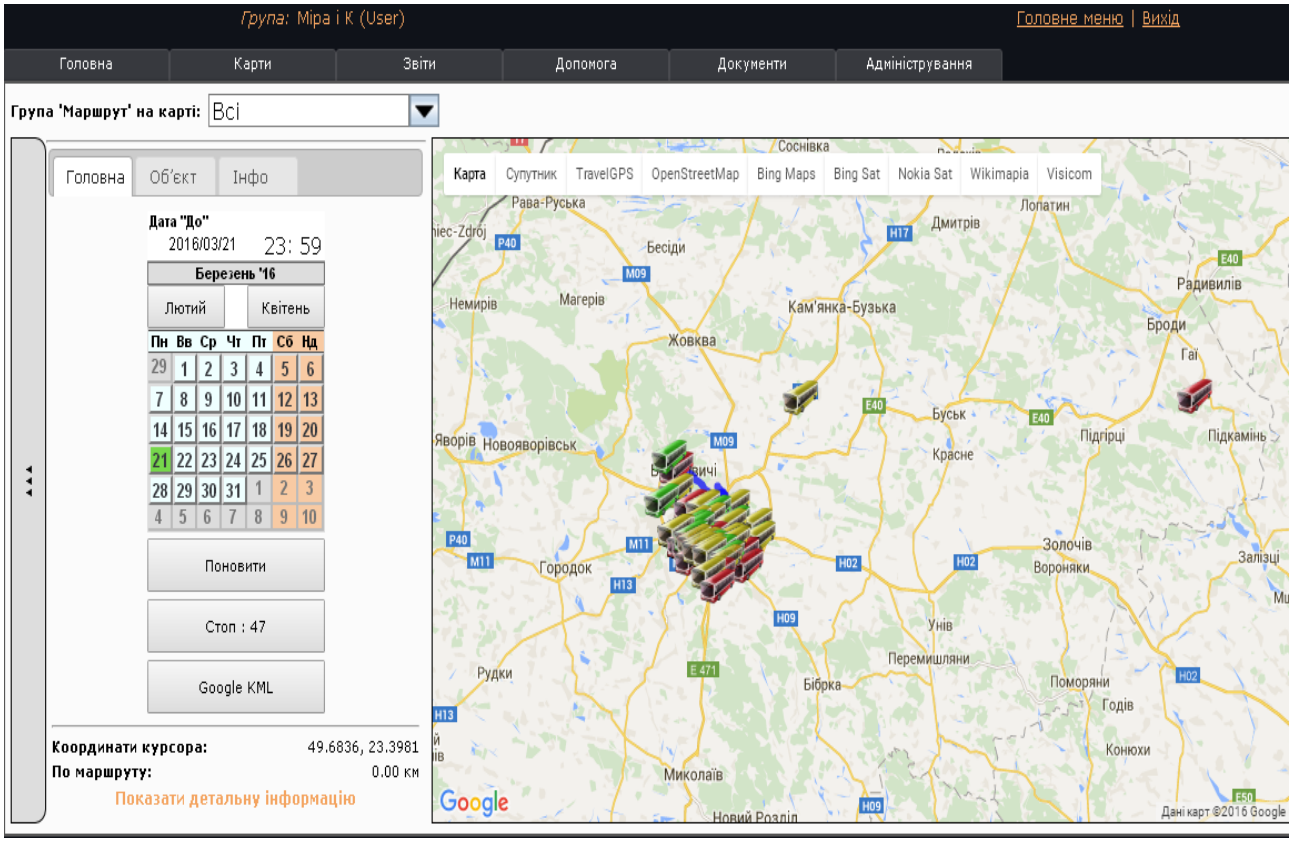


Рисунок 4.16 - Спрощений вигляд НС «ZITtrack» при роботі з об’єктами.

НС кіберфізичних систем підтримує GPS трекери різних виробників, серед яких як переносні GPS трекери, так і професійні GPS рішення з усього світу.

Для того, щоб користувач цього НС кіберфізичних систем отримав необхідні дані, що його цікавлять по відношенню до його автомобіля, цей НС кіберфізичних систем формує звіт (Рисунок 4.17).

НАВІГАЦІЙНО ДИСПЕТЧЕРСЬКА СИСТЕМА

Група: Mipa i K (User) — Підсумкові звіти по групі 'Маршрут' | Головне меню | Вихід

Останні зареєстровані координати групи [Всі]

Оновити — з 2016/03/21 по 2016/03/21 23:59:00 [Europe/Kiev]

#	Опис об'єкта	Ід.об'єкта	Дата	Час	Широта/Довгота	Адреса	Latest Batt %	З моменту перевірки
1	AO 8084 AA	620	2015/05/20	20:36:36	49.8120/24.0746			305d 14h 52m
2	BC 7080 AA	624	2015/05/20	22:36:48	49.8169/24.1347		93%	305d 12h 52m
3	AC 5858 BC	501	2015/05/20	22:06:48	49.8603/24.0415		93%	305d 13h 22m
4	AT 3646 BM	502	2015/05/20	22:34:51	49.8261/23.9316		93%	305d 12h 54m
5	BK 0873 AA	503	2015/05/20	21:40:29	49.8260/23.9316			305d 13h 48m
6	BC 0339 AA	504	2015/05/20	22:35:46	49.7930/24.0588			305d 12h 53m
7	BC 0343 AA	505	2015/05/20	21:42:41	49.8122/24.0743		93%	305d 13h 46m
8	BC 0345 AA	506	2015/05/20	22:25:44	49.8749/23.9345			305d 13h 03m
9	BC 0348 EH	507	2015/05/02	19:57:06	49.8069/23.9513			323d 15h 32m
10	BC 0425 AA	508	2015/05/20	22:08:01	49.8122/24.0743		93%	305d 13h 21m
11	BC 0431 AA	509	2015/05/20	22:35:48	49.8019/24.0175			305d 12h 53m
12	BC 0434 AA	510	2015/05/20	22:12:16	49.8261/23.9315			305d 13h 16m
13	BC 0571 AA	511	2015/05/02	08:32:49	49.8174/23.9716			324d 02h 56m
14	BC 0618 BT	512	2015/02/07	15:56:22	49.8552/24.0735			407d 18h 32m
15	BC 0698 AA	513	2015/05/19	20:45:21	49.9875/24.2402		93%	306d 14h 43m
16	BC 0815 AA	514	2015/05/20	21:14:57	49.8263/23.9305			305d 14h 14m
17	BC 0816 BP	515	2015/05/20	21:45:38	49.8121/24.0741		93%	305d 13h 43m

Рисунок 4.17 - Спрощений вигляд звіту в НС «ZITtrack»

Користувач у свою чергу може отримати наступні звіти від цього НС кіберфізичних систем:

- звіт по витраті (заправка, слив) палива;
- перевищення швидкості за довільний період;
- перевищення швидкості місячний (дні порушень і на скільки);
- завантаження транспортного засобу (за період (скільки і коли працював);
- поїздки та стоянки;
- заправки;
- звіт по використанню палива (витратомір);
- загальний звіт по парку за період (паливо, пробіг);
- добовий звіт про рух автотранспорту;
- відвідування контрольних точок (про рух по маршруту);
- можлива доробка різних звітів для вирішення поставлених завдань на підприємстві.
- інтеграція з програмою «Google Earth» (відображення розташування рухомих об’єктів у вікні Google Earth, відображення записаних трас руху);

- поряд з приладами приватної фірми можна використовувати персональні трекери різних виробників, для моніторингу людей, тварин, техніки;
- можливість відображення місця розташування рухомих об’єктів на веб-сторінці;
- великий вибір картографічних платформ.

Саме завдяки звітам, які формує цей НС кіберфізичних систем користувач може від слідкувати рух його автомобіля у зазначений ним період часу.

4.6 Можливості навігаційного сервісу «ZITtrack»

До можливостей програмного забезпечення НС «ZITtrack» можна віднести:

- відображення поточного статусу рухомих об’єктів (параметри руху, достовірність навігаційних даних, події тощо);
- перегляд записаної історії переміщень, виникнення подій за вибраний проміжок часу, з відображенням на карті маршруту переміщення, точок виникнення подій;
- проглядання записаної історії в прискореному масштабі часу;
- формування швидкісної діаграми, графіків зміни стану аналогових і цифрових датчиків, діаграми контролю якості каналу зв’язку;
- дистанційне керування різними виконавчими пристроями, підключеними до об’єктового обладнання, передача команд на об’єкт;
- відображення в реальному часі поточного місцезнаходження рухомого об’єкту(автомобіля) на карті місцевості.
- оповіщення користувача про виникаючі на об’єкті подіях текстовими повідомленнями, супроводжуваними звуковою сигналізацією і колірної індикацією;
- визначення контрольних точок (зон) на карті місцевості для рухомого об’єкта, індикація подій на вхід, вихід об’єкта за межі зони;
- формування великого набору друкованих звітів;

- інтеграція з програмою «Google Earth» (відображення розташування рухомих об’єктів у вікні Google Earth, відображення записаних трас руху);
- поряд з приладами приватної фірми можна використовувати персональні трекери різних виробників, для моніторингу людей, тварин, техніки;
- можливість відображення місця розташування рухомих об’єктів на веб-сторінці;
- великий вибір картографічних платформ.

Отже, НС «ZITtrack» кіберфізичних систем був реалізовий для моніторингу рухомих об’єктів.

4.7 Висновки до розділу

1. Запропоновано покращені мережні інтерфейси у навігаційних сервісах кіберфізичних систем для застосування в одному із класів задач: взаємодія навігаційного сервісу кіберфізичних систем з пристроями, взаємодія між інтерфейсом користувача і навігаційним сервісом кіберфізичних систем, взаємодія навігаційного сервісу кіберфізичних систем з базою даних.
2. Запропоновані вдосконалені методи для розробки мережного інтерфейсу, які визначають клас задач у навігаційних сервісах кіберфізичних систем.
3. В результаті використання вдосконалених методів, можна на 19% підвищити ефективність мережних інтерфейсів у навігаційних сервісах кіберфізичних систем.
4. Розроблений спосіб та проведене порівняння мережних інтерфейсів побудованих з застосуванням вдосконалених методів та традиційних підходів до побудови мережних інтерфейсів, які дали можливість скоротити у 2 рази час розробки мережних інтерфейсів навігаційного сервісу ZiTtrack.
5. Факт збіжності результатів досліджень мережних інтерфейсів з прогнозованими підтверджує ефективність запропонованого підходу.

6. Запропоновані засоби мережної взаємодії були використані при розробці навігаційного сервісу «ZITtrack» і протестовані на час реакції, а також отриманий термін розробки.

В И С Н О В К И

У дисертаційній роботі проведено теоретичне обґрунтування й нове вирішення актуальної наукової задачі підвищення ефективності мережних інтерфейсів навігаційних сервісів кіберфізичних систем. Ці сервіси передбачають велику кількість мережних інтерфейсів в порівнянні з іншими галузями застосування. При цьому було отримано такі наукові та практичні результати:

1. Проведено аналіз відомих методів та засобів забезпечення мережної взаємодії між навігаційними сервісами кіберфізичних систем та користувачами і навігаційними пристроями у безкабельній мережі.

2. Запропоновано вдосконалення методу клієнт-серверної взаємодії для навігаційних сервісів кіберфізичних систем шляхом параметризації системи команд та даних, які для кожного класу задач визначають принципи розробки мережного інтерфейсу та його специфікації, що дає можливість зменшити час реакції мережного інтерфейсу на запити від клієнтів та терміни розробки для його програмної реалізації.

3. Запропоновано метод розробки мережного інтерфейсу на базі моделі параметризованого мережного інтерфейсу у навігаційних сервісах кіберфізичних систем, який дав змогу спростити розробку мережних інтерфейсів та підвищити ефективність цих інтерфейсів.

4. Вдосконалено існуючі методи динамічної, статичної та статично-динамічної мережної взаємодії в навігаційних сервісах кіберфізичних систем шляхом оптимізації функцій мережного інтерфейсу, що дає можливість ефективно реалізовувати мережний інтерфейс навігаційних сервісів кіберфізичних систем.

5. Запропоновано вдосконалені методи розробки мережних інтерфейсів у навігаційних сервісах кіберфізичних систем на основі принципів - наслідування, використання та інстанціювання у відповідності до класу задач:

взаємодії з пристроями, взаємодії між інтерфейсом користувача і навігаційними сервісами кіберфізичних систем, взаємодії навігаційних сервісів кіберфізичних систем з базою даних, які дозволяють зменшити час реакції мережних інтерфейсів у навігаційних сервісах кіберфізичних систем.

6. Розроблений спосіб та проведене порівняння мережних інтерфейсів побудованих з застосуванням вдосконалених методів та традиційних підходів до побудови мережних інтерфейсів, які дали можливість скоротити у 2 рази час розробки мережних інтерфейсів навігаційного сервісу ZiTtrack.

7. Реалізовані методи розробки мережних інтерфейсів навігаційного сервісу кіберфізичних систем дали можливість на 19% підвищити ефективність мережної взаємодії кіберфізичних систем для моніторингу рухомих об’єктів.

8. Запропоновані засоби мережної взаємодії були використані при розробці навігаційного сервісу «ZITtrack» і протестовані на час реакції та отриманий термін розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chris Giametta «Pro Flex on Spring», 2009.- p.445.
2. Роберт Дж. Оберг «Технология COM + Основы и программирование = Understanding and Programming COM+: A Practical Guide to Windows 2000 First Edition». — М.: «Вильямс», 2000.- С. 480.
3. Тамре Л. «Введение в тестирование программного обеспечения ».- М.: Издательский дом «Вильямс», 2003.- С. 368.
4. Липаев В.В. «Обеспечение качества программных средств. Методы и стандарты». - М.: Синтег, 2001.- С. 246.
5. Макгрегор Дж., Сайкс Д. «Тестирование объектно-ориентированного программного обеспечения».- К: Диасофт, 2002.- С. 432.
6. Татарчук М. І. «Корпоративні інформаційні системи: Навч. Посібник», 2005.- С. 245.
7. Мухамедзянов Н. «Java. Server applications» - Издательство: СОЛОН - Р, 2003.- С. 267.
8. Орфалі Роберт, Ден Харкі «JAVA and CORBA in client server applications».
9. Дуглас Камер, Девід Л. Стівенс «Сети TCP/IP, том 3.Разработка приложений типа клиент/сервер», издательство «Вильямс», 2002.- С. 592.
10. Фленов М. Є. «Web-сервер глазами хакера: Проблемы безопасности Web-серверов; Ошибки в сценариях на PHP, Perl, ASP; SQL-инъекции», 2005.- С. 365.
11. Федоров А. Платформы и средства создания Web-сервисов // Комп'ютерПресс. — 2002.-С. 38-53.
12. Эммерих В. Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft/COM и Java/RMI. Пер. с англ. — М.: Мир, 2002. — С. 510.

13. Макгрегор Дж., Сайкс Д. Тестирование объектно-ориентированного программного обеспечения.- К: Диасофт, 2002.- С. 432.
14. Тамре Л. Введение в тестирование программного обеспечения.- М.: Издательский дом “Вильямс”, 2003.- С. 368.
15. Бормотов С.В. Системное администрирование на 100%. – СПб.; Питер, 2006.– С. 256.
16. Alex WebKnacKer Быстро и легко. Хакинг и антихакинг: защита и нападение. Учебное пособие.— М.: Лучшие книги, 2004.- С. 400.
17. Э. Гамм, Р. Хелм, Р. Джонсон, Д. Влиссидес «Приёмы объектно-ориентированного проектирования. Паттерны проектирования», Питер, Москва, 2007.- С. 366.
18. Джон Влиссидес «Применение шаблонов проектирования», Москва, 2003.- С. 130.
19. Пастернак І. Класифікація засобів модульної взаємодії між клієнтом і сервером / І.І. Пастернак, Ю.В. Морозов // Вісник «Комп’ютерні системи та мережі».- Львів: НУ «Львівська політехніка», 2011.- №717.- С. 108-113.
20. Пастернак І. Модель об’єктної клієнт-серверної взаємодії / І.І. Пастернак , Ю.В. Морозов // Вісник «Комп’ютерні науки та інформаційні технології».- Львів: НУ «Львівська політехніка», 2011.- №719.- С. 164-167.
21. Пастернак І. Мережні інтерфейси рівня клієнт-сервер / І.І. Пастернак, Ю.В. Морозов // Вісник «Інформаційні системи та мережі».- Львів: НУ «Львівська політехніка», 2012.- №743.- С. 121-131.
22. Пастернак І. Параметризовані мережні інтерфейси / І.І. Пастернак // Вісник «Комп’ютерні науки та інформаційні технології».- Львів: НУ «Львівська політехніка», 2012.- №744.- С. 3-9.
23. Пастернак І. Модульний інтерфейс клієнт-серверної взаємодії / І.І. Пастернак // Вісник «Комп’ютерні системи та мережі».- Львів: НУ «Львівська політехніка», 2012.- №745.- С. 160-163.
24. Пастернак І. Огляд та аналіз навігаційної системи OpenGTS / І.І. Пастернак // Вісник «Нові технології».- Кременчук: Кременчуцький університет

«Економіки, інформаційних технологій і управління», 2012.- № 2-3 (36-37).- С. 88-91.

25. Pasternak I. Modular network interface for distributed information navigation systems / I.I. Пастернак, Ю.В. Морозов// Computational problems of electrical engineering. – V.3, № 2. – Lviv, 2014. – p. 47-56.
26. Пастернак І. Оцінка ефективності виконання паралельних обчислень / І.І. Пастернак, Є.Я. Ваврук // Матеріали 5-тої міжвузівської науково-технічної конференції ІППТ-2010.- Львів, 2010.- С. 280-281.
27. Пастернак І. Навантажувальне тестування мережного інтерфейсу клієнтськими запитами / І.І. Пастернак, Ю.В. Морозов // Матеріали 10-тої відкритої наукової конференції ІМФН .- Львів, 2012.- С. І.7-І.10.
28. Пастернак И. Реализация сетевого интерфейса на основе формул / І.І. Пастернак, Ю.В. Морозов // Матеріали міжнародної конференції ITS-2012.- Мінськ, Білорусь, 2012.- С. 252-253.
29. Пастернак І. Об'єктна взаємодія клієнта з сервером через мережний інтерфейс / І.І. Пастернак // Матеріали 4-тої науково-практичної конференції ЕлІт-2012.- Чинадієво, 2012.- С. 102-106.
30. Iryna Pasternak Modular Interface / І.І. Пастернак // Матеріали 6-тої міжнародної конференції CSIT-2012.- Львів, 2012.- С. 117-118.
31. Пастернак І. Опис та аналіз мережного інтерфейсу на основі формул / І.І. Пастернак // Матеріали заочної науково-практичної конференції «Наукові підсумки 2012 р.» .- Харків, 2012.- С. 41-42.
32. Iryna Pasternak Implementation of modular interface with client-server interaction / І.І. Пастернак // Матеріали 9-тої міжнародної конференції MEMSTECH - 2013.- Поляна, 2013.- С. 91-92.
33. Iryna Pasternak Using the principle of modularity of client-server interaction in the global network / І.І. Пастернак, Ю.В. Морозов // Матеріали 6-тої міжнародної конференції ACSN - 2013.- Львів, 2013.- С. 61-64.

34. Pasternak I. Client-Server Interaction on the WWW / Iryna Pasternak, Yuriy Morozov // Матеріали VI міжнародної конференції молодих вчених CSE-2013.- Львів 2013. – С. 10-15.
35. Артамонов Г.Т. Топология вычислительных сетей и сред. – М.: Радио и связь, 1985. – 192 с.
36. Барлоу Р., Прошан Ф. Статистическая теория надежности и испытания на безопасность. – М.: Наука, 1984. – 328 с.
37. Басакер Р., Саати Т. Конечные графы и сети. – М.: Наука, 1974. – 366 с.
38. Бартсекас Д., Галлагер Р. Сети передачи данных. – М.: Мир, 1989. – 544 с.
39. Вычислительные сети (адаптивность, помехоустойчивость, надежность) // С.Н. Самойленко и др. – М.: Наука, 1981. – 280 с.
40. Гадасин В.А. Методы расчета структурной надежности сетей сложной конфигурации. – М.: Знание, 1984. – 104 с.
41. Додонов А.Г., Кузнецова М.Г., Горбачик Е.С. Введение в теорию живучести вычислительных систем. – Киев: Наукова Думка, 1990. – 184 с.
42. Зайченко Ю.П. Задачи проектирования структуры распределенных вычислительных сетей. // Автоматика, 1981. № 4. – с. 40.
43. Зайченко Ю.П., Гонта Ю.В. Структурная оптимизация сетей ЭВМ. – К.: Техника, 1986. – 168 с.
44. Зайченко Ю.П., Кондратова Л.П. Некоторые проблемы топологического проектирования распределенных сетей ЭВМ большой размерности. // Автоматика, 1983. № 2. – с. 58.
45. Зайченко Е.Ю. Оптимизация структуры РВС при ограничениях на показатели своевременности доставки сообщений и живучести сети. Тезисы докладов конференции «Информационные сети и системы», г. Суздаль, октябрь. 1995. – с. 50.
46. Зайченко Е.Ю. Анализ и синтез структуры глобальных вычислительных сетей. – К.: ЗАО «Укрспецмонтажпроект». – 1998. – 108 с.
47. Зайченко Ю.П., Зайченко Е.Ю. Нахождение максимального потока и анализ показателей живучести сети при отказах. // Автоматика и телемеханика, 1996.

№ 6. – 140 с.

48. Зайченко Ю.П., Зайченко Е.Ю. Синтез структуры РВС при ограничениях на показатели своевременности доставки сообщений и живучести сети. // Проблемы информатики и управления, 1995. № 5. – 125с.
49. Зайченко Ю.П., Зайченко Е.Ю., Поспелов И.А. Комплекс программ анализа и синтеза структуры региональных и глобальных вычислительных сеетй. // Управляющие системы и машины, 2000. № 5/6. – 134 с.
50. Зайченко Ю.П., Ехсан Асл Руста. Аналіз та оптимізація часових характеристик розподілених обчислювальних мереж при додаткових обмеженнях. // Проблемы информатизации и управления. Сб. научн. тр. – Вып.3, Киев. – КМУГА, 1998. –350 с.
51. Зайченко Ю.П., Ехсан Асл Руста. Моделі та методи аналізу характеристик розподілених обчислювальних мереж при додаткових обмеженнях. // Наукові вісті НУТУ “КПІ”. – 1998. № 3. – 230 с.
52. Зайченко О.Ю., Печурін М.К. Синтез структури регіональних обчислювальних мереж при додаткових обмеженнях. // Вісник Київського університету. Сер. фіз.-мат. Науки. – 1999. – Вып.4. – 170с.
53. Клейнрок Л. Вычислительные системы с очередями. – М.: Мир, 1979. – 600с.
54. Клейнрок Л. Теория массового обслуживания. – М.: Машиностроение, 1979. – 432 с.
55. Кульгин М. Технология корпоративных сетей. – СПб. Питер, 1999. – 706 с.
56. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – СПб. Питер, 2001. – 672 с.
57. Ценк Андреас. Novell Netware 4.X. Пер. с немецкого. – К.: Торговое-издательское бюро BHV, 1996. – 784 с.
58. Шварц М. Сети ЭВМ: анализ и проектирование. – М.: Радио и связь. 1981. – 336 с.
59. Шварц М. Сети связи: протоколы, моделирование и анализ. – В 2-х частях. Ч.I. М.: Наука. Гл. ред. физ.-мат. лит., 1992. – 336 с., Ч. II. М.: Наука, 1992. – 272 с.

60. Янбых Г.Ф., Столяров Б.А. Оптимизация информационно-вычислительных сетей. – М.: Радио и связь, 1987. – 232 с.
61. Березин С. Интернет у вас дома. – СПб.: BHV – Санкт–Петербург, 1997. – 180с.
62. Буров Є. Комп'ютерні мережі. – Львів: БаК, 1999. – 190 с.
63. Габбасов Ю.Ф. Internet 2000. – СПб.: БХВ – Санкт–Петербург, 2000. – 260 с.
64. Дибкова. Л.М. Інформатика та комп'ютерна техніка. – К.: Академвидав, 2002. – 164 с.
65. Симонович С.В. Информатика. Базовый курс/ Симонович С.В. и др.– СПб: Издательство "Питер", 2001. – 130 с.
66. Макарова Н.В. Информатика. Практикум по технологии работы на компьютере / Под ред. проф. Н.В. Макаровой. – 3-е изд., перераб. – М.: Финансы и статистика, 2000. – 40 с.
67. Інформатика. Комп'ютерна техніка. Комп'ютерні технології: Підручник/ В. А. Баженов, П. С. Венгеровський, В. М. Горлач та ін. – К.: Каравела, 2003. – 225 с.
68. Кон А.И. Секреты Internet. Изд. Ростов н/Д: «Феникс», 2000. – 237 с.
69. Коцюбинский А.О., Грошев С.В. Современный самоучитель работы в сети Интернет. Быстрый старт.: Практ. пособие. – М.: Издательство ТРИУМФ, 2002. – 140 с.
70. Кулаков Ю.А., Луцкий Г.М. Локальные сети. – К.: Юниор, 1988. – 75-97 с.
71. Левин А.С. Самоучитель работы на компьютере. Изд. 6-е, испр. и перераб. – М.: "Нолидж", 2001. – 87 с.
72. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. – СПб.: Питер, 2002. – 170 с.
73. Симонович С.В., Евсеев Г. Новейший самоучитель по работе в Internet. М.: изд. «ДЕСС КОМ», 2000. –156 с.
74. Тхір І.Л., Калушка В.П., Юзьків А.В. Посібник користувача ПК. СМП "Астон", Тернопіль, 2002. – 120 с.
75. Энциклопедия Интернет. Под ред. Мелиховой. – СПб: Издательство "Питер", 2000. – 250 с.
76. Luke Pierce, Spyros Tragoudas .Nanopipelined Luke Pierce, Spyros Tragoudas. Nanopipelined threshold network synthesis. In ACM Journal on JETC, 10(2): 17-21, 2014.

77. Paul Wettin, Anuroop Vidapalapati, Amlan Gangul, Partha Pratim Pande. Complex network-enabled robust wireless network-on-chip architectures. In ACM Journal on JETC, 9(3): 24-30, 2013.
78. Chandan S., Saha S., Barrett C., Eubank S., Marathe A., Marathe M., Swarup S., Kumar VS Anil (2013) Modeling the Interaction between Emergency Communications and Behavior in the Aftermath of a Disaster. In Proceedings of The International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction (SBP), 7812 476-485. Washington DC, April 2-5, 2013.
79. Beckman R., Channakeshava K., Huang F., Kim J., Marathe A., Marathe M., Saha S., Pei G., Kumar VS Anil (2013) Integrated Multi-Network Modeling Environment for Spectrum Management. IEEE Journal on Selected Areas in Communication, special issue on Network Science, 31(6): 1158-1168.
80. Kuhlman C, Kumar VS Anil, Ravi S (2012) Controlling opinion bias in online social networks. InWebSci '12 Proceedings of the 3rd Annual ACM Web Science Conference, 165-174. Evanston, IL, June 22-24, 2012.
81. Christiaan Ottow, Frank van Vliet, Pieter-Tjerk de Boer, Aiko Pras: "Impact of IPv6 on Network and Web Application Penetration Testing". EUNICE 2012 (LNCS 7479), Budapest, Hungary, August 2012.
82. Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner R.W. Scheinhardt, Sandeep Juneja: "Some Advances in Importance Sampling of Reliability Models Based on Zero Variance Approximation". 9th International Workshop on Rare Event Simulation (RESIM 2012), Trondheim, Norway, June 2012.
83. Channakeshava K, Bisset K, Marathe M, Kumar VS Anil, Yardi S (2011) High Performance Scalable and Expressive Modeling Environment to Study Mobile Malware in Large Dynamic Networks. In Proceedings of 25th IEEE International Parallel & Distributed Processing Symposium. Anchorage, Alaska, May 16-20, 2011.
84. Marathe M, Kuhlman C, Kumar VS Anil, Ravi S, Rosenkrantz D (2011) Generalized contagions over social and communication networks: Foundations, computational models, and modeling environments. In Human Social Culture

Behavior Modeling Program Focus 2011 Conference (HSCB2011). Chantilly, VA, February 8-10, 2011.

85. Barrett C, Beckman R, Channakeshava K, Huang F, Kumar VS Anil, Marathe A, Marathe M, Pei G (2010) Cascading failures in multiple infrastructures: From transportation to communication network. In Proceedings of Interacting Critical Infrastructures for the 21st Century, 1-8. Beijing, China, September 20-22, 2010.
86. P. Dourish, R. E. Grinter, J. Delgado de la Flor, and M. Joseph, "Security in the wild: user strategies for managing security as an everyday, practical problem," Personal and Ubiquitous Computing (ACM/Springer), vol. 8, no. 6, pp. 391-401, November 2004.
87. R. J. Wirfs-Brock, "Refreshing patterns," IEEE Software, vol. 23, no. 3, pp. 45-47, May/June 2006.
88. M. J. Ranum, "Security: The root of the problem," ACM Queue (Special Issue: Surviving Network Attacks), vol. 2, no. 4, pp. 44-49, June 2004.
89. H. H. Thompson and R. Ford, "Perfect storm: The insider, naivety, and hostility," ACM Queue (Special Issue: Surviving Network Attacks), vol. 2, no. 4, pp. 58-65, June 2004.
90. M. Dynamic, Adaptive and Reconfigurable Systems Overview and Prospective Vision. Proc. of the 23rd Intl. Conference on Distributed Computing Systems Workshops. IEEE 2003. pp. 84-89.
91. Al-Bar, A. Wakeman, I. A Survey of Adaptive Applications in Mobile Computing. Proc. of the 21st Intl. Conference on Distributed Computing Systems (ICDCS). IEEE 2001. pp. 246-251.
92. Aldrich, J., Dooley, J., Mandelsohn, S., Rifkin, S. Providing Easier Access to Remote Objects in Client-Server Systems. Proc. of the Annual Hawaii Intl. Conference on System Sciences. IEEE 1998. pp. 366-375.
93. Anderson, D. P. BOINC: A System for Public-Resource Computing and Storage. Proc. of the Intl. Workshop on Grid Computing. IEEE. 2004. pp. 4-10.

94. Anderson, D.P., Cobb. J., Korpels, E., Lebofsky, M., Werthimer, D. SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM. Vol. 45, Nr. 11. ACM 2002. pp. 56-61.

95. Astley, M., Sturman, D., Agha, G. Customizable Middleware for Modular Distributed Software. Communications of the ACM. Vol. 44, Nr. 5. ACM 2001. pp. 99-107.

96. Avvenuti, M., Vecchio, A. Embedding Remote Object Mobility in Java RMI. Proc. of the 8th Intl. Workshop on Future Trends of Distributed Computing Systems (FTDCS 2001). IEEE 2001. pp. 98-104.

97. Balasubramanian1, J., Natarajan, B., Schmidt, D. C., Gokhale, A., Parsons, J., Deng, G. Middleware Support for Dynamic Component Updating. Proc. of the 7th Intl. Symposium on Distributed Objects and Applications (DOA). LNCS 3761. Springer 2005. pp. 978-996.

98. Balfanz, D., Gong, L. Experience with Secure Multi-Processing in Java. Proc. of the Intl. Conference on Distributed Computing Systems (ICDCS). IEEE 1998. pp. 398-405.

99. Batista, T., Rodriguez, N. Dynamic Reconfiguration of Component-Based Applications. Proc. of the Intl. Symposium on Software Engineering for Parallel and Distributed Systems. IEEE 2000. pp. 32-39.

100. Blohm, H. Hierarchical Arrangement of Modified Class Loaders and Token Driven Class Loaders and Methods of Use. United States Patent 7.536.412.B2. 2009.

101. Bouchenak, S., Hagimont, D. Pickling Threads State in the Java System. Proc. of the Intl. Conference on Technology of Object-Oriented Languages (TOOLS). IEEE 2000. pp. 22-32.

102. Caughey, S. J., Hagimont, D., Ingham, D. B. Deploying Distributed Objects on the Internet. Advances in Distributed Systems. LNCS 1752. Springer 2000. pp. 213-237.

103. Chakravarti, A., Wang, X., Hallstrom, J. O., Baumgartner, G. Implementation of Strong Mobility for Multi-Threaded Agents in Java. Intl. Conference on Parallel Processing. IEEE 2003. pp. 321-330. 597-610.

104. Clark, D. Face-to-Face with Peer-to-Peer Networking. IEEE Computer. Vol. 34, Nr. 1. IEEE 2001. pp. 18-21.

105. Clark, D. Network Nirvana and the Intelligent Device. IEEE Concurrency. Vol. 7, Nr. 2. IEEE 1999. pp. 16-19.

106. Clarke, M., Blair, G. S., Coulson, G., Parlavantzas, N. An Efficient Component Model for the Construction of Adaptive Middleware. Proc. of the Intl. Conference on Middleware 2001. LNCS 2218. Springer 2001. pp. 160-178.

107. Corwin, J., Bacon, D.F., Grove, D., Murthy, C. MJ: A Rational Module System for Java and its Applications. Proc. of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA). ACM 2003. pp. 241-254.

108. Daynes, L., Czajkowski, G. Sharing the Runtime Representation of Classes across Class Loaders. Proc. of the European Conference on Object-Oriented Programming (ECOOP). LNCS 3586. Springer 2005. pp. 97-120.

109. Diaz y Carballo J.C., Dearle A., Connor R.C.H. Thin Servers - An Architecture to Support Arbitrary Placement of Computation in the Internet. Proc. of the Intl. Conference on Enterprise Information Systems (ICEIS 2002). ICEIS 2002. pp. 1080-1085.

110. Dikaiakos, M. D., Pallis, G., Katsaros, D., Mehra, P., Vakali. A. Cloud Computing - Distributed Internet Computing for IT and Scientific Research. IEEE Internet Computing. Vol. 13, Nr. 5. IEEE 2009. pp. 10-13.

111. Eberhard, J., Tripathi, A. Efficient Object Caching for Distributed Java RMI Applications. Proc. of the Intl. Conference on Middleware. LNCS 2218. Springer 2001. pp. 15-35.

112. Geihs, K. Selbst-Adaptive Software. Informatik Spektrum. Springer 2007. pp. 133-145.

113. Grossmann, R. L. The Case for Cloud Computing. IEEE IT Professional. Vol. 11, Nr. 2. IEEE 2009. pp. 23-27.

114. Paal, S., Kammüller, R., Freisleben, B. Separating the Concerns of Distributed Deployment and Dynamic Composition in Internet Application Systems. Proceedings of the 4th Intl. Conference on Distributed Objects and Applications (DOA 2003). LNCS 2888. Catania, Italy. Springer 2003. pp. 1292.

115. Elmenreich, W., D'Souza, R., Bettstetter, C. and de Meer, H., A Survey of Models and Design Methods for Self-Organizing Networked Systems, Proceedings of the Fourth International Workshop on Self-Organizing Systems, Springer Verlag, 2009.

116. Crisostomo, S., Barros, J. and Bettstetter, C., Flooding the Network: Multipoint Relays versus Network Coding, Proceedings of the 4th IEEE International Conference on Circuits and Systems for Communications (ICCSC), 2008.

117. Pei G, Kumar VS Anil Distributed Approximation Algorithms for Maximum Link Scheduling and Local Broadcasting in the Physical Interference Model. In IEEE INFOCOM 2013, 1339-1347. Turin, Italy, April 14-19, 2013.

118. Васильків Н. М. Опорний конспект лекцій з дисципліни “Ефективність інформаційних систем” / Васильків Н. М. – Тернопіль: Економічна думка, 2005. – 98 с.

119. Haifeng Yu, Amin Vahdat Ecient numerical error bounding for replicated network services. Computer science department. Duke university. USA, June 13, 2000.

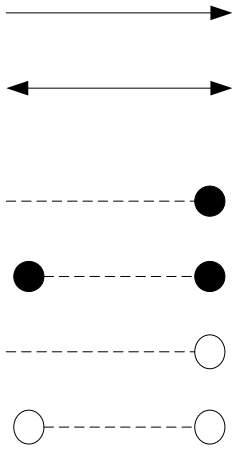
120. Шоботенко Н.М. Модель оцінки ефективності використання методу автоматизації / Н.М. Шоботенко // Вісник «Молодий вчений».- Київ: НТУ «Київський політехнічний інститут», 2014.- №6 (09).- С. 38-43.

121. Вовк П.Б. Проблеми проектування вбудованих систем / Вовк П.Б., Усійчук А.П. // Науковий журнал "Комп’ютерно-інтегровані технології: освіта, наука, виробництво ". - Луцьк: «Луцький інститут розвитку людини», 2013.- №11.- С. 82-87.

ДОДАТКИ

ДОДАТОК А

А.1 Умовно-графічні позначення за книжкою Г.Буча «Об'єктно-орієнтований аналіз та програмування».



Принцип насліування

Принцип
інстанціювання

Принцип використання

А.2 Математичний розрахунок ефективності модульного мережного інтерфейсу навігаційного сервісу кіберфізичних систем

Визначення терміну розробки модульного мережного інтерфейсу НС кіберфізичних систем.Ступінь новизни – Б. Код ступені новизни – Значення $К_н$ – Б (1). Ступінь охвату реалізованих функцій – значення $К_т$ – 83% (0,6).

$Т_{тз} = (385 \times 0,8) \times 0,12 \times 1 = 36,96$ (люд./год.);

$Т_{тп} = (385 \times 0,8) \times 0,15 \times 1 = 46,2$ (люд./год.);

$Т_{рп} = (385 \times 0,8) \times 0,58 \times 1 \times 0,6 = 107,184$ (люд./год.);

$Т_{вп} = (385 \times 0,8) \times 0,15 \times 1 = 46,2$ (люд./год.).

Розрахунок трудомісткості розробки ПЗ автоматизованих розрахунків представлено нами у формулах розрахунку трудомісткості розробки ПЗ.

Розраховуємо термін розробки спеціалізованого мережного інтерфейсу НС кіберфізичних систем:

$Т_{кк} = 1 \ N_{пз} = 1 \times 58 = 58$ (люд./год.);

$T_{нк} = 0,15\ \ N_{пз} = 0,15 \times 58 = 8,7$ (люд./год.);

Розраховуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем:

$T_{кк} = 1\ \ N_{вп} = 1 \times 24 = 24$ (люд./год.);

$T_{нк} = 0,15\ \ N_{вп} = 0,15 \times 24 = 3,64$ (люд./год.).

Розраховуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем з використанням вдосконаленого методу на основі принципу наслідування:

$T_{кк} = 1\ \ N_{рп} = 1 \times 18 = 18$ (люд./год.);

$T_{нк} = 0,15\ \ N_{рп} = 0,15 \times 18 = 2,7$ (люд./год.).

Розраховуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем з використанням вдосконаленого методу на основі принципу інстанціювання:

$T_{кк} = 1\ \ N_{тз} = 1 \times 2 = 2$ (люд./год.);

$T_{нк} = 0,15\ \ N_{тз} = 0,15 \times 2 = 0,3$ (люд./год.).

Розраховуємо термін розробки модульного мережного інтерфейсу НС кіберфізичних систем з використанням вдосконаленого методу на основі принципу використання:

$T_{кк} = 1\ \ N_{тп} = 1 \times 12 = 12$ (люд./год.);

$T_{нк} = 0,15\ \ N_{тп} = 0,15 \times 12 = 1,8$ (люд./год.);

Розраховуємо термін розробки ПЗ для вдосконалених методів розробки модульного мережного інтерфейсу на основі трьох принципів, коливався від ~25- ~75люд./год., візьмемо значення 58 люд./год.:

$T_{пз} = 1\ \ N_{пз} = 1 \times 58 = 58$ (люд./год.).

Визначення часу реакції модульного мережного інтерфейсу НС кіберфізичних систем. Для розрахунку часу реакції спочатку необхідно розрахувати функцій $\overline{r}(\lambda)$, $\overline{k}(\lambda)$, $\overline{t}_{оч.}(\lambda)$, $\overline{t}_{мі}(\lambda)$, для спеціалізованого мережного інтерфейсу НС кіберфізичних систем зробимо відповідні розрахунки і складемо відповідні

таблиці розрахунків. Де, λ_0 - кількість запитів від клієнта до сервера у кожному з інтерфесів.

Таблиця А.1 Час реакції для спеціалізованого мережного інтерфейсу НС кіберфізичних систем

λ_0	$\lambda=\lambda0*9$	$p=\lambda/100$	r	k	$t_{оч.}$	$t_{смі}$
1	9	0,09	0,0089	0,0989	0,001	0,01091
2	18	0,18	0,0395	0,2195	0,0023	0,012
3	27	0,27	0,0998	0,3698	0,0037	0,0134
4	36	0,36	0,2025	0,56	0,0056	0,0156
5	45	0,45	0,368	0,818	0,0082	0,018
6	54	0,54	0,634	1,1739	0,0117	0,022
7	63	0,63	1,0727	1,7027	0,017	0,027
8	72	0,72	1,851	2,571	0,0257	0,0357
9	81	0,81	3,45	4,263	0,0423	0,0526
10	90	0,9	8,1	9	0,09	0,1
11	99	0,99	98,01	99	0,99	1
12	**99**	**0,99**	**98,01**	**99**	**0,99**	**1,06**

Для розрахунку часу реакції спочатку необіхно розрахувати функцій $\overline{r}(\lambda)$, $\overline{k}(\lambda)$, $\overline{t}_{оч.}.(\lambda)$, $\overline{t}_{мн}(\lambda)$ для вдосконалених методів розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування зробимо відповідні розрахунки і складемо відповідні таблиці розрахунків.

Таблиця А. 2 Час реакції для вдосконалених методів розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу наслідування

λ0	λ=λ0*8	p=λ/100	r	k	$t_{оч.}$	$t_{мн}$
1	8,8	0,088	0,0085	0,096	0,0009	0,011
2	17,6	0,176	0,036	0,21	0,002	0,01
3	26,4	0,264	0,0947	0,358	0,004	0,013
4	35,2	0,352	0,19	0,54	0,005	0,015
5	44	0,44	0,345	0,786	0,008	0,018
6	52,8	0,528	0,59	1,119	0,011	0,021
7	61,6	0,616	0,988	1,60	0,016	0,026
8	70,4	0,704	1,67	2,38	0,024	0,033
9	79,2	0,792	3,015	3,80	0,038	0,048
11	88	0,88	6,45	7,33	0,073	0,083
12	**96,8**	**0,968**	**29,282**	**30,25**	**0,3**	**0,3**

Для розрахунку часу реакції спочатку необхідно розрахувати функцій $\overline{r}(\lambda)$, $\overline{k}(\lambda)$, $\overline{t}_{оч.}(\lambda)$, $\overline{t}_{мін.}(\lambda)$ для вдосконаленого методу розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання зробимо відповідні розрахунки і складемо відповідні таблиці розрахунків.

Таблиця А. 3 Час реакції для вдосконаленого методу розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу інстанціювання

λ0	λ=λ0*6	p=λ/100	r	k	$t_{оч.}$	$t_{мін.}$
1	10,8	0,108	0,013	0,12	0,0012	0,011
3	21,6	0,216	0,059	0,275	0,0027	0,012
5	32,4	0,324	0,155	0,479	0,00479	0,0148
7	43,2	0,432	0,328	0,76	0,0076	0,0176
9	54	0,54	0,63	1,17	0,012	0,022
10	64,8	0,648	1,193	1,84	0,018	0,028
12	**75,6**	**0,756**	**2,34**	**3,098**	**0,03**	**0,04**
14	86,4	0,864	5,49	6,35	0,063529	0,07
15	93	0,93	12,35571	13,286	0,132857	0,14
16	96	0,96	23,04	24	0,24	0,25
17	97,2	0,972	33,74	34,714	0,347	0,357
18	98,7	0,987	74,9	75,9	0,759	0,769
19	98,4	0,984	60,516	61,5	0,615	0,625

Для розрахунку часу реакції спочатку необіхно розрахувати функцій $\overline{r}(\lambda)$, $\overline{k}(\lambda)$, $\overline{t}_{оч.}(\lambda)$, $\overline{t}_{мв}(\lambda)$ для вдосконаленого методу розробки модульного мережного інтерфейсу на основі принципу використання зробимо відповідні розрахунки і складемо відповідні таблиці розрахунків.

Таблиця А. 4 Час реакції для вдосконаленого методу розробки модульного мережного інтерфейсу НС кіберфізичних систем на основі принципу використання

λ0	λ=λ0*4	p=λ/100	r	k	$t_{оч.}$	$t_{мв}$
2	10	0,1	0,011	0,11	0,0011	0,11
5	20	0,2	0,05	0,25	0,0025	0,125
7	30	0,3	0,12	0,42	0,0042	0,14
10	40	0,4	0,26	0,66	0,0066	0,16
12	**50**	**0,5**	**0,5**	**1**	**0,01**	**0,2**
15	60	0,6	0,9	1,5	0,015	0,25
17	70	0,7	1,63	2,33	0,02	0,33
20	80	0,8	3,2	4	0,04	0,5
22	88	0,88	6,45	7,33	0,07	0,83

Впровадження даного модульного мережного інтерфейсу сприяє покращенню надійності, точності та розв’язальної спроможності, що збільшить швидкість одержання результату на 7,3%.

ДОДАТОК Б

Б.1 Лістинг програми клієнта (TrackClientPacketHandler) з використанням модульного мережного інтерфейсу на мові Java та JavaScript

Наведений код частково базувався на функціях одержаних з відкритих джерел та адаптованих автором для розв'язання задачі модульності мережного інтерфейсу.

```
import java.lang.*;
import java.util.*;
import java.io.*;
import java.net.*;
import java.sql.*;

public class TrackClientPacketHandler
    extends AbstractClientPacketHandler
{


    public  static int          DATA_FORMAT_OPTION          = 1;
    public  static       boolean ESTIMATE_ODOMETER          = false;
    public  static       long    SIMEVENT_DIGITAL_INPUTS    = 0x0000L;
    private static       boolean DFT_INSERT_EVENT           = true;
    private static       boolean INSERT_EVENT               = DFT_INSERT_EVENT;
    public  static       double  MINIMUM_SPEED_KPH          = 0.0;
    public static final double  KILOMETERS_PER_KNOT     = 1.85200000;
    public static final double  KNOTS_PER_KILOMETER     = 1.0
    private static       boolean IGNORE_NMEA_CHECKSUM       = false;
    private  static  final  TimeZone  gmtTimezone                              =
DateTime.getGMTTimeZone();
    private static final int InputStatusCodes_ON[] = new int[] {
        StatusCodes.STATUS_INPUT_ON_00,
        StatusCodes.STATUS_INPUT_ON_01,
        StatusCodes.STATUS_INPUT_ON_02,
        StatusCodes.STATUS_INPUT_ON_03,
        StatusCodes.STATUS_INPUT_ON_04,
        StatusCodes.STATUS_INPUT_ON_05,
        StatusCodes.STATUS_INPUT_ON_06,
        StatusCodes.STATUS_INPUT_ON_07,
        StatusCodes.STATUS_INPUT_ON_08,
        StatusCodes.STATUS_INPUT_ON_09,
        StatusCodes.STATUS_INPUT_ON_10,
        StatusCodes.STATUS_INPUT_ON_11,
        StatusCodes.STATUS_INPUT_ON_12,
        StatusCodes.STATUS_INPUT_ON_13,
        StatusCodes.STATUS_INPUT_ON_14,
        StatusCodes.STATUS_INPUT_ON_15
    };
      private static final int InputStatusCodes_OFF[] = new int[] {
        StatusCodes.STATUS_INPUT_OFF_00,
        StatusCodes.STATUS_INPUT_OFF_01,
        StatusCodes.STATUS_INPUT_OFF_02,
        StatusCodes.STATUS_INPUT_OFF_03,
        StatusCodes.STATUS_INPUT_OFF_04,
        StatusCodes.STATUS_INPUT_OFF_05,
        StatusCodes.STATUS_INPUT_OFF_06,
        StatusCodes.STATUS_INPUT_OFF_07,
```

```
        StatusCodes.STATUS_INPUT_OFF_08,
        StatusCodes.STATUS_INPUT_OFF_09,
        StatusCodes.STATUS_INPUT_OFF_10,
        StatusCodes.STATUS_INPUT_OFF_11,
        StatusCodes.STATUS_INPUT_OFF_12,
        StatusCodes.STATUS_INPUT_OFF_13,
        StatusCodes.STATUS_INPUT_OFF_14,
        StatusCodes.STATUS_INPUT_OFF_15
    };

    private String          sessionID                   = null;
    private GPSEvent        gpsEvent                    = null;
    private String          ipAddress                   = null;
    private int             clientPort                  = 0;
    public TrackClientPacketHandler()
    {
        super();

    }

    {
        super.sessionStarted(inetAddr, isTCP, isText);
        super.clearTerminateSession();

    this.ipAddress           = (inetAddr != null)? inetAddr.getHostAddress() :
null;
        this.clientPort       = this.getSessionInfo().getRemotePort();

    }
    public   void   sessionTerminated(Throwable   err,   long   readCount,   long
writeCount)
    {
        super.sessionTerminated(err, readCount, writeCount);
    }

        public String getSessionID()
    {
        if (!StringTools.isBlank(this.sessionID)) {
            return this.sessionID;
        } else
        if (this.gpsEvent != null) {
            return this.createTcpSessionID(this.gpsEvent.getDevice());
        } else {
            return null;
        }
    }

       public   static   final   boolean   USE_STANDARD_TCP_SESSION_ID   =   true;
public static String getTcpSessionID(Device device)
    {
        if (USE_STANDARD_TCP_SESSION_ID || (device == null)) {
            return DCServerFactory.getTcpSessionID(device);
        } else {
                        String uid = device.getUniqueID();
            int    u   = uid.lastIndexOf("_");
            if (u < 0) {
                return DCServerFactory.getTcpSessionID(device);
            } else {
                String    sessID    =    device.getAccountID()    +    "/"    +
uid.substring(0,u+1);
                return sessID;
            }
        }
    }
```

```
    public String createTcpSessionID(Device device)
    {
        if (USE_STANDARD_TCP_SESSION_ID || (device == null)) {
            return DCServerFactory.createTcpSessionID(device);
        } else {
            String uid = device.getUniqueID();
            int    u   = uid.lastIndexOf("_");
            if (u < 0) {
                return DCServerFactory.createTcpSessionID(device);
            } else {
                String    sessID    =    device.getAccountID()    +    "/"    +
uid.substring(0,u+1);
                device.setLastTcpSessionID(sessID);
                return sessID;
            }
        }
    }

    public int getActualPacketLength(byte packet[], int packetLen)
    {
        if (Constants.ASCII_PACKETS) {

        } else {
                  return ServerSocketThread.PACKET_LEN_LINE_TERMINATOR;
        }

    }

        private void setTerminate()
    {
        this.terminate = true;
    }

    public boolean terminateSession()
    {
        return this.terminate;
    }

    public byte[] getInitialPacket()
        throws Exception
    {

    }

    public byte[] getHandlePacket(byte pktBytes[])
    {

       if ((pktBytes != null) && (pktBytes.length > 0)) {

            Print.logInfo("Recv[HEX]: " + StringTools.toHexString(pktBytes));
            String     s     =     StringTools.toStringValue(pktBytes).trim();
Print.logInfo("Recv[TXT]: " + s);

                        byte rtn[] = null;
            switch (DATA_FORMAT_OPTION) {
                case  1 : rtn = this.parseInsertRecord_ASCII_01(s); break;
                case  2 : rtn = this.parseInsertRecord_ASCII_02(s); break;
                case  3 : rtn = this.parseInsertRecord_ASCII_03(s); break;
                case  9 : rtn = this.parseInsertRecord_RTProps (s); break;
                case 11 : rtn = this.parseInsertRecord_Device_1(s); break;
                default: Print.logError("Unspecified data format"); break;
            }
            return rtn;
        } else {
```

```
                    Print.logInfo("Empty packet received ...");
            return null;
        }


    }

    public byte[] getFinalPacket(boolean hasError)
        throws Exception
    {

        return null;
    }

        private boolean parseInsertRecord_Common(GPSEvent gpsEv)
    {
        long fixtime    = gpsEv.getTimestamp();
        int  statusCode = gpsEv.getStatusCode();

            if (fixtime <= 0L) {
            Print.logWarn("Invalid date/time");
            fixtime = DateTime.getCurrentTimeSec();
            gpsEv.setTimestamp(fixtime);
        }

            if (!gpsEv.isValidGeoPoint()) {
            Print.logWarn("Invalid  lat/lon:  "  +  gpsEv.getLatitude()  +  "/"  +
gpsEv.getLongitude());
            gpsEv.setLatitude(0.0);
            gpsEv.setLongitude(0.0);
        }
        GeoPoint geoPoint = gpsEv.getGeoPoint();
        if (gpsEv.getSpeedKPH() < MINIMUM_SPEED_KPH) {
            gpsEv.setSpeedKPH(0.0);
            gpsEv.setHeading(0.0);
        }

          Device device = gpsEv.getDevice();
          double odomKM = 0.0;
            odomKM = (ESTIMATE_ODOMETER && geoPoint.isValid())?
                device.getNextOdometerKM(geoPoint) :
                device.getLastOdometerKM();
        } else {
            odomKM = device.adjustOdometerKM(odomKM);
        }
                Print.logInfo("Odometer KM: " + odomKM);
        gpsEv.setOdometerKM(odomKM);
       if (SIMEVENT_GEOZONES && geoPoint.isValid()) {
            java.util.List<Device.GeozoneTransition>            zone            =
device.checkGeozoneTransitions(fixtime, geoPoint);
            if (zone != null) {
                for (Device.GeozoneTransition z : zone) {
                    gpsEv.insertEventData(z.getTimestamp(),    z.getStatusCode(),
z.getGeozone());
                    Print.logInfo("Geozone    : " + z);
                }
            }
        }

        if (gpsEv.hasInputMask() && (gpsEv.getInputMask() >= 0L)) {
            long gpioInput = gpsEv.getInputMask();
            if (SIMEVENT_DIGITAL_INPUTS > 0L)
         long    chgMask    =    (device.getLastInputState()    ^    gpioInput)    &
SIMEVENT_DIGITAL_INPUTS;
                if (chgMask != 0L) {
```

```
                    an input state has changed
                    for (int b = 0; b <= 15; b++) {
                        long m = 1L << b;
                        if ((chgMask & m) != 0L) {
                             this bit changed
                            int     inpCode   =   ((gpioInput   &   m)   !=   0L)?
InputStatusCodes_ON[b] : InputStatusCodes_OFF[b];
                            long inpTime = fixtime;
                            gpsEv.insertEventData(inpTime, inpCode);
                            Print.logInfo("GPIO          :          "          +
StatusCodes.GetDescription(inpCode,null));
                        }
                    }
                }
            }
            device.setLastInputState(gpioInput & 0xFFFFL);
        }

        return true;

    }

       private byte[] parseInsertRecord_ASCII_01(String s)
    {
        Print.logInfo("Parsing: " + s);
        if (s == null) {
            Print.logError("String is null");
            return null;
        }
        String fld[] = StringTools.parseString(s, ',');
        if ((fld == null) || (fld.length < 7)) {
            Print.logWarn("Invalid number of fields");
            return null;
        }

        String   modemID    = fld[0].toLowerCase();
        long     fixtime    = this._parseDate(fld[1],fld[2]);
        int      statusCode = StatusCodes.STATUS_LOCATION;
        double   latitude   = StringTools.parseDouble(fld[3],0.0);
        double   longitude  = StringTools.parseDouble(fld[4],0.0);
        double   speedKPH   = StringTools.parseDouble(fld[5],0.0);
        double   heading    = StringTools.parseDouble(fld[6],0.0);
        double   altitudeM  = 0.0;
        if (StringTools.isBlank(modemID)) {
            Print.logWarn("ModemID not specified!");
            return null;
        }

            this.gpsEvent = new GPSEvent(Main.getServerConfig(null),
            this.ipAddress, this.clientPort, modemID);
        Device device = this.gpsEvent.getDevice();
        if (device == null) {

            return null;
        }
        this.sessionID = this.createTcpSessionID(device);
        this.gpsEvent.setTimestamp(fixtime);
        this.gpsEvent.setStatusCode(statusCode);
        this.gpsEvent.setLatitude(latitude);
        this.gpsEvent.setLongitude(longitude);
        this.gpsEvent.setSpeedKPH(speedKPH);
        this.gpsEvent.setHeading(heading);
        this.gpsEvent.setAltitude(altitudeM);
        if (this.parseInsertRecord_Common(this.gpsEvent)) {
            return null;
```

```
    } else {
        return null;
    }

}

private long _parseDate(String yyyymmdd, String hhmmss)
{
    String d[] = StringTools.parseString(yyyymmdd,"/");
    String t[] = StringTools.parseString(hhmmss  ,":");
    if ((d.length != 3) && (t.length != 3)) {
                    return 0L;
    } else {
        int YY = StringTools.parseInt(d[0],0);
        int MM = StringTools.parseInt(d[1],0);
        int DD = StringTools.parseInt(d[2],0);
        int hh = StringTools.parseInt(t[0],0);
        int mm = StringTools.parseInt(t[1],0);
        int ss = StringTools.parseInt(t[2],0);
        if (YY < 100) { YY += 2000; }
        DateTime dt = new DateTime(gmtTimezone,YY,MM,DD,hh,mm,ss);
        return dt.getTimeSec();
    }
}
private long _parseDate(long yyyymmdd, long hhmmss)
{
    if ((yyyymmdd <= 0L) || (hhmmss < 0L)) {
        return 0L;
    } else {
        int YY = (int)((yyyymmdd / 10000L)       );
        int MM = (int)((yyyymmdd /   100L) % 100L);
        int DD = (int)((yyyymmdd /     1L) % 100L);
        int hh = (int)((hhmmss   / 10000L)       );
        int mm = (int)((hhmmss   /   100L) % 100L);
        int ss = (int)((hhmmss   /     1L) % 100L);
        if (YY < 100) { YY += 2000; }
        DateTime dt = new DateTime(gmtTimezone,YY,MM,DD,hh,mm,ss);
        return dt.getTimeSec();
    }
}
    private long _parseDate(long yyyymmddhhmmss)
{
    if (yyyymmddhhmmss <= 0L) {
        return 0L;
    } else {
        int YY = (int)((yyyymmddhhmmss / 10000000000L)       );
        int MM = (int)((yyyymmddhhmmss /   100000000L) % 100L);
        int DD = (int)((yyyymmddhhmmss /     1000000L) % 100L);
        int hh = (int)((yyyymmddhhmmss /       10000L) % 100L);
        int mm = (int)((yyyymmddhhmmss /         100L) % 100L);
        int ss = (int)((yyyymmddhhmmss /           1L) % 100L);
        if (YY < 100) { YY += 2000; }
        DateTime dt = new DateTime(gmtTimezone,YY,MM,DD,hh,mm,ss);
        return dt.getTimeSec();
    }
}

private byte[] parseInsertRecord_ASCII_02(String s)
{

    Print.logInfo("Parsing: " + s);
           if (s == null) {
        Print.logError("String is null");
        return null;
    }
```

```
                String fld[] = StringTools.parseString(s, '/');
        if ((fld == null) || (fld.length < 3)) {
            Print.logWarn("Invalid number of fields");
            return null;
        }

        String   accountID  = fld[0].toLowerCase();
        String   deviceID   = fld[1].toLowerCase();
        Nmea0183 gprmc      = new Nmea0183(fld[2], IGNORE_NMEA_CHECKSUM);
        long     fixtime    = gprmc.getFixtime();
        int      statusCode = StatusCodes.STATUS_LOCATION;
        double   latitude   = gprmc.getLatitude();
        double   longitude  = gprmc.getLongitude();
        double   speedKPH   = gprmc.getSpeedKPH();
        double   heading    = gprmc.getHeading();
        double   altitudeM  = 0.0;
          if     (deviceID.equals("")     ||     deviceID.equals("_mid_"))     {
deviceID  = accountID;
            accountID = "";
        }

                if (StringTools.isBlank(deviceID)) {
            Print.logWarn("DeviceID not specified!");
            return null;
        }

            this.gpsEvent = new GPSEvent(Main.getServerConfig(null),
            this.ipAddress, this.clientPort, accountID, deviceID);
        Device device = this.gpsEvent.getDevice();
        if (device == null) {
            return null;
        }
        this.sessionID = this.createTcpSessionID(device);
        this.gpsEvent.setTimestamp(fixtime);
        this.gpsEvent.setStatusCode(statusCode);
        this.gpsEvent.setLatitude(latitude);
        this.gpsEvent.setLongitude(longitude);
        this.gpsEvent.setSpeedKPH(speedKPH);
        this.gpsEvent.setHeading(heading);
        this.gpsEvent.setAltitude(altitudeM);
        if (this.parseInsertRecord_Common(this.gpsEvent)) {
                        return null;
        } else {
            return null;
        }

    }
        private byte[] parseInsertRecord_ASCII_03(String s)
    {

        Print.logInfo("Parsing: " + s);
               if (s == null) {
            Print.logError("String is null");
            return null;
        }

        String kv = null;
        int kvPos = s.indexOf(';');
        if (kvPos >= 0) {
            kv = s.substring(kvPos+1);
            s  = s.substring(0,kvPos);
        }
        String fld[] = StringTools.parseString(s, ',');
        if ((fld == null) || (fld.length < 10)) {
            Print.logWarn("Invalid number of fields");
```

```
            return null;
        }

        int      statusCode = StatusCodes.STATUS_LOCATION;
        int      sequence   = StringTools.parseInt(fld[0],0);
        int      eventCode  = StringTools.parseInt(fld[1],0);
        String   modemID    = fld[2].toLowerCase();
        int      format     = StringTools.parseInt(fld[3],0);
        long     yyyymmdd   = StringTools.parseLong(fld[4],0L);
        long     hhmmss     = StringTools.parseLong(fld[5],0L);
        long     fixtime    = this._parseDate(yyyymmdd,hhmmss);
        boolean  validGPS   = fld[6].equals("1");
        double    latitude    = validGPS? StringTools.parseDouble(fld[ 8],0.0) :
0.0;
        double    longitude   = validGPS? StringTools.parseDouble(fld[ 9],0.0) :
0.0;
        double        heading          =   validGPS   &&   (fld.length   >   10)?
StringTools.parseDouble(fld[10],0.0) : 0.0;
        double        speedKPH         =   validGPS   &&   (fld.length   >   11)?
StringTools.parseDouble(fld[11],0.0) : 0.0;
        double         altitudeM      =   validGPS   &&   (fld.length   >   12)?
StringTools.parseDouble(fld[12],0.0) : 0.0;

        if (StringTools.isBlank(modemID)) {
            Print.logWarn("ModemID not specified!");
            return null;
        }

            this.gpsEvent = new GPSEvent(Main.getServerConfig(null),
            this.ipAddress, this.clientPort, modemID);
        Device device = this.gpsEvent.getDevice();
        if (device == null) {
            return null;
        }
        this.sessionID = this.createTcpSessionID(device);
        this.gpsEvent.setTimestamp(fixtime);
        this.gpsEvent.setStatusCode(statusCode);
        this.gpsEvent.setLatitude(latitude);
        this.gpsEvent.setLongitude(longitude);
        this.gpsEvent.setSpeedKPH(speedKPH);
        this.gpsEvent.setHeading(heading);
        this.gpsEvent.setAltitude(altitudeM);

        if (this.parseInsertRecord_Common(this.gpsEvent)) {
                        return null;
        } else {
            return null;
        }

    }

    private  static  String  RTP_ACCOUNT[]          =  new  String[]  {  "acct"  ,
"accountid"    };
    private static String RTP_DEVICE[]      = new String[] { "dev"  , "deviceid"
};
    private static String RTP_MODEMID[]      = new String[] { "mid"  , "modemid"
, "uniqueid"    , "imei" };
    private  static  String  RTP_TIMESTAMP[]      =  new  String[]  {  "ts"     ,
"timestamp"    , "time"        };
    private  static  String  RTP_STATUSCODE[]    =  new  String[]  {  "code"  ,
"statusCode"   };
    private static String RTP_GEOPOINT[]    = new String[] { "gps"  , "geopoint"
};
    private static String RTP_GPSAGE[]       = new String[] { "age"  , "gpsAge"
};
```

```
    private static String RTP_SATCOUNT[]    = new String[] { "sats" , "satCount"
};
    private static String RTP_SPEED[]        = new String[] { "kph"  , "speed"
, "speedKph"    };
    private static String RTP_HEADING[]     = new String[] { "dir"  , "heading"
};
    private static String RTP_ALTITUDE[]     = new String[] { "alt"  , "altm"
, "altitude"    };
    private static String RTP_ODOMETER[]    = new String[] { "odom" , "odometer"
};
    private static String RTP_INPUTMASK[]     = new String[] { "gpio" ,
"inputMask"    };
    private static String RTP_SERVERID[]    = new String[] { "dcs"  , "serverid"
};
    private static String RTP_ACK[]         = new String[] { "ack"  };
    private static String RTP_NAK[]         = new String[] { "nak"  };

    /* parse and insert data record */
    private byte[] parseInsertRecord_RTProps(String s)
    {

        Print.logInfo("Parsing: " + s);

                if (StringTools.isBlank(s)) {
            Print.logError("Packet string is blank/null");
            return null;
        }

        RTProperties rtp = new RTProperties(s);
        String   accountID  = rtp.getString(RTP_ACCOUNT,   null);
        String   deviceID   = rtp.getString(RTP_DEVICE,    null);
        String   mobileID   = rtp.getString(RTP_MODEMID,   null);
        long     fixtime    = rtp.getLong(  RTP_TIMESTAMP, 0L);
        int                                    statusCode      =      rtp.getInt(
RTP_STATUSCODE,StatusCodes.STATUS_LOCATION);
        String   gpsStr     = rtp.getString(RTP_GEOPOINT,  null);
        long     gpsAge     = rtp.getLong(  RTP_GPSAGE,    0L);
        int      satCount   = rtp.getInt(   RTP_SATCOUNT,  0);
        double   speedKPH   = rtp.getDouble(RTP_SPEED,     0.0);
        double   heading    = rtp.getDouble(RTP_HEADING,   0.0);
        double   altitudeM  = rtp.getDouble(RTP_ALTITUDE,  0.0);
        double   odomKM     = rtp.getDouble(RTP_ODOMETER,  0.0);
        long     gpioInput  = rtp.getLong(  RTP_INPUTMASK, -1L);
        String   dcsid      = rtp.getString(RTP_SERVERID,  null);
        String   ack        = rtp.getString(RTP_ACK,       null);
        String   nak        = rtp.getString(RTP_NAK,       null);
        GeoPoint geoPoint   = new GeoPoint(gpsStr);
        if (StringTools.isBlank(mobileID)) {
            Print.logError("UniqueID/ModemID not specified!");
            return (nak != null)? (nak+"\n").getBytes() : null;
        }

        String     dcsName     =     !StringTools.isBlank(dcsid)?     dcsid     :
Main.getServerName();
        DCServerConfig dcserver = DCServerFactory.getServerConfig(dcsName);
        if (dcserver == null) {
            Print.logWarn("DCServer name not registered: " + dcsName);
        }

        boolean hasAcctDevID = false;
        if (!StringTools.isBlank(accountID)) {
            if (StringTools.isBlank(deviceID)) {
                Print.logError("'deviceid' required if 'accountid' specified");
                return (nak != null)? (nak+"\n").getBytes() : null;
            } else
```

```
            if (!StringTools.isBlank(mobileID)) {
                Print.logError("'mobileID'    not    allowed    if    'accountid'
specified");
                return (nak != null)? (nak+"\n").getBytes() : null;
            }
            hasAcctDevID = true;
        } else
        if (!StringTools.isBlank(deviceID)) {
            Print.logError("'accountid' required if 'deviceid' specified");
            return (nak != null)? (nak+"\n").getBytes() : null;
        } else
        if (StringTools.isBlank(mobileID)) {
            Print.logError("'mobileID' not specified");
            return (nak != null)? (nak+"\n").getBytes() : null;
        }

        this.gpsEvent = hasAcctDevID?
            new  GPSEvent(dcserver,  this.ipAddress,  this.clientPort,  accountID,
deviceID) :
            new GPSEvent(dcserver, this.ipAddress, this.clientPort, mobileID);
        Device device = this.gpsEvent.getDevice();
        if (device == null) {
            return (nak != null)? (nak+"\n").getBytes() : null;
        }
        this.sessionID = this.createTcpSessionID(device);

        this.gpsEvent.setTimestamp(fixtime);
        this.gpsEvent.setStatusCode(statusCode);
        this.gpsEvent.setGeoPoint(geoPoint);
        this.gpsEvent.setGpsAge(gpsAge);
        this.gpsEvent.setSatelliteCount(satCount);
        this.gpsEvent.setSpeedKPH(speedKPH);
        this.gpsEvent.setHeading(heading);
        this.gpsEvent.setAltitude(altitudeM);
        this.gpsEvent.setOdometerKM(odomKM);
        if (gpioInput >= 0L) { this.gpsEvent.setInputMask(gpioInput); }

                if (this.parseInsertRecord_Common(this.gpsEvent)) {
            return (ack != null)? (ack+"\n").getBytes() : null;
        } else {
            return (nak != null)? (nak+"\n").getBytes() : null;
        }

    }


    private byte[] parseInsertRecord_Device_1(String s)
    {

        Print.logInfo("Parsing: " + s);

                if (StringTools.isBlank(s)) {
            Print.logError("Packet string is blank/null");
            return null;
        } else
        if (!s.startsWith("$")) {
            Print.logError("Packet string does not start with '$'");
            return null;
        }

        String fld[] = StringTools.parseString(s.substring(1), ',');
        if ((fld == null) || (fld.length < 11)) {
            Print.logWarn("Invalid number of fields");
            return null;
        }
```

```
        String   eventCode  = fld[0];
        String   modemID    = fld[1].toLowerCase();
        long     fixtime    = Nmea0183.parseFixtime  (fld[10], fld[2], true);
        boolean  validGPS   = fld[3].equalsIgnoreCase("A");
        double   latitude   = validGPS? Nmea0183.ParseLatitude (fld[ 4], fld[5],
90.0) : 0.0;
        double   longitude  = validGPS? Nmea0183.ParseLongitude(fld[ 6], fld[6],
180.0) : 0.0;
        double    speedKnot  = validGPS? StringTools.parseDouble(fld[ 8], 0.0) :
0.0;
        double   speedKPH   = validGPS? (speedKnot * KILOMETERS_PER_KNOT) : 0.0;
        double    heading    = validGPS? StringTools.parseDouble(fld[ 9], 0.0) :
0.0;
        double   altitudeM  = 0.0;

        int      statusCode = StatusCodes.STATUS_LOCATION;
        if (eventCode.equalsIgnoreCase("POS")) {

            statusCode = StatusCodes.STATUS_LOCATION;
        } else
        if (eventCode.equalsIgnoreCase("IN1")) {
            statusCode = StatusCodes.STATUS_WAYMARK_1;
        } else
        if (eventCode.equalsIgnoreCase("IN2")) {
            statusCode = StatusCodes.STATUS_WAYMARK_2;
        } else
        if (eventCode.equalsIgnoreCase("IN3")) {
                    statusCode = StatusCodes.STATUS_WAYMARK_3;
        } else
        if (eventCode.equalsIgnoreCase("LPA")) {
                    statusCode = StatusCodes.STATUS_LOW_BATTERY;
        } else
        if (eventCode.equalsIgnoreCase("CPA")) {
                    statusCode = StatusCodes.STATUS_POWER_FAILURE;
        } else
        if (eventCode.equalsIgnoreCase("SPD")) {
                    statusCode = StatusCodes.STATUS_MOTION_EXCESS_SPEED;
        } else
        if (eventCode.equalsIgnoreCase("GOF")) {
                    statusCode = StatusCodes.STATUS_GEOFENCE_VIOLATION;
        }

        if (StringTools.isBlank(modemID)) {
            Print.logWarn("ModemID not specified!");
            return null;
        }

               this.gpsEvent = new GPSEvent(Main.getServerConfig(null),
            this.ipAddress, this.clientPort, modemID);
        Device device = this.gpsEvent.getDevice();
        if (device == null) {
            return null;
        }
        this.sessionID = this.createTcpSessionID(device);

        this.gpsEvent.setTimestamp(fixtime);
        this.gpsEvent.setStatusCode(statusCode);
        this.gpsEvent.setLatitude(latitude);
        this.gpsEvent.setLongitude(longitude);
        this.gpsEvent.setSpeedKPH(speedKPH);
        this.gpsEvent.setHeading(heading);
        this.gpsEvent.setAltitude(altitudeM);

        if (this.parseInsertRecord_Common(this.gpsEvent)) {
```

```
            return null;
        } else {
            return null;
        }

    }

    public static void configInit()
    {
        DCServerConfig dcsc     = Main.getServerConfig(null);
        if (dcsc == null) {
            Print.logWarn("DCServer not found: " + Main.getServerName());
            return;
        }

        DATA_FORMAT_OPTION              =  dcsc.getIntProperty(Main.ARG_FORMAT,
DATA_FORMAT_OPTION);

        MINIMUM_SPEED_KPH       = dcsc.getMinimumSpeedKPH(MINIMUM_SPEED_KPH);
        ESTIMATE_ODOMETER       = dcsc.getEstimateOdometer(ESTIMATE_ODOMETER);
        SIMEVENT_GEOZONES       = dcsc.getSimulateGeozones(SIMEVENT_GEOZONES);
        SIMEVENT_DIGITAL_INPUTS                                              =
dcsc.getSimulateDigitalInputs(SIMEVENT_DIGITAL_INPUTS) & 0xFFFFL;

    }


    private static int _usage()
    {
        String cn = StringTools.className(TrackClientPacketHandler.class);
        Print.sysPrintln("Test/Load Device Communication Server");
        Print.sysPrintln("Usage:");
        Print.sysPrintln("     $JAVA_HOME/bin/java  -classpath  <classpath>  %s
{options}", cn);
        Print.sysPrintln("Options:");
        Print.sysPrintln("  -insert=[true|false]     Insert parsed records into
EventData");
        Print.sysPrintln("  -format=[1|2]           Data format");
        Print.sysPrintln("  -debug                   Parse internal sample/debug
data (if any)");
        Print.sysPrintln("  -parseFile=<file>         Parse data from specified
file");
        return 1;
    }

        public static int _main(boolean fromMain)
    {

          INSERT_EVENT = RTConfig.getBoolean(Main.ARG_INSERT, DFT_INSERT_EVENT);
        if (!INSERT_EVENT) {
            Print.sysPrintln("Warning:  Data  will  NOT  be  inserted  into  the
database");
        }

        TrackClientPacketHandler tcph = new TrackClientPacketHandler();

                if (RTConfig.getBoolean(Main.ARG_DEBUG,false)) {
            String data[] = null;
            switch (DATA_FORMAT_OPTION) {
                case  1: data = new String[] {
                    "123456789012345,2006/09/05,07:47:26,35.3640,-
142.2958,27.0,224.8",
                }; break;
                case  2: data = new String[] {
```

```
"account/device/$GPRMC,025423.494,A,3709.0642,N,14207.8315,W,12.09,108.52,200505
,,*2E",

"/device/$GPRMC,025423.494,A,3709.0642,N,14207.8315,W,12.09,108.52,200505,,*2E",
                }; break;
                case  3: data = new String[] {
                    "2,123,1234567890,0,20101223,110819,1,2.1,39.1234,-
142.1234,33,227,1800",
                }; break;
                case  9: data = new String[] {
                    "mid=123456789012345 lat=39.12345 lon=-142.12345 kph=123.0"
                }; break;
                default:
                    Print.sysPrintln("Unrecognized     Data     Format:     %d",
DATA_FORMAT_OPTION);
                    return _usage();
            }
            for (int i = 0; i < data.length; i++) {
                tcph.getHandlePacket(data[i].getBytes());
            }
            return 0;
        }

        if (RTConfig.hasProperty(Main.ARG_PARSEFILE)) {

                    File parseFile = RTConfig.getFile(Main.ARG_PARSEFILE,null);
            if ((parseFile == null) || !parseFile.isFile()) {
                Print.sysPrintln("Data source file not specified, or does not
exist.");
                return _usage();
            }

                        FileInputStream fis = null;
            try {
                fis = new FileInputStream(parseFile);
            } catch (IOException ioe) {
                Print.logException("Error openning input file: " + parseFile,
ioe);
                return 2;
            }

            try {
                for (;;) {
                    String data = FileTools.readLine(fis);
                    if (!StringTools.isBlank(data)) {
                        tcph.getHandlePacket(data.getBytes());
                    }
                }
            } catch (EOFException eof) {
                Print.sysPrintln("");
                Print.sysPrintln("***** End-Of-File *****");
            } catch (IOException ioe) {
                Print.logException("Error reaading input file: " + parseFile,
ioe);
            } finally {
                try { fis.close(); } catch (Throwable th) {/* ignore */}
            }

            return 0;

        }

                return _usage();

    }
```

```
        public static void main(String argv[])
    {
        DBConfig.cmdLineInit(argv,false);
        TrackClientPacketHandler.configInit();
        System.exit(TrackClientPacketHandler._main(false));
    }

}
```

Б.2 Лістинг програми сервера (TrackServer) з використанням модульного мережного інтерфейсу на мові Java та JavaScript

```
import java.lang.*;
import java.util.*;
import java.io.*;
import java.net.*;
import java.sql.*;

public class TrackServer
{
    public static void configInit()
   {        DCServerConfig dcs = Main.getServerConfig(null);
        if (dcs != null) {
            TrackServer.setTcpIdleTimeout(   dcs.getTcpIdleTimeoutMS(
Constants.TIMEOUT_TCP_IDLE   ));
            TrackServer.setTcpPacketTimeout( dcs.getTcpPacketTimeoutMS(
Constants.TIMEOUT_TCP_PACKET ));

TrackServer.setTcpSessionTimeout(dcs.getTcpSessionTimeoutMS(Constants.TIMEOUT_TC
P_SESSION));
TrackServer.setUdpIdleTimeout(   dcs.getUdpIdleTimeoutMS(
Constants.TIMEOUT_UDP_IDLE   ));
TrackServer.setUdpPacketTimeout( dcs.getUdpPacketTimeoutMS(
Constants.TIMEOUT_UDP_PACKET ));
TrackServer.setUdpSessionTimeout(dcs.getUdpSessionTimeoutMS(Constants.TIMEOUT_UD
P_SESSION));
        } else {
            Print.logWarn("DCServer not found: " + Main.getServerName());
      }
   }

    public static TrackServer startTrackServer(int tcpPorts[], int udpPorts[],
int commandPort)
        throws Throwable
 {        if (trackServerInstance == null) {
            trackServerInstance = new TrackServer(tcpPorts, udpPorts,
commandPort);
 }
        return trackServerInstance;
   }
   public static TrackServer getInstance()
    {
        return trackServerInstance;
    }
```

```
        private static long tcpTimeout_idle = Constants.TIMEOUT_TCP_IDLE;
    public static void setTcpIdleTimeout(long timeout)
    {
        TrackServer.tcpTimeout_idle = timeout;
    }
    public static long getTcpIdleTimeout()
    {
        return TrackServer.tcpTimeout_idle;
    }
    private static long tcpTimeout_packet = Constants.TIMEOUT_TCP_PACKET;
    public static void setTcpPacketTimeout(long timeout)
    {
        TrackServer.tcpTimeout_packet = timeout;
    }
    public static long getTcpPacketTimeout()
    {
        return TrackServer.tcpTimeout_packet;
    }
        private static long tcpTimeout_session  = Constants.TIMEOUT_TCP_SESSION;
public static void setTcpSessionTimeout(long timeout)
    {
        TrackServer.tcpTimeout_session = timeout;
    }
    public static long getTcpSessionTimeout()
    {
        return TrackServer.tcpTimeout_session;
    }
    private static long udpTimeout_idle = Constants.TIMEOUT_UDP_IDLE;
    public static void setUdpIdleTimeout(long timeout)
    {
        TrackServer.udpTimeout_idle = timeout;
    }
    public static long getUdpIdleTimeout()
    {
        return TrackServer.udpTimeout_idle;
    }
private static long udpTimeout_packet = Constants.TIMEOUT_UDP_PACKET;
    public static void setUdpPacketTimeout(long timeout)
    {
        TrackServer.udpTimeout_packet = timeout;
    }
    public static long getUdpPacketTimeout()
    {
        return TrackServer.udpTimeout_packet;
    }
        private static long udpTimeout_session = Constants.TIMEOUT_UDP_SESSION;
    public static void setUdpSessionTimeout(long timeout)
    {
        TrackServer.udpTimeout_session = timeout;
    }
    public static long getUdpSessionTimeout()
    {
        return TrackServer.udpTimeout_session;
    }
    private ServerSocketThread                      cmdThread = null;
    private TrackServer(int tcpPorts[], int udpPorts[], int commandPort)
throws Throwable    {
        int listeners = 0;
                if (!ListTools.isEmpty(tcpPorts)) {
            this.tcpThread = new OrderedMap<Integer,ServerSocketThread>();
            for (int i = 0; i < tcpPorts.length; i++) {
                int port = tcpPorts[i];
                if (ServerSocketThread.isValidPort(port)) {
                 try {
                        ServerSocketThread sst = this._startTCP(port);
```

```
                        this.tcpThread.put(new Integer(port), sst);
                        listeners++;
                    } catch (java.net.BindException be) {
                        Print.logError("TCP: Error binding to port: %d", port);
}
                } else {
                    throw new Exception("TCP: Invalid port number: " + port);
              }
            }
       }if (!ListTools.isEmpty(udpPorts)) {
            this.udpThread = new OrderedMap<Integer,ServerSocketThread>();
this.udpSocket = new OrderedMap<Integer,DatagramSocket>();
            for (int i = 0; i < udpPorts.length; i++) {
                int port = udpPorts[i];
                if (ServerSocketThread.isValidPort(port)) {
                    try {
                        ServerSocketThread sst = this._startUDP(port);
                        this.udpThread.put(new Integer(port), sst);
                        this.udpSocket.put(new Integer(port),
sst.getDatagramSocket());
                        listeners++
                    } catch (java.net.BindException be) {
Print.logError("UDP: Error binding to port: %d", port);
                    }
                } else {
                    throw new Exception("UDP: Invalid port number: " + port);
}
}
        }
                if (listeners <= 0) {
            Print.logWarn("No active device communication listeners!");
      }               if (commandPort > 0) {
            if (ServerSocketThread.isValidPort(commandPort)) {
                try {
                    this._startCommand(commandPort);
                } catch (java.net.BindException be) {
                    Print.logError("Command: Error binding to port: %d",
commandPort);
                }
            } else {
                throw new Exception("Command: Invalid port number: " +
commandPort);
            }
        } else {
                        Print.logWarn("Ignoring CommandPort listener");
        }
    }           private ServerSocketThread _startTCP(int port)
        throws Throwable
  {        ServerSocketThread sst = null;
        try {
            sst = new ServerSocketThread(port);
        } catch (Throwable t) {
            Print.logException("ServerSocket error", t);
            throw t;
       }
```

```
        sst.setTextPackets(Constants.ASCII_PACKETS);
        sst.setBackspaceChar(null);
sst.setLineTerminatorChar(Constants.ASCII_LINE_TERMINATOR);
        sst.setMaximumPacketLength(Constants.MAX_PACKET_LENGTH);
        sst.setMinimumPacketLength(Constants.MIN_PACKET_LENGTH);
        sst.setIdleTimeout(TrackServer.getTcpIdleTimeout());
sst.setPacketTimeout(TrackServer.getTcpPacketTimeout());
sst.setSessionTimeout(TrackServer.getTcpSessionTimeout());
sst.setLingerTimeoutSec(5);
        sst.setTerminateOnTimeout(Constants.TERMINATE_ON_TIMEOUT);
        sst.setClientPacketHandlerClass(TrackClientPacketHandler.class);
        DCServerConfig.startServerSocketThread(sst,"Event");
        return sst;

    }

    public ServerSocketThread getServerSocketThread_tcp(int port)
    {
        if (ListTools.isEmpty(this.tcpThread)) {
            return null;
        } else
        if (port <= 0) {
            return this.tcpThread.getFirstValue();
        } else {
            ServerSocketThread sst = this.tcpThread.get(new Integer(port));
            if (sst != null) {
                return sst;
            } else {
                Print.logWarn("TCP 'Listen' port["+port+"] not found, returning
first ServerSocketThread");
                return this.tcpThread.getFirstValue();
            }
        }
    }

    private ServerSocketThread _startUDP(int port)
        throws Throwable
    {
        ServerSocketThread sst = null;

        /* create server socket */
        try {
            sst                                 =                                   new
ServerSocketThread(ServerSocketThread.createDatagramSocket(port));
        } catch (Throwable t) {
            Print.logException("ServerSocket error", t);
            throw t;
        }

        sst.setTextPackets(Constants.ASCII_PACKETS);
        sst.setBackspaceChar(null);
        sst.setLineTerminatorChar(Constants.ASCII_LINE_TERMINATOR);
        sst.setMaximumPacketLength(Constants.MAX_PACKET_LENGTH);
        sst.setMinimumPacketLength(Constants.MIN_PACKET_LENGTH);
        sst.setIdleTimeout(TrackServer.getUdpIdleTimeout());
        sst.setPacketTimeout(TrackServer.getUdpPacketTimeout());
        sst.setSessionTimeout(TrackServer.getUdpSessionTimeout());
        sst.setTerminateOnTimeout(Constants.TERMINATE_ON_TIMEOUT);
        sst.setClientPacketHandlerClass(TrackClientPacketHandler.class);

        DCServerConfig.startServerSocketThread(sst,"Event");
        return sst;

    }
```

```
    public DatagramSocket getUdpDatagramSocket(int port)
    {
        if (ListTools.isEmpty(this.udpSocket)) {
            return null;
        } else
        if (port <= 0) {
            return this.udpSocket.getFirstValue();
        } else {
            DatagramSocket ds = this.udpSocket.get(new Integer(port));
            if (ds != null) {
                return ds;
            } else {
                Print.logWarn("'Listen'  port["+port+"]  not  found,  returning
first DatagramSocket");
                return this.udpSocket.getFirstValue();
            }
        }
    }

        private void _startCommand(int port)
        throws Throwable
    {
        ServerSocketThread sst = null;

                Class cmdPktClass = null;
        try {
            cmdPktClass                                                    =
Class.forName("org.opengts.servers.template.TemplateCommandHandler");
        } catch (Throwable th) {
            return;
        }

        try {
            sst = new ServerSocketThread(port);
        } catch (Throwable t) {
            Print.logException("ServerSocket error", t);
            throw t;
        }

        sst.setTextPackets(true);
        sst.setBackspaceChar(null);
        sst.setLineTerminatorChar(new int[] { '\r', '\n' });
        sst.setIgnoreChar(null);
        sst.setMaximumPacketLength(1200);
sst.setMinimumPacketLength(1);
        sst.setIdleTimeout(1000L);
        sst.setPacketTimeout(1000L);
        sst.setSessionTimeout(10000L);
sst.setLingerTimeoutSec(5);
        sst.setTerminateOnTimeout(true);
        sst.setClientPacketHandlerClass(cmdPktClass);

        Print.logInfo("Starting Command listener thread on port " + port + "
[timeout=" + sst.getSessionTimeout() + "ms] ...");
        sst.start();
        this.cmdThread = sst;

    }


}
```

Б.3. Лістинги допоміжних модулів

```
public class Constants

{


    public static final String  TITLE_NAME             = "Template Example";
    public static final String  VERSION                = "0.2.7";
    public static final String  COPYRIGHT              = Version.COPYRIGHT;
    public  static  final  String    DEVICE_CODE                              =
DCServerFactory.TEMPLATE_NAME;
    public static final boolean ASCII_PACKETS          = false;
    public static final int     ASCII_LINE_TERMINATOR[] = new int[] { '\r', '\n'
};

        public static final int     MAX_PACKET_LENGTH       = 600;

        public static final boolean TERMINATE_ON_TIMEOUT    = true;

        public static final long    TIMEOUT_TCP_IDLE        = 10000L;
    public static final long    TIMEOUT_TCP_PACKET      = 4000L;
    public static final long    TIMEOUT_UDP_IDLE        = 5000L;
    public static final long    TIMEOUT_UDP_PACKET      = 4000L;
    public static final long    TIMEOUT_UDP_SESSION     = 60000L;
    public static void main(String argv[])
    {
        Print.sysPrintln(VERSION);
    }

}



public class Main


{
    public   static  final  String  ARG_PARSEFILE[]  =  new  String[]  {  "parse"   ,
"parseFile" };
    public  static final String ARG_HELP[]      = new String[] { "help"   , "h"
};
    public   static  final  String  ARG_CMD_PORT[]   =  new  String[]  {  "command",
"cmd" };
    public  static final String ARG_START[]     = new String[] { "start"  };
    public  static final String ARG_DEBUG[]     = new String[] { "debug"  };
    public   static  final  String  ARG_FORMAT[]      =  new  String[]  {  "format"  ,
"parseFormat" };
    public  static final String ARG_INSERT[]    = new String[] { "insert" };
        public static String getServerName()
    {
        return Constants.DEVICE_CODE;
    }
        public static DCServerConfig getServerConfig(Device dev)
    {
        return DCServerFactory.getServerConfig(Main.getServerName());
    }
        public static int[] getTcpPorts()
    {
        DCServerConfig dcs = Main.getServerConfig(null);
        if (dcs != null) {
            return dcs.getTcpPorts();
        } else {
```

```
            Print.logError("DCServerConfig   not   found   for   server:   "   +
getServerName());
           return null;
        }
    }

 public static int[] getUdpPorts()

 {
        DCServerConfig dcs = Main.getServerConfig(null);
        if (dcs != null) {
            return dcs.getUdpPorts();
        } else {
            Print.logError("DCServerConfig   not   found   for   server:   "   +
getServerName());
            return null;
        }
    }
        public static int getCommandDispatcherPort()
    {
        DCServerConfig dcs = Main.getServerConfig(null);
        if (dcs != null) {
            return dcs.getCommandDispatcherPort();
        } else {
            return RTConfig.getInt(ARG_CMD_PORT,0);
        }
    }
        private static void usage(String msg)
    {
        String tcp = StringTools.join(getTcpPorts(),",");
        String udp = StringTools.join(getUdpPorts(),",");
        String cmd = String.valueOf(getCommandDispatcherPort());
                if (msg != null) {
            Print.logInfo(msg);
        }
                String className = Main.class.getName();
        Print.logInfo("");
        Print.logInfo("Usage:");
        Print.logInfo("  java ... " + className + " -h[elp]");
        Print.logInfo(" or");
        Print.logInfo("  java ... " + className + " -parseFile=<filePath>");
        Print.logInfo(" or");
        Print.logInfo("   java  ...  "  +  className  +  "  [-port=<port>[,<port>]]  -
start");
        Print.logInfo("Options:");
        Print.logInfo("  -help                This help");
        Print.logInfo("  [-port=<p>[,<p>]]   Server TCP/UDP port(s) to listen");
        Print.logInfo("   [-tcp=<p>[,<p>]]      Server  TCP  port(s)  to  listen  on
[dft="+tcp+"]");
        Print.logInfo("   [-udp=<p>[,<p>]]      Server  UDP  port(s)  to  listen  on
[dft="+udp+"]");
        Print.logInfo("    [-command=<p>]           Command  port  to  listen  on
[dft="+cmd+"]");
        Print.logInfo("          [-dcs=<serverId>]               DCServer     ID
[dft="+Constants.DEVICE_CODE+"]");
        Print.logInfo("  [-format=<parser#>] Parser Format #");
        Print.logInfo("   -start                   Start  server  on  the  specified
port.");
        Print.logInfo("    -parseFile=<file>     File  from  which  data  will  be
parsed.");
        Print.logInfo("");
        System.exit(1);
    }
        public static void main(String argv[])
    {
```

```
        DBConfig.cmdLineInit(argv,false);

        DEVICE_CODE = RTConfig.getString(ARG_DEVCODE, Constants.DEVICE_CODE);
        if (StringTools.isBlank(DEVICE_CODE)) {
            Print.logFatal("Invalid device-code specified");
            Main.usage("");
            System.exit(1);
        }
               TrackClientPacketHandler.configInit();
        TrackServer.configInit();
        String SEP =


"------------------------------------------------------------------------";


   Print.logInfo(SEP);
        Print.logInfo(Constants.TITLE_NAME    +    "    Server    Version    "    +
Constants.VERSION);
        Print.logInfo("DeviceCode           : " + Constants.DEVICE_CODE);
        Print.logInfo("ParseFormat          : " +

TrackClientPacketHandler.DATA_FORMAT_OPTION);
        Print.logInfo("MinimumSpeed         : " +

TrackClientPacketHandler.MINIMUM_SPEED_KPH);
        Print.logInfo("EstimateOdom         : " +

TrackClientPacketHandler.ESTIMATE_ODOMETER);
        Print.logInfo("TCP Idle Timeout     : " +

TrackServer.getTcpIdleTimeout()    + " ms");
        Print.logInfo("TCP Packet Timeout   : " +

TrackServer.getTcpPacketTimeout()  + " ms");
        Print.logInfo("TCP Session Timeout  : " +

TrackServer.getTcpSessionTimeout() + " ms");
        Print.logInfo("UDP    Idle    Timeout                          :    "    +
TrackServer.getUdpIdleTimeout()    + " ms");
        Print.logInfo("UDP Packet Timeout   : " +

TrackServer.getUdpPacketTimeout()  + " ms");
        Print.logInfo("UDP Session Timeout  : " +

TrackServer.getUdpSessionTimeout() + " ms");
        Print.logInfo(Constants.COPYRIGHT);
        Print.logInfo(SEP);
               if (RTConfig.getBoolean(ARG_HELP,false)) {
            Main.usage("Help ...");
            System.exit(0);
        }

if (!DBAdmin.verifyTablesExist()) {
            Print.logFatal("MySQL   database   has   not   yet   been   properly
initialized");
            System.exit(1);
        }
        if (RTConfig.hasProperty(ARG_PARSEFILE)) {
            Print.sysPrintln("Attempting to parse data from file: " +

RTConfig.getString(ARG_PARSEFILE));
            RTConfig.setString("parseFile", RTConfig.getString(ARG_PARSEFILE));
            int exit = TrackClientPacketHandler._main(true);
            System.exit(exit);
```

```
        }
        if (RTConfig.getBoolean(ARG_START,false)) {
                            try {
                int tcpPorts[]  = getTcpPorts();
                int udpPorts[]  = getUdpPorts();
                int commandPort = getCommandDispatcherPort();
                TrackServer.startTrackServer(tcpPorts, udpPorts, commandPort);
            } catch (Throwable t) {
                Print.logError("Error: " + t);
            }

                while (true) {
                try { Thread.sleep(60L * 60L * 1000L); } catch (Throwable t) {}


   Main.usage("Missing '-start' ...");
        System.exit(1);
    }
}
```

“ЗАТВЕРДЖУЮ”
Перший проректор Львівського
державного університету безпеки
життєдіяльності
_____ М.С.Коваль
27.02 2016 р.

АКТ

про впровадження результатів дисертаційних досліджень
Пастернак Ірини Ігорівни
на тему “ Підвищення ефективності мережних інтерфейсів навігаційних сервісів кіберфізичних систем ”

Комісія у складі: голови – начальника відділу організації науково-дослідної діяльності, к.т.н., доцента Рудика Ю.І., членів – заступника начальника інституту цивільного захисту, к.ф.-м.н., доцента Меньшикової О.В. та доцента кафедри управління проектами, інформаційних технологій та телекомунікацій, к.т.н. Мальця І.О. встановила, що наукові положення дисертаційних досліджень Пастернак І.І. були впроваджені при розробленні прогнозно-аналітичної системи надзвичайних ситуацій.

До основних результатів дисертаційних досліджень Пастернак І.І., що знайшли своє впровадження, слід віднести вдосконалені методи створення мережних інтерфейсів для зв’язку інтерфейсу користувача та серверів аналізу інформації про надзвичайні ситуації, які дозволили уніфікувати їх та зменшити об’єм коду при програмній реалізації.

Розроблені наукові положення дисертаційних досліджень Пастернак І.І. дозволили підвищити технічну ефективність та надійність мережних інтерфейсів навігаційних сервісів кіберфізичних систем.

Начальник відділу організації
науково-дослідної діяльності,
к.т.н., доцент Ю.І.Рудик

Заступник начальника
інституту цивільного захисту,
к.ф.-м.н., доцент О.В.Меньшикова

Доцент кафедри управління проектами,
інформаційних технологій та телекомунікацій,
к.т.н. І.О.Малець