Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель вибору pattern-архітектури
для розробки систем IoT

(тема)

Виконав:
студент II курсу, групи СПм-19-1
Філіпчик А.А.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

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

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Токарєв В.В.
(посада, прізвище, ініціали)

Допускається до захисту

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

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Філіпчику Антону Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Модель вибору pattern-архітектури для розробки систем IoT

затверджена наказом по університету від " 30 " жовтня 2020 р. № 1486 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020 р.

3. Вхідні дані до роботи Метод Эрлана
Система резолюцій DO – Handle system
Можливість роботи з інтерфейсом Wi-Fi.
Операційна система – Windows 10.
Технічне забезпечення: IBM - сумісний комп'ютер, AVR – мікроконтролер – 328P .
Представлення вихідних даних: згідно нормативних документів.

4. Перелік питань, що потрібно опрацювати в роботі
1) огляд літератури за темою роботи
2) наліз предметної області
3) вибір та обґрунтування методики дослідження
4) проведення експериментальних досліджень
5) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Плакати – № 18- арк. ф. А4

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою роботи	03.11.20 - 20.11.20	
2	Вибір та обґрунтування методики дослідження	21.11.20 - 24.11.20	
3	Вибір інструментальних засобів	25.11.20 - 01.12.20	
4	Проведення експериментів	02.12.20 - 05.12.20	
5	Оформлення матеріалів атестаційної роботи	06.12.20 - 11.12.20	
6	Подання атестаційної роботи керівникові та її	14.12.20 - 15.12.20	
	попередній захист		
7	Подання атестаційної роботи на рецензування	16.12.20	

Дата видачі завдання 02 листопада 2020 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Токарєв В.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 76 с., 13 рис., 1 табл., 1 дод., 15 джерел.

БЕЗДРОТОВА МЕРЕЖА, ІНТЕРНЕТ, ПРОТОКОЛ, СЕРВЕР, IoT IIoT, GPS, WI-FI, WLAN.

Метою атестаційної роботи є розробка алгоритму вибора pattern-архітектури для розробки систем IoT.

Принципи фізичної, інформаційної безпеки та надійності призводять до появи кількісних оцінок якості ПЗ і можуть бути обчислені в залежності від проекту. Дані принципи є параметрами, що досягаються, структурно-параметричного синтезу системи і, як передбачається в цій атестаційній роботі, залежать від параметрів якості програмної архітектури, а значить, основа для їх реалізації може бути закладена на початковому етапі проєкту. З урахуванням вищесказаного, узагальнюючи принципи і параметри систем IIoT на більш загальний клас систем IoT, виникає питання про знаходження загальної залежності досягнутих параметрів якості систем IoT від використовуваної програмної архітектури.

ABSTRACT

Master’s thesis: 76 pages, 13 figures, 1 tables, 1 appendices, 15 sources.

INTERNET, IIoT, GPS, PROTOCOL, SERVER, WI-FI, WIRELESS NETWORK, WLAN.

The purpose of the certification work is to develop an algorithm for selecting a pattern - architecture for the development of IoT systems.

The principles of physical, information security and reliability lead to the emergence of quantitative assessments of software quality and can be calculated depending on the project. These principles are achievable parameters of the structural-parametric synthesis of the system and, as it is assumed in this certification work, depend on the quality parameters of the software architecture, which means that the basis for their implementation can be laid at the initial stage of the project. Taking into account the foregoing, generalizing the principles and parameters of IoT systems to a more general class of IoT systems, the question arises of finding the general dependence of the achieved quality parameters of IoT systems on the software architecture used.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

МСЕ – Міжнародний союз електрозв'язку

ОПР – особа, яка приймає рішення

ПЗ – програмне забеспечення

СППР – система підтримки прийняття рішень

DARPA – агентство перспективних оборонних дослідницьких проектів (англ., Defense Advanced Research Projects Agency)

ERM – модель відносин сущності (англ., Entity-Relationship Model)

JMS – служба повідомлень Java (англ., Java Message Service)

IoT – інтернет речей (англ., Internet of Things)

IIoT – промисловий інтернет речей (англ., Industrial Internet of Things)

ВСТУП

В останнє десятиліття швидкий розвиток Internet-технологій, алгоритмів мережевої взаємодії і підвищення доступності обчислювальних пристроїв привели до появи і швидкого розвитку технології, яка отримала назву IoT. На міжнародному рівні дана концепція набуває сформовані риси, що можна простежити по появі чіткої класифікації типів подібних систем, стандартів взаємодії пристроїв, а також виділенню фундаментальних характеристик технології відповідно до рекомендацій Міжнародного союзу електрозв'язку (МСЕ).

Слідом за появою IoT стала можливою побудова високоінтелектуальних виробничих систем і додатків, в основі управління яких лежить зв'язок між усіма основними пристроями виробничої системи, а також центрами управління, в яких проводиться моніторинг та розрахунок операцій. Подібні системи отримали в літературі назву промислового інтернету IIoT. З точки зору конкурентоспроможності підприємств, які впроваджують подібну концепцію, кількість подібних мереж має тенденцію до зростання з огляду на те, що позитивний вплив систем IoT на різні показники ефективності було неодноразово доведено успішними проектами (можна розглянути системи Smart house і, наприклад, значне підвищення рівня безпеки подібного житла). У той же час, беручи до уваги, що системи IoT можна з упевненістю віднести до складних і вимоги до певних параметрів таких комплексів сильно залежать від галузі, досягнення необхідних атрибутів за допомогою тільки мережевих алгоритмів взаємодії не представляється можливим. Особливо актуальна проблема вибору програмної архітектури, так як в кінцевому підсумку помилка на керуючому рівні зводить нанівець ефективність роботи комунікаційних і мережних алгоритмів. При цьому, поряд з широким розвитком апаратних і мережевих стандартів технології, в сфері стандартів розробки програмної частини

спостерігається пробіл і для ряду класів подібних систем питання оптимальної побудови програмної частини платформи залишається відкритим. З огляду на поширення технології і величини подібних систем, помилки в побудові програмної архітектури ведуть до значного подорожчання і збільшення термінів проєкту, що підтверджується дослідженнями. Частково дана прогалина була заповнена галузевими стандартами. Наприклад, при розробці систем IIoT, рекомендують дотримуватися офіційного документа, розробленого для підтримки побудови подібних комплексів – стандарту на розробку типової архітектури. В даному стандарті, що розробляється Консорціумом щодо впровадження IIoT, показані основні моделі побудови, елементи, зв'язки та концептуальні обмеження, властиві зазначеним системам.

1 АНАЛІЗ СПЕЦИФІКИ СИСТЕМ IoT І ПРОГРАМНОЇ АРХІТЕКТУРИ

На початку XXI століття, поряд з широким розвитком мережі Internet, стали з'являтися так звані всюдипроникливні сенсорні мережі. З'явилася можливість взаємодії пристроїв між собою без участі людини, що в свою чергу показало необхідність розвитку нової концепції пристроїв, які комутують між собою. IoT як термін був вперше згаданий в 1999 році Кевіном Ештоном, засновником MIT Auto-ID центру.

В останні роки технології Інтернету речей набирають популярність у багатьох сферах науки і техніки. Тут слід зазначити, що ідеї IoT багато в чому перегукуються з ідеєю многоагентних систем, що самоорганізуються. Деякі паралелі многоагентних систем з системами IoT також простежуються, що дозволяє говорити про певну зрілість появи технології, як логічного прикладного застосування теорії агентних систем. Однак, поряд з цим, дане визначення до цих пір не несе ясного сенсу для користувачів технології. У літературі і на web-ресурсах в більшості випадків використовують дві основні характеристики.

Перша – IoT є мережею, що має на увазі використання терміна тільки в одиничній формі (як і в разі Internet, IoT тільки один).

Друга стосується так званих «речей», що відносяться до даного поняття. Це будь-які фізичні об'єкти нашого світу. Дані об'єкти є частиною самої технології, з'єднані в мережу і мають унікальні ідентифікатори. Стан кожного об'єкта можна відстежити, а в деяких випадках і керувати ним. Дані пристрої мають зв'язок між собою і можуть працювати спільно, тим самим отримуючи певну перевагу від кооперації. Якраз в цей момент спостерігається очевидна кореляція з теорією агентних систем, яка говорить про синергетичний ефект від кооперації агентів. Однак, проводячи таку аналогію, неможливо не відзначити певну ступінь спрощення технології в

разі IoT. Переваги від кооперації можуть в широкому сенсі зачіпати багато сфер діяльності (так як широта поширення систем IoT зараз досить велика). У число їх входять як переваги в досягненні нової функціональності (чого не було раніше через нестачу технології) – наприклад, системи оперативного моніторингу стану продуктів в логістиці, так і поліпшення якості та здешевлення існуючих процесів (наприклад, збільшення якості виробничих процесів, пов'язане з превентивним обслуговуванням устаткування).

Таким чином, новизна технології безпосередньо пов'язана з її визначенням – «речі» в даному контексті належать до щоденних фізичних об'єктів, які до цього не мали зв'язок один з одним і тепер створюють новий тип взаємодії і даних, а Internet в даному контексті відноситься до комунікації між цими пристроями. Стандарт Міжнародного союзу електрозв'язку визначає речі в концепції IoT як «Об'єкти фізичного світу (фізичні речі) або інформаційного світу (віртуальні речі), які можна ідентифікувати і інтегрувати в мережі зв'язку». У той же час, офіційне визначення IoT, наведене в Рекомендації MCE-T Y.2060, говорить, що «IoT - глобальна інфраструктура інформаційного суспільства, що забезпечує передові послуги за рахунок організації зв'язку між речами (фізичними або віртуальними) на основі сумісних інформаційних і комунікаційних технологій, що існують і розвиваються».

«Річчю» в даному контексті називається як фізичний об'єкт, так і об'єкт віртуального (інформаційного) світу (наприклад, програма), які можуть бути ідентифіковані та об'єднані через комунікаційні мережі. Схема взаємодії фізичних і віртуальних речей показана на рисунку 1.1.

Найбільшу популярність системи IoT придбали в 2010-2017 роках, із здешевленням кінцевих пристроїв, появою технологій для безпечної і швидкої передачі даних, а також з розвитком Internet і протоколів взаємодії. Дані умови сприятливо позначилися на появі великої кількості відносно недорогих і простих кінцевих пристроїв, які в короткі терміни можна було зібрати в готове рішення.

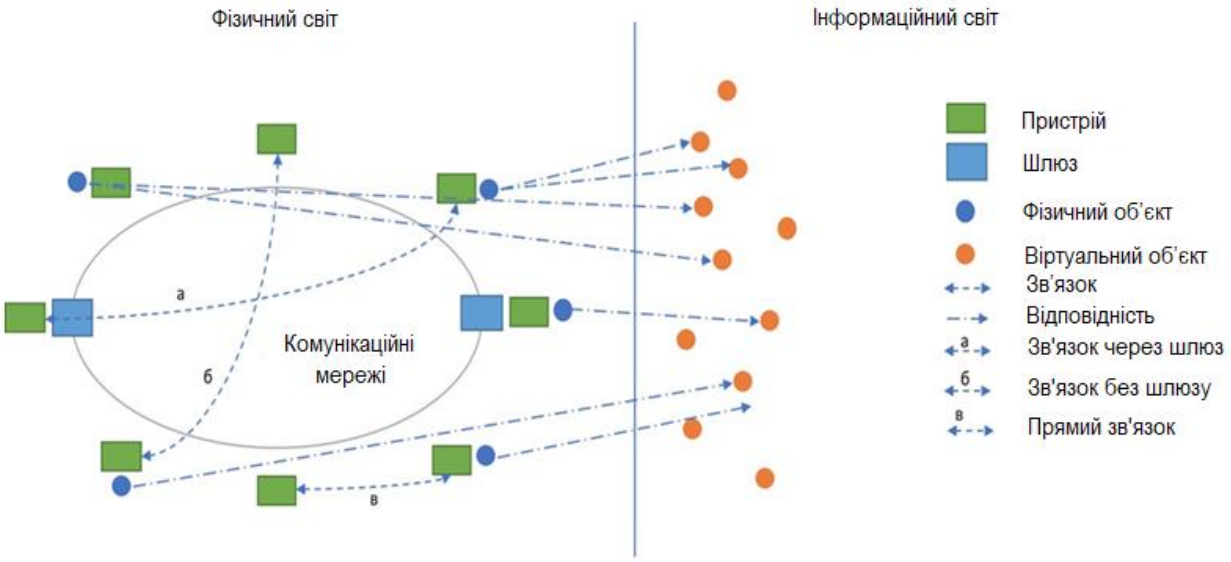


Рисунок 1.1 – Взаємодія фізичних і віртуальних речей

При цьому горизонт планування до 2025 року провідних фахівців телекомунікацій оцінюється в 50 мільярдів речей, підключених між собою в мережу. Питання стандартизації технології або окремих її складових розглядаються низкою міжнародних організацій, альянсів виробників пристроїв і неурядових асоціацій. В цілому, на даний момент для систем IoT побудовані загальні концептуальні та архітектурні рекомендації. Міжнародний союз електрозв'язку об'єднує під собою три глобальні ініціативи, що мають на увазі послідовність робіт, які виконуються паралельно і служать спільній меті. У число даних ініціатив входять:

- стандартизація IoT;
- стандартизація мереж наступних поколінь;
- системи телебачення на основі Internet.

В рамках ініціативи IoT-GSI була приведена еталонна модель IoT, що включає горизонтальні рівні взаємодії пристроїв (рисунок 1.2). У моделі розрізняються чотири горизонтальних рівня, починаючи з рівня пристроїв, що включає як кінцеві пристрої, так і шлюзи. Мережевий рівень включає можливості передачі даних, управління мобільністю, а також забезпечення функцій аутентифікації і пов'язаності даних. На рівні підтримки додатків

здійснюється забезпечення зберігання інформації, а також обробка даних. Самий верхній рівень (рівень додатків) розглядає різні кінцеві додатки Інтернету речей, які детально не описані в Стандарті. Разом з горизонтальними рівнями, вказані два вертикальних рівня: управління і безпеки, що охоплюють всі горизонтальні шари.

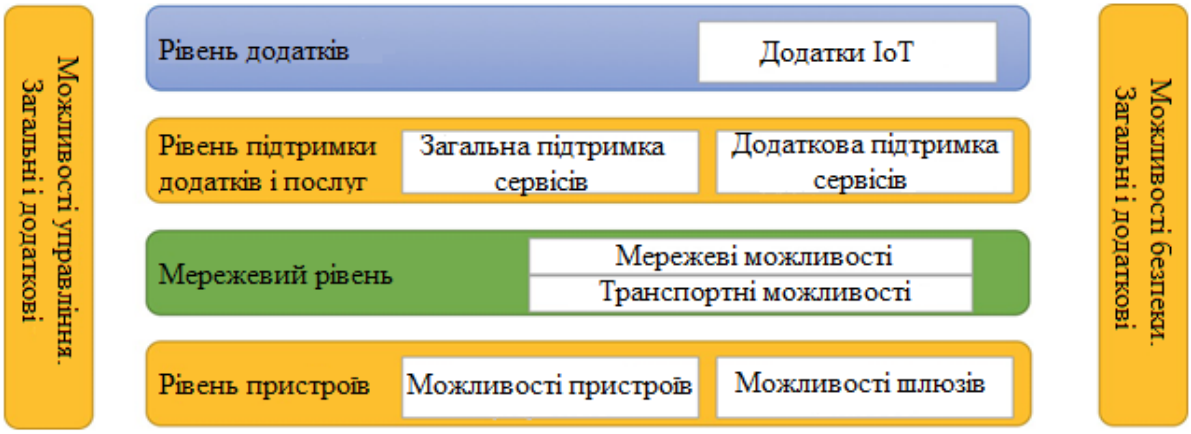


Рисунок 1.2 – Еталонна модель IoT згідно МСЕ-Т Y.2060

Поряд з МСЕ, досить великий вплив має Європейський інтеграційний проект IoT-A12, в який входять різні компанії і метою якого є розробка архітектурної моделі IoT. Функціональна модель IoT-A відрізняється від моделі МСЕ (рисунок 1.3) і складається з семи горизонтальних рівнів і двох вертикальних. З точки зору практичного розвитку систем IoT, поява на ринку великої кількості продуктів, пов'язаних з даною технологією, дещо випереджає зрілість стандартів. На даний момент присутні професійні системи з досить складною і розгалуженою логікою, до появи яких причетні вже цілі компанії і міжнародні комітети (наприклад, консорціум промислового Інтернету речей). В даному випадку подібні системи оснащені специфічною архітектурою і логікою роботи, що дозволяє їм одночасно отримувати дані і управляти багатьма пристроями, зберігаючи при цьому можливість взаємодії в реальному часі.

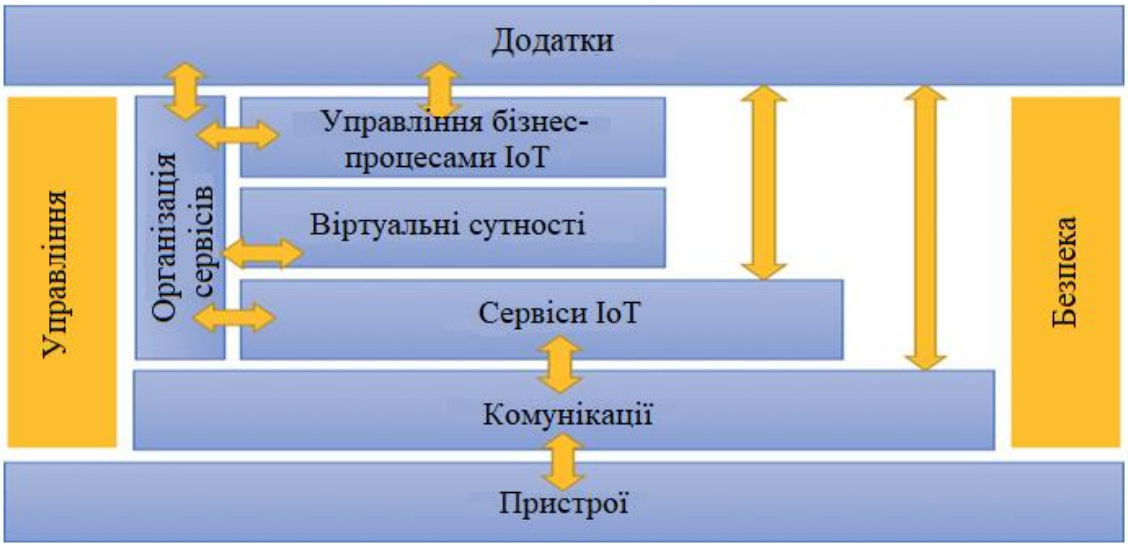


Рисунок 1.3 – Функціональна модель IoT-A

Усередині систем IoT існують великі відмінності в побудові і застосуванні, що йдуть від їх функціональних і структурних вимог, застосовності і надійності. Наприклад, існують рішення, призначені для логістики, інтелектуального обслуговування, Industrial 4.0, а також для Smart медицини. Зрозуміло, наприклад, допустимий параметр масштабованості у даних типів систем буде мати різне значення: для систем Smart медицини (де кількість пристроїв умовно обмежена вимірами на тілі людини) він нижчий, ніж для систем попереднього обслуговування, де безліч пристроїв можуть бути розподілені по всьому світу і кількість цих пристроїв безперервно зростає.

Поряд з впровадженням функціональних і архітектурних стандартів, які зачіпають правила побудови загальної архітектури IoT, з'являються стандарти, що регламентують побудову окремих видів подібних систем. Наприклад, при побудові систем Industrial 4.0 зважаючи на складність і неоднозначність функціональної області, існують окремі стандарти, що регламентують побудову таких систем. У 2013 році консорціумом індустріального IoT для систем IoT, що відносяться до промислового IoT був розроблений документ під назвою «Industrial Internet Reference Architecture» –

стандарт, який регламентує рівні побудови, складність, технічні деталі і особливості впровадження подібних систем на виробництві. Зазначений документ брали до уваги більшість західних компаній при розробці систем IIoT, що показало високу придатність методології на практиці. Безліч наукових питань і відкритих можливостей для досліджень в області Інтернету речей включають питання інформаційної безпеки, ефективності пристроїв, а також питання розробки ефективних протоколів взаємодії пристроїв і обладнання.

Окремим рядком можна виділити проблему розробки оптимальних програмних архітектур для різних типів систем IoT. Незважаючи на велику кількість мережевих і комунікаційних протоколів і стандартів, в даній області все ще спостерігається певний пробіл, заповнити який певною мірою покликане дане дослідження.

На даний момент створено безліч класифікацій систем IoT, багато з яких прийшли з промисловості і відображають функціональні та структурні характеристики рішень. Однією з найбільш популярних практичних класифікацій систем IoT є класифікація Postscapes, яка ділить всі безліч рішень на вісім різних категорій:

- розумний будинок;
- розумний транспорт;
- розумна охорона здоров'я;
- розумні продажі;
- розумний транспорт;
- розумний будинок;
- розумне виробництво;
- розумне місто.

Класифікація Postscapes практично повністю повторює класифікацію, розроблену в ході проекту IoT-A. Основними функціями Платформ IoT, в число виробників яких входять великі промислові холдінги, такі як «National

Instruments», «SAP», «Siemens», «General Electric», «PTC», є управління підключеними пристроями, підтримка протоколів взаємодії пристроїв і обладнання, а також функції базової аналітики і надання користувальницьких інтерфейсів з управління / моніторингу. Окремо можна виділити рішення візуалізації, які розробляються спеціально для специфічних систем IoT, а також рішення, що надають розгорнуті аналітичні функції для збору і маніпуляції даними від пристроїв.

Класифікація систем IoT, використовувана в роботі, була проведена на підставі 20 відкритих проектів, при цьому не враховувалися відомості про архітектуру та якість зазначених проектів. Відповідно до цього були виділені класифікаційні таксони, на підставі яких визначався тип системи IoT. Класифікація показана на малюнку 1.4.

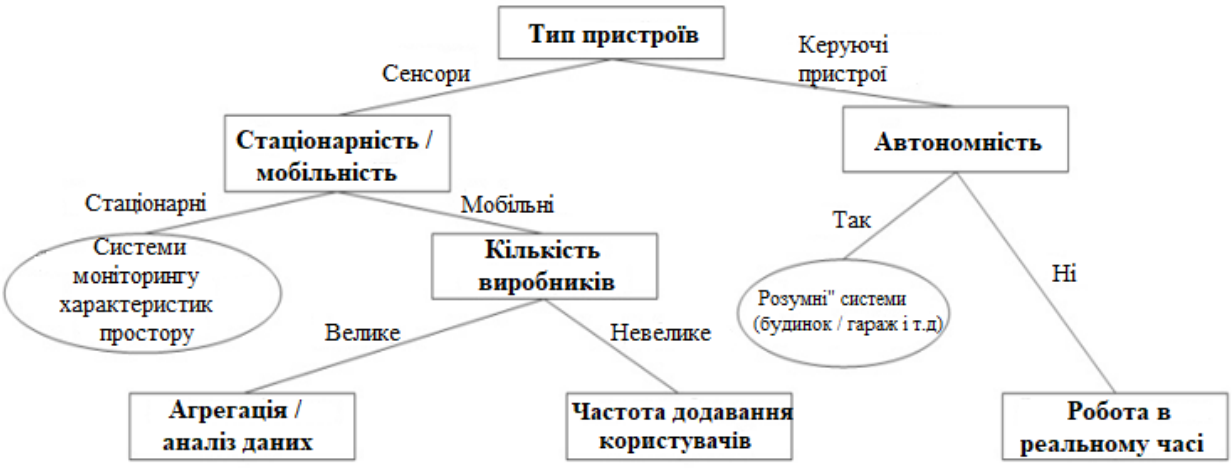


Рисунок 1.4 – Класифікаційні таксони предметної області

Зазначені класифікаційні таксони далі покладені в основу виробничої системи вибору параметрів якості. Відповідно, безліч рішень IoT розділяється залежно від типу пристроїв, їх стаціонарності / мобільності, кількості виробників, частоти додавання пристроїв, автономності, роботи в реальному часі, щоб в подальшому на підставі типу системи спрогнозувати необхідні параметри якості.

1.1 Моделі якості програмних систем і найважливіші параметри якості систем IoT

Для моделі зовнішньої та внутрішньої якості стандартом запропонована модель, що складається з 6 основних чинників і 27 параметрів. Основними атрибутами, зазначеними в даному стандарті, є:

- функціональна придатність;
- продуктивність;
- сумісність;
- зручність використання;
- надійність;
- захищеність;
- супроводжуваність;
- переміщення.

Кожен з цих атрибутів в свою чергу має підмножину характеристик. Вимірювання зазначених характеристик в більшості випадків проводиться з використанням метрик якості, що не описаних в новому стандарті, але широко висвітлених у документах ISO. Дані стандарти, зокрема, визначають поділ на зовнішні і внутрішні метрики, а також метрики якості у використанні, які вимірюють ефекти застосування програмного забезпечення в певному середовищі застосування. Спосіб застосування метрик описаний в технічних звітах, що застосовуються для будь-якого виду програмного забезпечення для будь-якої програми.

Короткий опис характеристик.

1. Функціональна придатність. Дана характеристика показує рівень того, наскільки бажаний продукт або система задовольняють заявленим функціональним вимогам. Властивість складається з трьох допоміжних характеристик:

- функціональна повнота. Ступінь, до якої безліч функцій покривають специфічні завдання і цілі користувачів;

- функціональна коректність. Рівень, до якого система надає коректні результати з заданим рівнем точності;

- функціональна доцільність. Характеристика, що показує, наскільки функціональність системи полегшує виконання тих чи інших завдань користувача.

2. Продуктивність. Показує ефективність використання ресурсів в ході виконання роботи. До складу включають:

- часові характеристики. Показують, наскільки відповідь системи та час обробки запиту задовольняє заявленим вимогам;

- використання ресурсів. Характеристика, що показує, які ресурси і в якій кількості використовуються системою;

- потенційні можливості. Міра, до якої максимальне споживання ресурсів системи задовольняє вимогам.

3. Сумісність. Показує ефективність взаємодії системи з іншими системами (підсистемами). Складається з наступних підхарактеристик:

- проіснування. Рівень, до якого продукт / сервіс може виконувати певні функції спільно з іншими продуктами в системі без прямого впливу на них;

- здатність до взаємодії. Показує, наскільки добре системи можуть обмінюватися інформацією між собою і «розуміти» її з семантичної точки зору.

4. Зручність використання. Визначає ефективність продукту для конкретного користувача (наскільки продукт / сервіс допомагає досягти мети).

Включає характеристики:

- визначеність придатності. Рівень, до якого користувачі можуть розпізнати продукт і його придатність для конкретного завдання;

- вивчаємість. Характеристика, що показує, наскільки система може бути зрозуміла і швидко освоєна новими користувачами;

- керованість. Рівень, до якого система є легкою в керуванні;

- захищеність від помилки. Ступінь, до якої система захищена від
- призначених для користувача помилок;
- естетика призначеного для користувача інтерфейсу. Рівень зручності взаємодії з користувачем;
- доступність системи. Показує, наскільки система може бути досяжна в відкритому доступі для різних груп користувачів.

5. Надійність. Визначає, наскільки система, продукт або компонент можуть зберігати заявлені функції в зазначений період часу при зазначених умовах.

Деталізація включає:

- захищеність. Рівень, до якого система, продукт або компонент задовольняють характеристикам надійності в звичайному стані;
- готовність. Рівень, до якого система доступна кожен раз, коли приходить запит на використання;
- відмовостійкість. Показує, наскільки система стійка до виходу з ладу частини ПЗ або апаратного забезпечення;
- відновлюваність. Рівень, до якого система може відновитися після збою зі збереженням налаштувань / даних користувача.

6. Захищеність. Рівень захищеності системи від зовнішніх / внутрішніх атак і неавторизованого доступу. Включає характеристики:

- конфіденційність. Характеристика, що показує, наскільки дані захищені від неавторизованих користувачів;
- цілісність. Рівень, до якого система підтримує цілісність даних;
- непідробність. Характеристика, що показує, наскільки авторство дій в системі неможливо підмінити;
- відстеження. Рівень, до якого дії в системі можна відстежити автентичність. Однозначна ідентифікація користувача в системі.

7. Супроводжуваність. Показує рівень, з яким система буде підтримуватися і модифікуватися. Складається з наступних допоміжних характеристик:

- модульність. Показує, до якого рівня система є подільною на частини;
- повторне використання. Характеристика, що показує рівень, до якого система може бути повторно використовувана;
- аналізованість. Визначає легкість аналізу модулів і функцій системи;
- модифікованість. Показує здатність системи до модифікації;
- тестування. Показує ступінь легкості та ефективності тестування системи.

8. Переміщення. Рівень ефективності перенесення системи з однієї апаратної / програмної бази на іншу. Складається з наступних допоміжних характеристик:

- адаптованість. Ступінь легкості, з якою програмна система може бути адаптована до різних типів апаратного забезпечення;
- встановлюванність. Характеристика, що показує ефективність установки / видалення на різних середовищах виконання;
- взаємозамінність. Визначає можливість заміни програмного продукту / сервісу схожим по функціональності.

1.2 Аналіз застосування параметрів якості програмної системи до систем IoT

В рамках дослідження, проведеного в IoT-A, було виділено вісім якостей системи, найбільш істотних для побудови систем IoT.

Даними параметрами якості є:

- здатність до взаємодії (інтероперабельність);
- здатність еволюціонувати;
- продуктивність;
- масштабованість;
- готовність;
- стійкість;

- інформаційна безпека;
- конфіденційність.

Для систем IoT зазначені характеристики є критичними, і багато досліджень показують важливість здатності до взаємодії і модифікованості для функціонування подібних систем.

Продуктивність визначається трьома вхідними характеристиками – тимчасовою поведінкою, споживанням ресурсів і потужністю. Архітектура системи може впливати на кожну з цих характеристик.

Стійкість програмної системи визначається через комбінацію відмовостійкості і відновлюваності (характеристики – складові надійності). Отже, можна в цілому замінити оцінку стійкості і готовності оцінкою надійності з Стандарту. Конфіденційність в даному випадку може бути розглянута як підхарактеристика захищеності.

Таким чином, в підсумкову безліч розглянутих атрибутів якості включені:

- здатність до взаємодії;
- модифікованість;
- продуктивність;
- модульність;
- надійність;
- захищеність.

Термін «програмна архітектура» відноситься до процесу прийняття рішень при розробці систем для задоволення специфічних вимог. Прийняті рішення є компромісом між параметрами якості розроблюваної системи, її вартістю і характеристиками і засновані на аналізі вимог до системи. Таким чином, рішення, прийняті на цьому абстрактному рівні, впливають на якість системи в цілому, яку вона буде демонструвати. Організація «Institute of Electrical and Electronics Engineers» (IEEE) визначає програмну архітектуру як «фундаментальну організацію систем, втілену в їх компонентах, їхні зв'язки один з одним, середовищем і принципах, що керують проектуванням і

еволюцією». Під час обговорення архітектури специфічних систем, архітектор системи абстрагується від деталей реалізації і фокусується на рішеннях, які необхідно здійснити для досягнення заданих показників системи. Ці рішення в кінцевому підсумку, визначають те, як система буде декомпозована на елементи і як вони згодом будуть взаємодіяти один з одним. Якщо подібні рішення починають застосовуватися для всіх систем певного типу для досягнення схожих цілей, дані способи декомпозиції стають видимими і отримують назву архітектурних шаблонів.

Дослідження процесу і методологій проектування ПЗ представлені в роботах R. Buchanan і A. Pranam. В процесі розробки програмної архітектури початкові вимоги до системи можна розділити на функціональні та нефункціональні, які фундаментально відрізняються за визначенням. Функціональні вимоги описують поведінку системи стосовно до очікуваних від неї дій, в той час як нефункціональні характеристики системи визначають параметри якості, яким розроблювальна система повинна відповідати. Таким чином, ключовим моментом в розробці програмної архітектури є декомпозиція системи на архітектурні елементи. Дана дія виконується найчастіше з урахуванням, в основному, функціональних вимог системи. У той же час, вже на момент розробки системи, особа яка приймає рішення (ОПР) не може брати до уваги виключно функціональні вимоги, необхідно також зважати на параметри якості системи. У цьому процесі в більшості випадків ОПР починає враховувати існуючі шаблони архітектури, так як використання шаблонів дозволяє заздалегідь включити в систему певні параметри якості. З іншого боку, програмна архітектура, будучи абстракцією, в той же час залишається специфічною для кожної системи. Архітектурний шаблон в свою чергу є абстракцією програмної архітектури в тому сенсі, що узагальнює безліч програмних архітектур.

Іншими словами, програмна архітектура може бути конкретизацією одного або безлічі архітектурних шаблонів. Архітектурні шаблони можуть розглядатися як інструкція по прийняттю рішень по декомпозиції системи

при проектуванні, що враховує тип компонентів системи і спосіб їх взаємодії. Шаблони засновані на досвіді проектування робочих систем. Шаблон архітектури ПЗ – іменована колекція проектувальних рішень, які можна застосувати до даного контексту, що обмежують розробку в даному контексті і також надають досяжні атрибути якості. Інше визначення архітектурного шаблону: «Скоординований набір архітектурних обмежень, який обмежує ролі / властивості архітектурних елементів і допустимих зв'язків між елементами в межах будь-якої реалізації архітектури, що належить цьому шаблону».

Виходячи з вищесказаного, можна резюмувати, що шаблон архітектури містить, крім елементів, обмеження як на склад, так і на взаємодію цих елементів з іншими.

Архітектурний стиль (шаблон) складається з чотирьох основних понять.

1. Перше – архітектурні елементи. Так само, як і в конкретній реалізації, можливо визначити шаблон як набір компонентів і зв'язків між ними. Як приклад можна привести клієнт-сервер, де компонентами є клієнт і сервер, а зв'язками – запити і відповіді між ними.

2. Другим елементом архітектурного стилю є вибір правил проектування системи. Правила проектування системи – безліч конфігураційних властивостей і топологічних обмежень, які визначають допустимі поєднання архітектурних елементів. Наприклад, правило, що елемент може бути з'єднаний тільки з двома іншими елементами.

3. Третя ланка є семантична зв'язаність, що означає, що певне поєднання елементів архітектури повинно мати значення, що добре простежується. Наприклад, що означає для класу реалізувати інтерфейс? Що означає для певного компонента зв'язок з іншим компонентом?

4. Четвертий елемент архітектурного стилю відстежує обмеження і допущення.

Кожен архітектурний шаблон має переваги, недоліки і певні типи

систем, для яких він застосовується найкраще. Ці припущення також відносяться до атрибутів якості.

Можлива ситуація використання декількох архітектурних шаблонів для створення схожої функціональності, при цьому сумарні параметри якості будуть залежати від конкретної реалізації і від суми реалізацій конкретних стилів. На рисунку 1.5 показаний процес розробки системи з використанням архітектурних стилів. У лівій частині зібрані вимоги замовника, що відображають цілі на розробку системи. В основі будь-якої вимоги лежить проблема, яку необхідно вирішити. Проблему в свою чергу можна визначити як різницю між поточним і цільовим станом. Деякі проблеми є абсолютно новими, при цьому можуть бути присутніми уривчасті знання про те, як вирішувати її частини. Подібні знання є знаннями предметної області. Кожна програмна розробка містить вимоги, що надаються замовником. Крім вимог, існують інші фактори, які можуть розглядатися як додаткові – цільова аудиторія продукту, ринок, конкуренти і т.д.

У підсумку, після визначення шаблону архітектури ПЗ, починається фаза реалізації проекту. Якщо рішення в кінцевому підсумку вирішує призначену для користувача проблему і задовольняє вимогам, воно може вважатися успішним. Якщо ні – цілі, архітектура і реалізація можуть ітеративно повторюватися до досягнення ситуації, прийнятною для замовника.

Отримана в результаті такого проектування архітектура ПЗ в загальному випадку є відображенням використовуваного архітектурного шаблону.

У світі існує величезна кількість різних програмних архітектур (так само, як і систем), але серед архітектурних шаблонів ПЗ в даний час використовується кінцева кількість широко використовуваних і певних. Вибір правильної початкової точки проектування для системи в кінцевому підсумку впливає на тривалість, вартість і трудомісткість проекту, а також надійність отримуваного продукту. Найчастіше

застосована тактика розробки адресує один параметр якості, хоча можливі ситуації, коли вона впливає на кілька атрибутів.

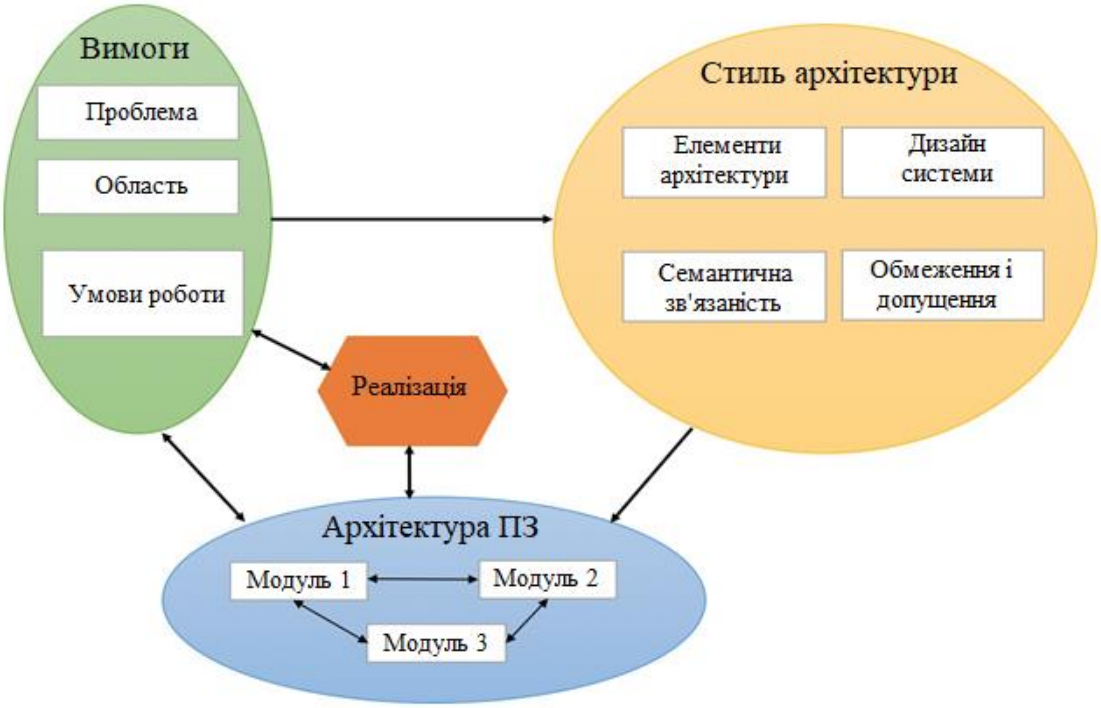


Рисунок 1.5 - Процес розробки ПЗ

Таким чином, тактики є своєрідними сценаріями, реалізованими в системі для досягнення параметра якості. Тому вибір тактик повинен бути безпосередньо пов'язаний з необхідними параметрами якості. Так як тактики і шаблони архітектури існують тільки в зв'язці один з одним, то реалізація тактик в кінцевому підсумку істотно змінює базовий шаблон, а шаблон архітектури в свою чергу впливає на спосіб реалізації тактики. Основні групи тактик розробки прив'язані до параметрів якості і включають:

- тактики для доступності системи;
- тактики для досягнення модифікованості;
- тактики для продуктивності;
- тактики для інформаційної безпеки;
- тактики для тестованості;

- тактики для здатності до взаємодії;
- тактики для досягнення модульності.

1.3 Зв'язок тактик розробки і шаблонів архітектури

Шаблони архітектури в загальному випадку є реалізацією кількох специфічних тактик в єдиній структурі. Наприклад, шаблон MVC (Model-View-Controller – модель-уявлення-контролер). Це спосіб організації коду, який передбачає виділення блоків, що відповідають за вирішення різних завдань, один блок відповідає за дані додатки, інший відповідає за зовнішній вигляд, а третій контролює роботу додатка. Шаблон MVC реалізує тактики інкапсуляції, використання проміжного компонента, реєстрацію під час виконання і семантичну пов'язаність. Однак той же самий шаблон MVC щодо побудови має недостатню продуктивність. Використання тактик збільшення ресурсів і використання декількох копій компонентів може виправити цю проблему. Таким чином, шаблони архітектури добре пристосовані для вирішення певної провідної проблеми, з якою з високою часткою ймовірності може зіткнутися система, але часто мають слабкі сторони по відношенню до інших параметрів якості. Тому для адресації необхідних додаткових параметрів якості, ОПР включає в систему сценарії, що забезпечують реалізацію додаткових тактик розробки. При цьому, реалізація тактик на різних шаблонах може істотно відрізнятися, так як сценарії в загальному випадку не регламентують методи і способи зв'язку елементів, а вказують «напрямок руху» в частині побудови архітектури.

Відмінності ж в області вихідної побудови шаблонів архітектури призводять до абсолютно різних значень трудомісткості і досяжності сценаріїв на конкретному шаблоні. Тому такою важливою в даний момент є роль експертної оцінки в побудові програмної архітектури – без знань про методи комбінації шаблонів і тактик, побудова програмної архітектури може стати досить складним завданням.

1.4 Методи оцінки трудомісткості програмних проектів

Серед способів оцінки витрат на інформаційні системи виділяються кілька основних типів:

- експертна оцінка. Метод застосовується в основному в проектах, які вирішують інноваційні завдання і використовують абсолютно нові технології. При цьому проводиться опитування інженерів-розробників в кілька стадій по Дельфійській або розширеній Дельфійській методиці. Суть цього методу полягає в тому, щоб за допомогою серії послідовних дій – опитувань, інтерв'ю, мозкових штурмів – домогтися максимального консенсусу при визначенні правильного рішення. Аналіз за допомогою дельфійського методу проводиться в кілька етапів, результати обробляються статистичними методами. Оцінки окремих експертів в подальшому приводяться до консенсусу. Надійність зазначеного методу сильно залежить від кваліфікації і досвіду присутніх експертів. Важливим недоліком методу є неможливість відтворити процес прийняття рішення кимось, крім самого експерта. Дана технологія успішно використовується в ситуаціях попередньої якісної оцінки трудомісткості;

- оцінка по аналогії. Метод як такої є різновидом експертної оцінки, але часто називається окремо. Оцінка заснована на емпіричних даних про попередні проекти зі схожими технологіями і характеристиками закінчених проектів. Для відбору проектів, близьких до досліджуваного, застосовується метод вимірювання Евклидової відстані в n-вимірному просторі, коли кожній характеристиці присвоюється значення ваги (множник), що визначає його значущість для проекту. Слід згадати, що параметричні методи також використовують дані про завершені проекти, але в цьому випадку дані швидше використовуються для коригування та калібрування параметрів моделей;

- параметричні моделі. У параметричних моделях час розробки і трудомісткість проекту обчислюється як функція від багатьох змінних. Дані

змінні представляють собою характеристики проекту і заявлені як найбільш важливі для формування вартості. В залежності від використання історичних даних, моделі поділяють на емпіричні та аналітичні, а по вигляду використовуваних функцій розрізняють лінійні, мультиплікаційні і степеневі. У даній роботі для оцінки трудомісткості був обраний клас параметричних моделей, зокрема, модифікована модель COCOMO II (COnstructive COst MOdel – модель витрат розробки). Це алгоритмічна модель оцінки вартості розробки програмного забезпечення, яка де-факто стала стандартом. Ця модель розроблена Баррі Боем (Barry Boehm) і використовує просту формулу регресії з параметрами, визначеними з даних, зібраних по ряду проектів.

1.5 Оцінка за методикою COCOMO II

COCOMO є найбільш прозорим на даний момент і добре документованим методом, який став стандартом в оцінці трудомісткості програмних систем. Метод, розроблений в 80-х роках XX століття, містить три моделі:

- базова – статична модель, що обчислює витрати, необхідні для розробки ПЗ, як функцію розміру програми, вираженої в рядках вихідного коду;

- проміжна – обчислює витрати, необхідні для розробки ПЗ, як функцію розміру програми і набору з 15 поправочних коефіцієнтів, що включають суб'єктивні оцінки;

- детальна (поглиблена, розширена) – модель включає всі характеристики проміжних версій з оцінкою впливу витрат на кожному кроці в процесі розробки.

COCOMO не дає відповідь на питання про розмір програмного проекту – до оцінки розмір (в KLOC) повинен бути відомий. У більшості випадків базова модель дає не дуже достовірні результати (так як грунтується тільки на розмірі проекту), тому для оцінки використовується просунута або

детальна модель. COCOMO надає кілька залежностей, заснованих на історичних даних про 63 проектах, виконаних в різних режимах: органічному (стабільне середовище розробки, мало інновацій, маленький розмір проекту), вбудованому (складний інноваційний проект, швидко мінливі вимоги) і усередненому режимах. Для оцінки проекту у всіх трьох моделях застосовується два основних рівняння (відмінності виявляються в ступені деталізації коефіцієнтів):

$$Q = a \cdot KLOCK^{b} \cdot EAF.$$

$$D = c \cdot Q^{d}.$$

де Q – трудовитрати в людино-місяцях $

α,b,c,d – коефіцієнти, що калібруються за статистичними даними;

KLOC – розмір програми в тисячах рядків вихідного коду;

D – тривалість проекту;

EAF – фактор уточнення витрат, який визначається як добуток мультиплікаторів. Для базової моделі даний фактор дорівнює одиниці, для проміжної є добутком 15 факторів вартості.

У порівнянні з вихідною моделлю COCOMO, методика COCOMO II, що стала її природним розвитком, адаптована до більш сучасних методів розробки ПЗ.

Наприклад, вона придатна для спіральної і итеративной моделі життєвого циклу. COCOMO II дозволяє використовувати дані, доступні на ранніх етапах життєвого циклу системи (зокрема - кількість функціональних точок).

Крім того, COCOMO II дозволяє оцінювати проекти, засновані на повторному використанні коду, додатку з об'єктно-орієнтованим підходом, а також враховувати вплив зрілості процесів розробки в компанії.

У COCOMO II можливо застосовувати три способи оцінки розміру

програмного продукту:

- в об'єктних одиницях;
- в функціональних точках;
- в кількості рядків вихідного коду.

Оцінка розміру проекту в об'єктних одиницях добре підходить для систем візуального проектування, які оперують поняттями «екранна форма», «звіт», «компонент». Але для об'єктно-орієнтованих мов програмування така оцінка є недоцільною.

Оцінка на базі функціональних точок, що описана в попередньому розділі, зручна тим, що грунтується на інформації, доступної на ранніх етапах проектування. Оцінка на базі рядків коду в порівнянні з вихідною моделлю COCOMO зазнала змін в частині визначення поняття «рядки коду».

У зв'язку з різними способами підрахунку виконуваних елементів для різних мов програмування, в моделі COCOMO II в якості одиниці використовуються логічні рядки коду (тобто рядки, які несуть в собі конкретні оператори і вирази, що дозволяють врахувати особливості мови на відміну від фізичних рядків коду). Існує три види моделі COCOMO II: модель композиції додатку, модель раннього етапу проектування і пост-архітектурна модель.

Модель композиції додатку COCOMO II застосовується для оцінки призначеного для користувача інтерфейсу. Метою є оцінка інтерфейсу, продуктивності, взаємодії з користувачем. Для оцінки використовуються об'єктні одиниці.

Модель раннього проектування COCOMO II застосовується для вивчення альтернативних архітектур і концепцій роботи. Модель включає використання функціональних точок і додаткові фактори витрат. Метою є оцінка доцільності використання апаратних і програмних засобів в процесі розробки. Для визначення розміру програми використовується поняття неприведеної функціональної одиниці, яка потім перетворюється в KLOC.

Пост-архітектурна модель COCOMO II використовується на етапі, коли

розроблена архітектура рішення, перевірені ризики, визначена структура компонентів. Метою моделі є управління витратами в процесі розробки продукту.

Для раннього етапу проекту, що розглядається в даній роботі, може бути застосована модель раннього проектування COCOMO II. Основні параметри моделі наведені нижче.

Трудомісткість проекту в моделі раннього проектування вважається як:

$$Q = \lambda \cdot (KLOCK)^{B} \cdot EAF + E_{auto}.$$

де Q – трудовитрати, виражені в людино-місяцях;

EAF – фактор коригування трудовитрат в залежності від середовища;

λ – фактор, що залежить від детальності оцінки (постійна величина, для попередньої оцінки рівна 2,94);

Eauto – характеристика, яка використовується в тому випадку, якщо деяка частина програмного коду генерується автоматично.

1.6 Завдання вибору архітектури для систем IoT

Критеріями вибору базового шаблону програмної архітектури та супутніх тактик розробки для систем IoT є:

- з одного боку – досягнення необхідних для реалізації параметрів якості системи;
- з іншого – мінімізація загальної трудомісткості програмного проєкту.

Визначення необхідних параметрів якості архітектури системи може включати в себе оцінку на основі класифікації систем IoT, або експертні оцінки.

Важливість вибору конкретного архітектурного шаблону вже на стадії прототипу обумовлена тим, що більшість архітектурних стилів:

- мають певні переваги і недоліки, добре знайомі експертам в галузі

архітектури ПЗ і впливають на параметри якості. В даний час вибір конкретної архітектури в програмних проектах визначається скоріше досвідом і емпіричною оцінкою відповідальної особи.

- бувши одного разу обраним, шаблон архітектури задає подальший тон розробці системи. У разі неправильно побудованої архітектури в подальшому будуть потрібні колосальні зусилля із перекроювання рішення з нуля, так як базовий архітектурний шаблон – це своєрідний «скелет» проекту.

Разом з тим, на початкових стадіях проекту, коли в наявності присутні тільки вимоги і обмеження на систему, стає складніше визначити придатність того чи іншого шаблону через нестачу даних, обмеженості знань експерта або його недоступності. У свою чергу, певна модель дозволяє оцінювати шаблони архітектури і тактики розробки на предмет застосування до конкретних функціональних вимог на систему (і супутнім необхідним параметрам якості) і передбачати дані показники програмної архітектури на основі неповних даних про систему.

В якості базових понять онтології розглянемо її складові частини. Безліч концептів онтології розробки архітектури ПЗ включають в себе наступні елементи:

- архітектура системи;
- особливість проектування;
- тактика розробки;
- особливість архітектури системи;
- параметр якості;
- шаблон архітектури;
- елементи шаблону архітектури;
- компоненти;
- зв'язки.

Архітектура системи пов'язана відношенням агрегації з реалізацією шаблону архітектури (так як реалізація може існувати поза контекстом

конкретної програмної архітектури). Реалізація шаблону архітектури в свою чергу реалізує інтерфейс шаблону архітектури, який визначає особливості архітектури системи і містить (відношення композиції) елементи шаблону архітектури. На елементи шаблону архітектури впливають вибрані тактики розробки, які в свою чергу визначаються параметрами якості і є особливістю архітектури.

Говорячи про структуру понять в області розробки архітектури необхідно звернути особливу увагу на зв'язок елементів шаблону архітектури і тактик розробки. Компоненти архітектури та зв'язку між ними є елементами шаблону архітектури (відношення узагальнення). Тактика розробки впливає на архітектуру шляхом впливу на компоненти шаблону і зв'язку між ними.

2 АЛГОРИТМ ВИБОРУ ПРОГРАМНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМ IoT

Завдання вибору базового шаблону і супутніх тактик розробки відноситься до класу комбінаторних задач структурного синтезу з мінімізацією цільової функції вартості. Обмеження щодо завдання, включають:

- структурні обмеження на комбінацію тактик і шаблонів: використання не більше двох основних шаблонів разом, невключення тактик, якщо вони реалізовані в шаблоні;
- обмеження переваги: в області побудови архітектури велику роль відіграють переваги (досвід) команди з певним шаблоном / тактиками розробки;
- обмеження на максимальну вартість рішення;
- мінімізація відносної вартості рішення, розрахованої за методиками COCOMO II або функціональних точок.

В якості методу вирішення зазначеного завдання було вибрано локальний алгоритм пошуку. В якості початкового значення для пошуку вибирався шаблон архітектури на основі призначених для користувача переваг. У разі відсутності переваг, вибиралася випадкова комбінація, яка задовольнить безлічі обмежень. Необхідною умовою для вирішення завдання конфігурації є визначення вектора досягнутих параметрів якості системи, описаних раніше. Локальний алгоритм пошуку був обраний, так як можливо досить точно визначити локальне оточення точки шляхом присвоєння початкової конфігурації виходячи з переваг користувача. В якості евристики використовується евристика з мінімальними конфліктами, яка знаходить найближчу конфігурацію-наступник виходячи з умови мінімізації кількості конфліктів з вже присвоєними змінними.

Запуск алгоритму виконується за трьома різними сценаріями (рисунок 2.1), що розрізняються характером застосовуваних обмежень або врахованих параметрів якості системи. У число зазначених сценаріїв входять:

- оптимальний сценарій – включає обмеження на максимальну вартість рішення, а також бере до уваги параметри з високим або середнім показником впливу;

- мінімальний сценарій – включає обмеження на максимальну вартість, при цьому враховує тільки атрибути якості з високим показником впливу;

- максимальний сценарій – враховує всі основні досяжні параметри якості систем, при цьому не бере до уваги обмеження вартості.

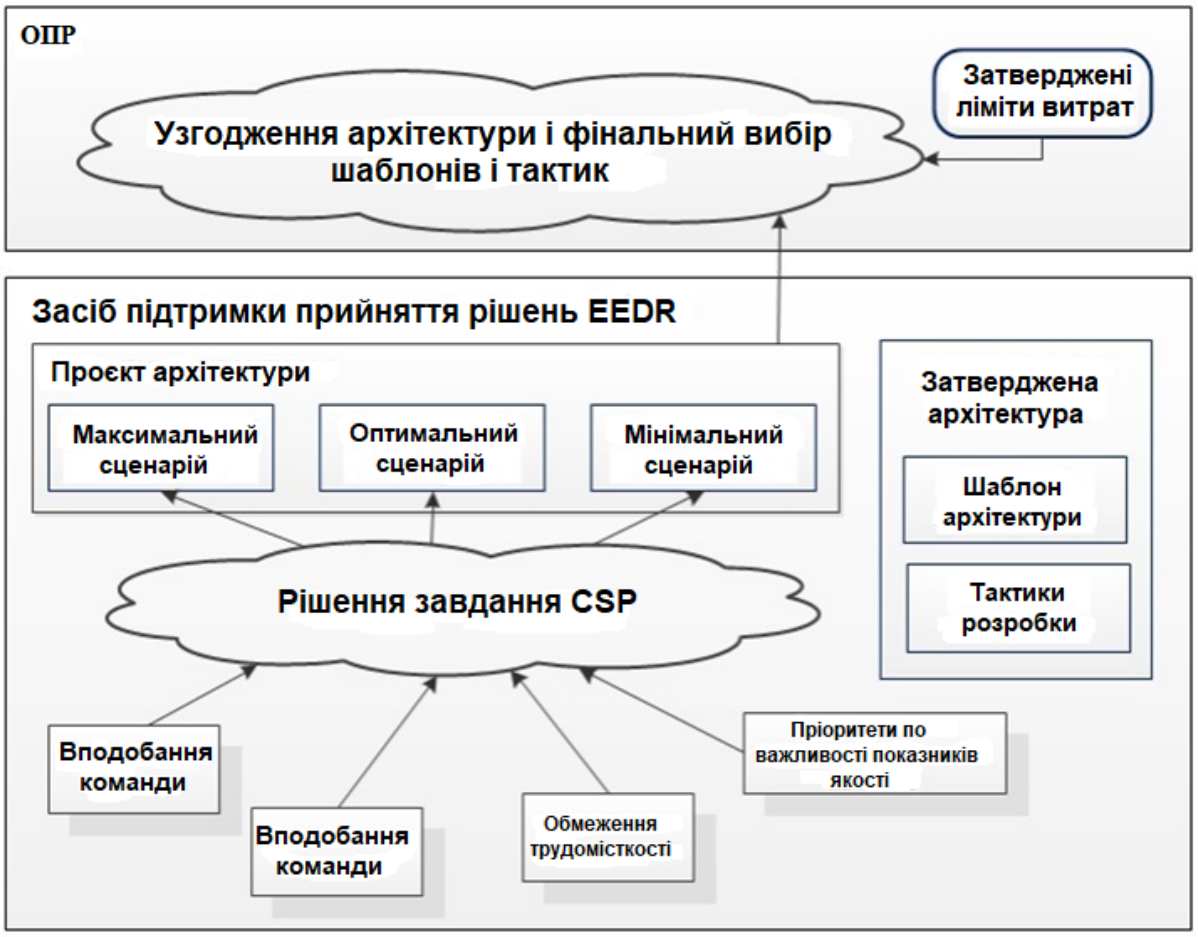


Рисунок 2.1 – Сценарії виконання алгоритму

Основні кроки алгоритму.

Крок № 1. Вибір сценарію роботи алгоритму. Основні типи сценаріїв роботи відрізняються характером обмежень, а також способом вибору параметрів якості.

Крок № 2. Формування вектора параметрів якості системи в залежності від обраного сценарію. На даному етапі запускається продукційна підсистема виведення і в залежності від введених вимог виділяються прогнозовані необхідні параметри якості.

Крок № 3. Введення переваг користувача в частині шаблонів архітектур. Введені символи використовуються для визначення початкової конфігурації рішення на етапі пошуку. Даний крок може бути опущений.

Крок № 4. Формується початкова конфігурація рішення на підставі вектора параметрів якості. У разі наявності обмежень переваги, пошук починається з задоволення цих обмежень, в разі їх відсутності – початкова конфігурація вибирається випадковим чином.

Крок № 5. Для обраної конфігурації вважається відносна трудомісткість Q за методикою COCOMO II.

Крок № 6. Для конфігурації формується окіл шляхом виділення набору конфігурацій із заміненим елементом (тактикою або шаблоном), який задовольнить вектору параметрів якості. На даному етапі застосовується евристика з мінімальними конфліктами.

Крок № 7. Для елементів окілу вважається відносна трудомісткість за методикою COCOMO II. Вибирається нова початкова конфігурація за умовою $Q \rightarrow \min$.

Розроблений алгоритм вибору шаблонів архітектури і тактик розробки лягли в основу функціонального моделювання методики вибору програмної архітектури, що отримала назву EEDR (Exploration – Evaluation – Definition – Resolution - Розвідка - Оцінка - Визначення - Дозвіл). На вхід процесу

надходять функціональні вимоги до системи, а також всі можливі досяжні параметри якості та затверджені ліміти витрат на проект. За допомогою функціонального опису, існуючих моделей якості і методики управління життєвим циклом, команда проекту, використовуючи засіб підтримки прийняття рішень, ідентифікує найбільш підходящі конфігурації архітектур проекту (з можливістю коригування). Після закінчення процесу є можливість відстежувати ефективність реалізації параметрів якості.

Основні кроки процесу EEDR.

Крок № 1. На підставі загальних досягнутих параметрів якості та функціональних вимог до системи формується конкретний вектор параметрів якості, що визначає, які з параметрів якості моделі (моделей) якості необхідні для реалізації в проекті. При цьому використовується розроблена продукційна система.

Крок № 2. Вектор параметрів якості, що надходить на вхід другого етапу, ставиться у відповідність тактикам розробки, що зберігаються в БД, необхідним для його досягнення. Поряд з цим, командою проекту виконується аналіз шаблонів архітектур з включеними тактиками. Як результат даного етапу, формується безліч шаблонів архітектури і тактик розробки, що представляють собою область визначення для завдання задоволення обмежень.

Крок № 3. На підставі вектора параметрів якості та затверджених лімітів витрат на проект виконуються три сценарії знаходження конфігурації шаблонів архітектур і тактик розробки. Для кожного з сценаріїв запускається алгоритм локального пошуку з урахуванням відповідних типів обмежень (в тому числі на ліміти витрат). На зазначеному етапі виконується відносна оцінка (параметрезована інтервалом K) кращої архітектури користувача.

Крок № 4. Відносна оцінка (параметрезована K) надходить на вхід наступного етапу, де командою проекту на підставі історичних даних та

експертної оцінки виконується уточнена оцінка обсягу в межах інтервалу K, а отже – скоригована оцінка трудомісткості. На даному етапі можливо варіювання користувачами деяких тактик і шаблонів (видалення, заміна на схожі за функціональністю) і нова оцінка відповідно до кроку № 3.

Крок № 5. Верифікація підходу виконується після / під час розробки проекту. Конкретна фаза залежить від використовуваної методології життєвого циклу розробки. На даному етапі виконується оцінка фактичних необхідних параметрів якості розробленої або розроблюваної системи відповідно до метрик обраної моделі якості. Отриманий результат інтерпретується як досягнення або недосягнення зазначеного параметра, що показується користувачеві у вигляді звіту.

2.1 Розробка програмного засобу підтримки прийняття рішень при виборі архітектури систем IoT

Додаток містить два підмодуля, що реалізують сценарії взаємодії експерта і користувача з системою. Призначений для користувача підмодуль дозволяє взаємодіяти з рішенням, виконувати введення-виведення початкових даних і аналізувати результати. Підмодуль експерта в свою чергу відповідальний за завантаження правил, сценаріїв роботи і обмежень.

Правила та обмеження зберігаються в окремій області пам'яті, недоступній для запису користувачеві. Структурна схема системи приведена на рисунку 2.2. Користувач взаємодіє з системою через інтерфейс, завантажуючи вхідні дані, такі як функціональні вимоги до системи, переваги та обмеження (підмножина, яку дозволено змінювати). Після перевірки вимог, дані завантажуються в базу даних, що містить поточний стан. При запуску вирішувача, на підставі призначених для користувача даних, також приймаються в розрахунок бази незмінних обмежень і правил, заповнені експертом. Проміжні результати розрахунку, такі як вектор параметрів якості

і список конфігурацій з відносною вартістю, також показується користувачеві через інтерфейс.

На рівні експерта можлива взаємодія з базою правил додатка (перегляд/ зміна/завантаження). Вводяться основні сценарії, а також реалізовано додавання або налаштування обмежень, які є незмінними для користувача системи.

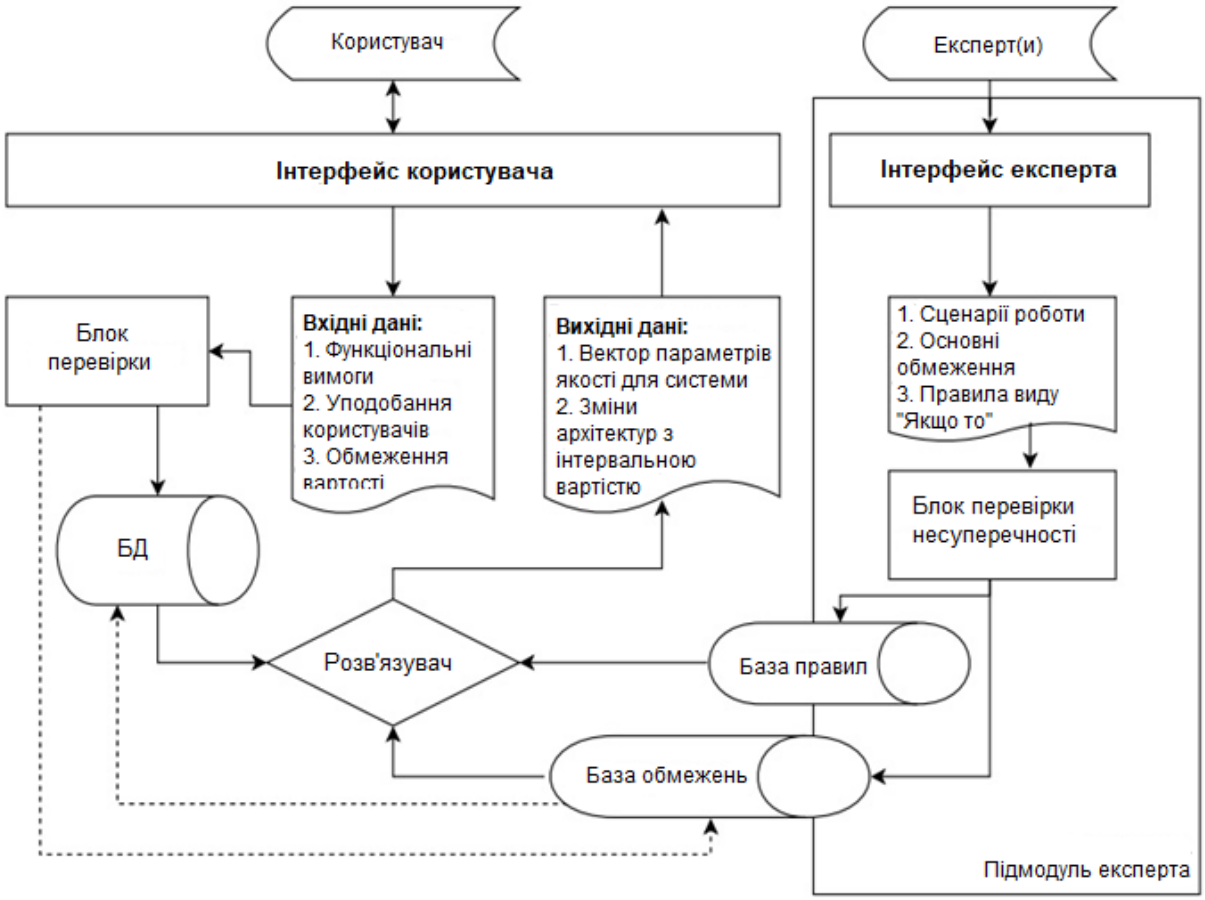


Рисунок 2.2 – Структурна схема СППР EEDR і управління процесом проектування

При цьому виконується перевірка несуперечності правил і обмежень. На рисунку 2.3 показана логічна модель зв'язку основних збережених сутностей процесу і зв'язку між ними.

Діаграма розбита на підсистеми, відповідальні за різні функціональні

кроки процесу, що дозволяє логічно ізолювати пов'язані модулі.

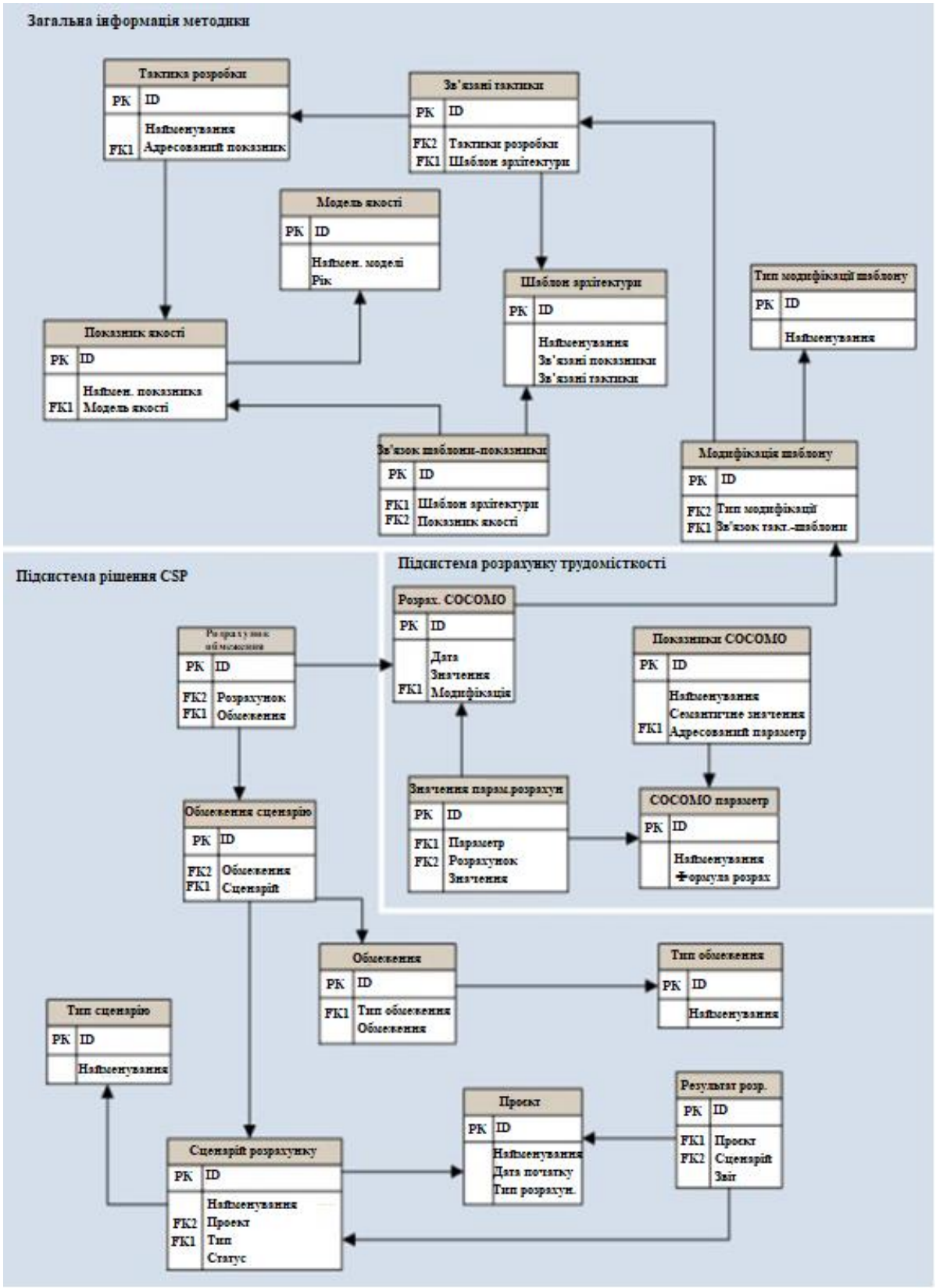


Рисунок 2.3 – ER- діаграма сутностей додатку

Представлені наступні ключові сутності, що використовуються в процесі вибору програмної архітектури, логічно розбиті по підсистемах.

1. Підсистема зберігання загальної інформації. Містить основні відомості про зв'язок тактик розробки і шаблонів архітектури, списки тактик і шаблонів, а також показники якості і моделі якості програмних систем.

Основні сутності:

- тактика розробки – інформація по тактиці розробки, що включає основні атрибути:

1) id тактики – унікальний ідентифікаційний код тактики;

2) найменування – ім'я в літературі;

3) адресований показник – пов'язаний показник якості, що досягається за допомогою застосування тактики;

- показник якості – інформація про показник якості ПЗ:

1) id показника – унікальний ідентифікаційний код показника;

2) найменування – ім'я в моделі якості;

3) модель якості – модель, яка містить даний показник;

- модель якості - модель, яка використовується для оцінки:

1) id моделі – унікальний ідентифікаційний код;

2) найменування – найменування стандарту;

3) рік – рік прийняття стандарту;

4) пов'язані показники – показники якості, що мають високі значення для даного шаблону;

5) пов'язані тактики – тактики розробки, реалізовані в зазначеному шаблоні;

- шаблон архітектури – перелік основних шаблонів архітектури:

1) id моделі – унікальний ідентифікаційний код;

2) найменування – загальноприйняте найменування шаблону;

- тип модифікації шаблону – сутність, яка адресує основні типи модифікацій:

1) id типа – унікальний ідентифікаційний код;

2) найменування – найменування модифікації;

- модифікація шаблону – конкретний тип модифікації, який можна застосовувати для кожної пари «тактика розробки-шаблон архітектури»:

1) id модифікації – унікальний ідентифікаційний код;

2) зв'язок тактика-шаблон – зв'язка, до якої може бути застосована модифікація;

- зв'язок тактики-шаблони – пари тактик розробки і шаблонів архітектури:

1) id связки – унікальний ідентифікаційний код;

2) тактика розробки – зв'язана тактика;

3) шаблон архітектури – зв'язаний шаблон;

- зв'язок шаблони-показники – сутність, яка ідентифікує адресовані показники якості в шаблоні архітектури:

1) id зв'язки – унікальний ідентифікаційний код;

2) шаблон архітектури – зв'язаний шаблон;

3) показник якості – показник, що адресується в шаблоні.

2. Підсистема рішення CSP. Зберігаються поточні дані про сценарії розрахунку, розрахунках, поточних обмеженнях для сценаріїв і результати розрахунку.

Основні сутності:

- тип сценарію – представляє один з трьох можливих типів сценаріїв:

1) id типа сценарію – унікальний ідентифікаційний код;

2) найменування – найменування типу сценарію;

- сценарій розрахунку – конкретний сценарій:

1) id сценарію – унікальний ідентифікаційний код;

2) найменування сценарію – найменування;

3) проєкт – зв'язок з проєктом;

4) тип – тип сценарію;

5) статус – поточний крок сценарію;

- проєкт – представляє сутність проєкту:

1) id проекта – унікальний ідентифікаційний код;

2) найменування – найменування проекту;

3) дата початку проекту;

- обмеження сценарію – зв'язок-сутність між сценаріями і застосовуваними в них обмеженнями:

1) id – унікальний ідентифікаційний код;

2) обмеження – застосовуване обмеження;

3) сценарій – сценарій, для якого може бути застосовано обмеження;

- обмеження – представляє сутність обмеження в алгоритмі:

1) id – унікальний ідентифікаційний код;

2) тип обмеження – тип (перевага, вартість і т.д.);

3) обмеження – ім'я обмеження;

- розрахунок-обмеження – визначає функцію розрахунку для обмеження вартості:

1) id – унікальний ідентифікаційний код;

2) расчет – посилання на розрахунок за методикою COCOMO II;

3) обмеження – посилання на обмеження;

- тип обмеження – визначає тип обмеження:

1) id – унікальний ідентифікаційний код;

2) найменування – ім'я типу обмеження (вартості, переваги, конфігурації);

- результат розрахунку – визначає представлення результату:

1) id – унікальний ідентифікаційний код результату розрахунку;

2) проєкт – посилання на проєкт;

3) сценарій – посилання на сценарій;

4) звіт – представлення звіту.

3. Підсистема розрахунку трудомісткості зберігає і оперує даними

розрахунку за методикою COCOMO II, в тому числі поточними показниками і загальними значеннями коефіцієнтів.

Основні сутності:

- показники COCOMO – визначає коефіцієнти методики, що застосовуються при обчисленні параметрів:

1) id – унікальний ідентифікаційний код показника;

2) найменування – найменування показника;

3) семантичне значення – значення, яке стоїть за даним показником;

4) параметр, що адресується – параметр, при розрахунку якого застосовується зазначений показник;

- COCOMO параметр – обчислюваний параметр за методикою COCOMO:

1) id – унікальний ідентифікаційний код параметра;

2) найменування – найменування параметру;

3) формула розрахунку – формула, котра зв'язує показники і параметр;

- значення параметра розрахунку – значення показників, прив'язаних до поточного розрахунку:

1) id – унікальний ідентифікаційний код показника;

2) параметр – параметр, що адресується;

3) розрахунок – посилання на поточний розрахунок;

4) значення – поточне значення параметра;

- розрахунок COCOMO - розрахунок, прив'язаний до обмеження вартості і адресує конкретний тип модифікації шаблону:

1) id – унікальний ідентифікаційний код показника;

2) дата – дата розрахунку;

3) значення – значення трудомісткості;

4) модифікація – посилання на тип модифікації шаблону.

2.2 Модель вибору базового шаблону архітектури і тактик розробки систем IoT

Вважаємо, що тип архітектури програмної системи складається з сукупності елементів, обмежень, семантичних правил і припущень:

$$A = \{E_T, O_T, C_T\}.$$

де E_T – елементи;

O_T – обмеження;

C_T – семантичні правила.

Елементи типу, в свою чергу, представлені компонентами і зв'язками базового шаблону архітектури, на реалізацію якого витрачається вартість і час проектної команди:

$$E_T = \{K_T, C_T\}.$$

де K_T – компоненти;

C_T – зв'язки між ними.

Завдання вибору базового шаблону архітектури і тактик розробки для систем IoT полягає в знаходженні такої сукупності компонентів і зв'язків між ними для базового шаблону і безлічі тактик, реалізація яких призводить до досягнення необхідних функціональних параметрів і параметрів якості системи при мінімізації трудомісткості.

Вважаємо, що критерієм досягнення параметра якості програмної системи є реалізація в обраній архітектурі безлічі тактик розробки $T = \{t_i\}, i = 1, 2, ..., n$, відповідальних за згаданий параметр якості. При цьому, деякі шаблони програмних архітектур по своїй вихідній побудові задовольняють певним параметрам якості. Дану умову висловимо за

допомогою булевої змінної x_{ij}, що визначає відповідність базового шаблону архітектури і необхідному параметру якості j:

$$x_{ij} = \begin{cases} 1 \text{ реалізація тактик для параметра якості не потрібна} \\ 0 \text{ реалізація тактик для параметра якості необхідна} \end{cases}.$$

Для вирішення завдання вибору шаблону архітектури і тактик розробки введемо поняття вектора прогнозованих параметрів якості системи (або просто вектора параметрів якості). Ним будемо називати сукупність параметрів якості, що залежить від типу системи IoT $S_t = \{S_{i,t}\}$, где $i = l,...k$ – параметр якості, $t = l,...k$ – тип системи. Кожний параметр якості характеризується або реалізацією його в базовому шаблоні архітектури, або сумою тактик розробки, необхідних для його досягнення на базовому шаблоні:

$$S_j = \begin{cases} x_{ij} \\ \sum_{k=1}^{m} T_k \end{cases}.$$

де T_k – тактика розробки;

m – кількість тактик, необхідне для реалізації параметра $x_{ij} = 1$, якщо реалізація тактики розробки для досягнення не потрібна, $x_{ij} = 0$ – в зворотньому випадку.

Тактика розробки, в свою чергу, може бути представлена як модифікація елементів архітектурного шаблону (компонентів і зв'язків між ними). Однак реалізація одних і тих же тактик розробки для різних базових шаблонів архітектури може істотно відрізнятися, а, отже, відрізнятися буде і трудомісткість реалізації. Для того щоб врахувати це обмеження, було введено поняття типу зміни шаблону архітектури тактикою. Тип зміни

шаблону, по суті, відображає сутність модифікації шаблону архітектури і, отже, впливає на трудомісткість.

2.3 Розрахунок інтервальної функції оцінки за допомогою методики COCOMO II

Розрахуємо відносну трудомісткість реалізації додавання компонента в базовий шаблон архітектури, що складається з n - компонентів. Трудомісткість проекту задається формально як функція залежності від кількості тисяч рядків коду (без урахування фактора E_{auto}):

$$Q = \alpha \cdot KLOC^{b} \cdot EAF.$$

Тоді вихідну трудомісткість реалізації (базового шаблону) можна уявити, як:

$$Q_0 = \alpha \cdot KLOC_0 \cdot EAF.$$

де $KLOC_0$ – кількість тисяч рядків коду на реалізацію в базовому шаблоні.

Так як детальність оцінки залишається незмінною, реалізація додаткового компонента залежить від кількості тисяч рядків коду KLOC реалізації і фактора масштабу b. Далі отримаємо трудомісткість готового проекту через суму трудомісткостей компонентів. При цьому загальна трудомісткість не дорівнює сумі складових (так як не враховує зв'язків між елементами і витрат на їх інтеграцію).

Загальний розмір проєкту обчислюємо як суму розмірів його n - модулів:

$$KLOC_0 = \sum_{i=1}^{n} KLOC_i$$

Трудомісткість базового шаблону проекту:

$$Q^{база} = \alpha \cdot KLOC_o^b \cdot SCED.$$

де SCED – фактор стиснення розкладу. Виходячи з цього, базову трудомісткість кожного компонента приймаємо:

$$Q_i^{база} = Q^{база} \cdot \frac{KLOC_i}{KLOC_o} = \alpha \cdot SCED \cdot KLOC_o^{b-1} \cdot KLOC_i .$$

Після цього розраховуємо реальну оцінку компонентів (з урахуванням решти чинників середовища):

$$Q_i = Q^{база} \cdot \prod_{j=1}^{6} SF_i = \alpha \cdot SCED \cdot KLOC_o^{b-1} \cdot KLOC_i \cdot \prod_{j=1}^{6} SF_i .$$

де SF_i – фактори середовища. Тоді загальна трудомісткість n-компонентного базового шаблону буде дорівнювати:

$$Q_0 = \sum_{i=1}^{n} Q_i .$$

Переходячи до інтервального числення, приймаємо, що кількість тисяч рядків коду компонента проєкту є інтервалом $\tilde{K}_i$ з межами $[x, \tilde{x}]$. Тоді функція трудомісткості є інтервальною оціночною функцією $\tilde{Q}$ від величини $\tilde{K}_i$. При цьому виконується обмеження:

$$\sum_{i=1}^{n} \overline{x}_i \le KLOC .$$

де KLOC – об'єм коду всього проекту.

Далі, розглядаючи реалізацію тактик розробки, приймаємо, що кількість рядків коду також є інтервалом, величина якого залежить від типу зміни шаблону.

Фактор масштабу, при використанні в цій залежності, є, по суті, також залежним від типу зміни шаблону (так як ризики, архітектура і т.д. залежать від типу зміни).

В якості базової величини для порівняння трудомісткості розглянемо трудомісткість розробки одного компонента шаблону на початковій стадії проекту.

На рисунку 2.4 показана інтервальна оцінка трудомісткості розробки компонента шаблону в інтервалі від 1 до 2 тис. рядків коду при обраному значенні фактора масштабу для початкової фази проєкту, що дорівнює 1,09.

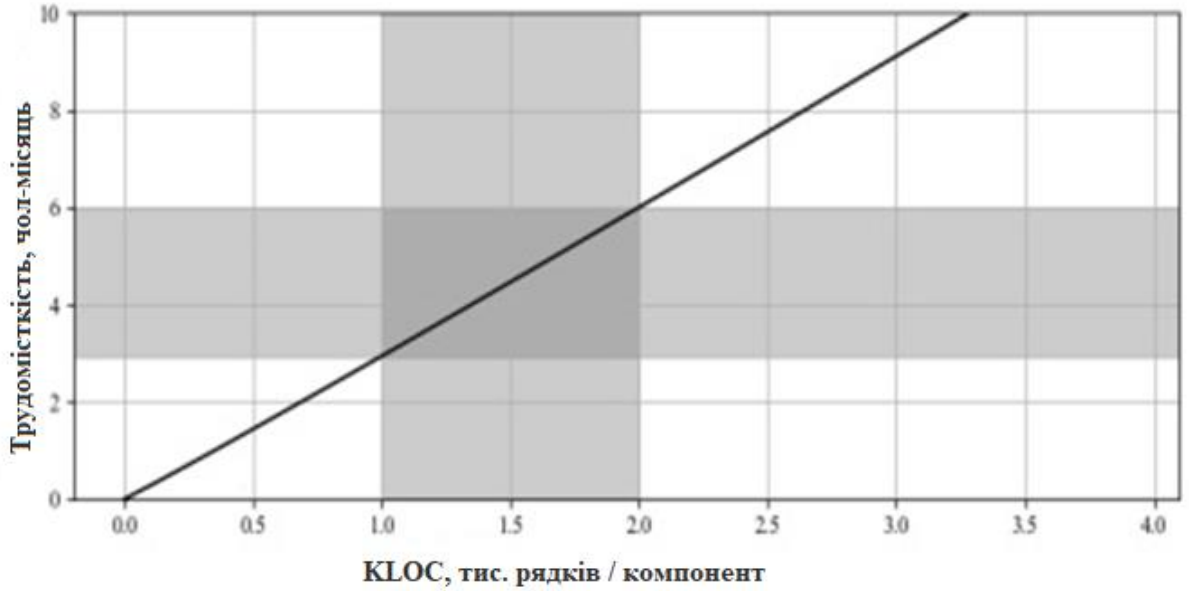


Рисунок 2.4 – Базове значення інтервалу трудомісткості при фіксованій величині інтервалу

На рисунку 2.5 показані значення інтервалу трудомісткості проекту при

зміні обсягу коду від 1 до 2 тис. рядків / компонент в порівнянні з базовим.

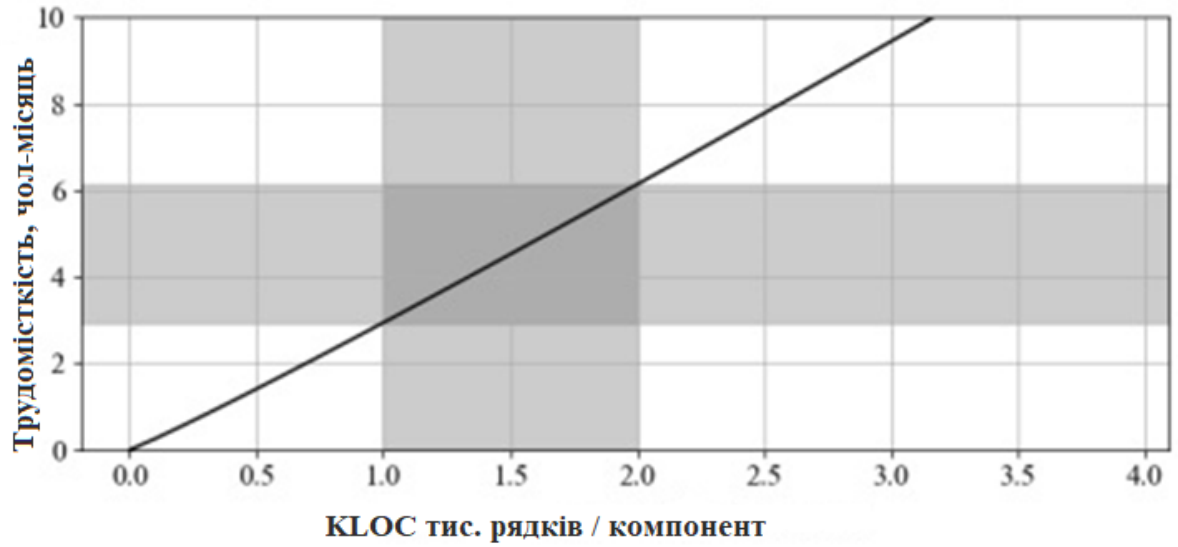


Рисунок 2.5 – Значення інтервалів трудомісткості при фіксованій величині інтервалу (додавання компонента в шаблон)

Загальна трудомісткість проєкту для обраного базового типу складається з базової трудомісткості реалізації шаблону (вираженої через інтервальний параметр $\tilde{K}_k$) і суму трудомісткостей n - тактик розробки:

$$\tilde{Q} = \tilde{Q}_0(t) + \sum_{i=1}^{n} \tilde{Q}_l(t,r) = \alpha \cdot EAF'((n \cdot \tilde{K}_k)^{b(m)} + \sum_{i=1}^{n} \tilde{Q}_l(t,m)).$$

Величина $\tilde{K}_k$ є інтервальним параметром, що залежить як від шаблону архітектури, так і від реалізації компонента, і обчислюється із застосуванням історичних і експертних даних на фінальному етапі розробки. Завдання, таким чином, зводиться до знаходження такої комбінації базового шаблону архітектури і тактик, при яких величина інтервалу трудомісткості є мінімальною:

$$\tilde{Q} \to \min.$$

Задача відноситься до класу комбінаторних задач структурного синтезу із застосуванням функції оптимізації з інтервальним параметром.

Так як величини $Q_1....Q_n$ (n - кількість можливих конфігурацій) є довільно розташованими інтервалами відносно один одного, то розрахована мінімальна трудомісткість, дозволяє перейти від завдання «відшукання мінімального з інтервалів» до еквівалентної задачі «відшукання мінімального з центрів інтервалів» $M_{Q_1}....M_{Q_n}$. Таким чином, результуюча функція оптимізації представляється як:

$$M_Q \to \min .$$

3 ПРОЄКТ ДЛЯ ДОМАШНЬОЇ АВТОМАТИЗАЦІЇ «ECLIPSE SMART HOME»

Платформа «Eclipse SmartHome™» є проміжною ланкою для побудови призначених для користувача рішень, що відносяться до класу домашньої автоматизації. Основними цілями платформи заявлені уніфікація протоколів, підходів і стандартів при розробці систем класу «Розумний будинок», полегшення створення кінцевих продуктів і надання низькорівневих сервісів для їх подальшого розширення.

Платформа надає наступні основні функції:

- управління даними – наявність розширюваного модуля для управління апаратними системами домашньої автоматизації, механізмів абстракції даних і доступу до пристроїв, модуля управління подіями для обміну інформацією;

- управління правилами взаємодії пристроїв – наявність гнучких модулів побудови правил взаємодії на етапі виконання програми, що дозволяє створювати логіку роботи;

- декларативне створення призначених для користувача інтерфейсів, включаючи сторінки, віджети, іконки;

- управління зберіганням – журнал аудиту доступу до даних з можливістю його розширення;

- можливість інтеграції з мінливими стандартами – реалізація та надання бібліотек для основних протоколів в області взаємодії пристроїв.

До додаткових особливостей системи можна віднести:

- відсутність обов'язкового хмарного сервісу, при цьому наявність можливості інтеграції з ним;

- наявність інтегрованого середовища розробки для можливостей розширення платформи і протоколів новими пристроями;

- підтримка Plug-and-Play інтерфейсу для сумісних пристроїв.

До особливостей і необхідним вимогам до якості систем типу «Розумний будинок» можна віднести:

- високі вимоги показника здатності до взаємодії. З'єднання в єдину систему пристроїв від різних виробників (що зазвичай є нормою в даному випадку) веде до змішання протоколів, стандартів і методів передачі даних;

- толерантні вимоги до продуктивності платформи. Так як вузьким місцем в даному випадку є кінцеві пристрої, до їх взаємодії не пред'являється вимог роботи в реальному часі. Продуктивність, таким чином, є не критичною вимогою;

розташування пристроїв на відносно невеликій території робить можливим використання локального сервера при розробці подібних систем. Це означає, що в більшості випадків можна обійтися без хмарного сервісу.

Розгляд програмної архітектури проекту полягала в аналізі документації та відкритого вихідного коду рішення.

3.1 Архітектура системи

Побудова архітектури обстежуваної платформи заснована на шаблоні мікросервісів. Проект написаний на мові Java з використанням стандарту і бібліотек OSGi, що має на увазі модульну сервісну архітектуру самого рішення. Використання мікросервісів є, з одного боку, еволюцією сервісноорієнтованого підходу (так як включає концепцію «сервісів»), з іншого – архітектури шарів. Тому даний шаблон не є, строго кажучи, абстракцією, створеною для вирішення конкретної проблеми - шлях від архітектури шарів до архітектури мікросервісів, наприклад, був викликаний необхідністю безперервного оновлення продукту. Монолітні додатки в даному випадку складаються з сильно пов'язаних компонентів, зміна яких веде до зміни всього рішення, складності тестування і розгортання. Ці фактори часто викликали помилки при розгортанні нової версії. Архітектура мікросервісів вирішує дані проблеми, розділяючи монолітний додаток на

кілька модулів (сервісів), що самостійно розгортаються і які можуть бути встановлені індивідуально, окремо від інших. Подібно шаблону архітектури SOA, архітектура мікросервісів також оперує поняттям «сервіс», але долає проблеми складності реалізації, дорожнечі і складності сприйняття зазначеного шаблону. Сервіси в архітектурі мікросервісів в порівнянні з SOA є менш навантаженими по функціональності, що спрощує їх розробку і взаємодію.

Сервіс в поняттях даного архітектурного шаблону є сутністю, що складається з одного або декількох модулів (класів / функцій / процедур), які представляють одну вузькоспеціалізовану дію (наприклад, повернути температуру для геолокації) або незалежну частину бізнес-додатку (наприклад, розрахунок заробітної плати). Визначення відповідного рівня грануляції є найважливішим викликом в проектуванні архітектури мікросервісів.

Відмінною особливістю архітектури мікросервісів є розпорошеність модулів. Кожен сервіс повністю індивідуальний, відокремлений від решти і надає доступ по одному з протоколів (JMS, SOAP, RMI, і т.д.). Дана властивість позитивно впливає на масштабованість, а також шаблон за визначенням має високі показники модульності. Іншою важливою особливістю є те, що сервіси можливо розгортати окремо один від одного. Як показано на рисунку 3.1, кожен компонент архітектури може бути розгорнений без зв'язку з іншими.

OSGi є специфікацією динамічної модульної системи і сервісної платформи, розробленої на мові Java. Вона визначає модель побудови програми з компонентів, склад яких здатний змінюватися під час виконання програми.

У термінах OSGi кожен компонент (сервіс) є «зв'язкою», яка має власний незалежний від усього рішення життєвий цикл. Дана зв'язка є логічно завершеним програмним компонентом, які реалізують певний функціонал. Це означає, що вона може бути запущена, зупинена і видалена

незалежно від інших компонентів. Специфікація OSGi також надає можливість отримувати повідомлення при запуску / зупинці сервісу або його видаленні.

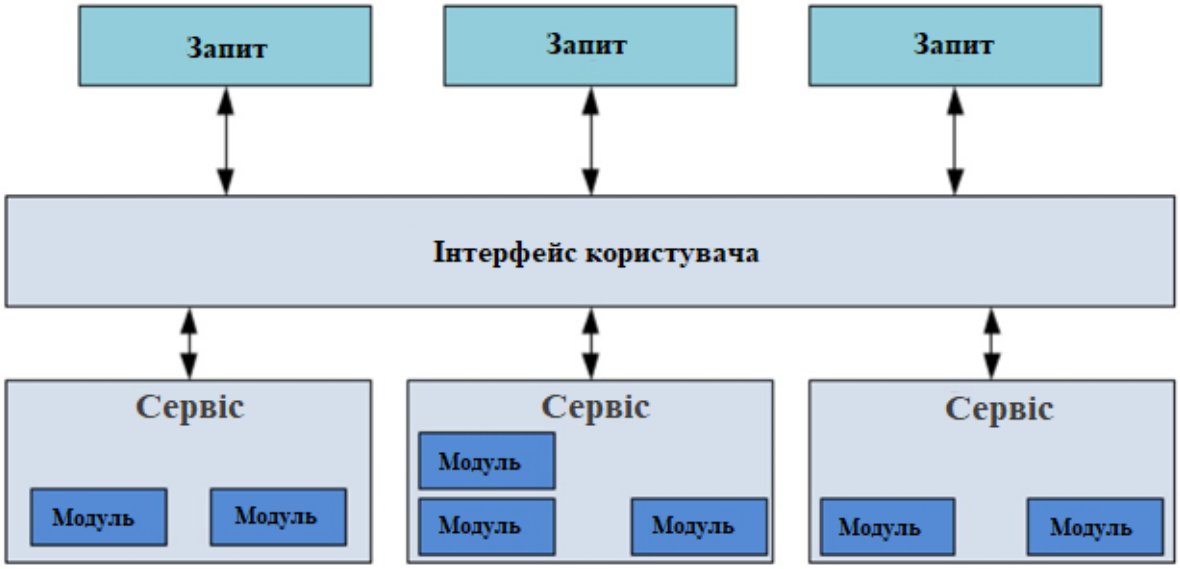


Рисунок 3.1 – Архітектура OSGI

Компоненти зв'язки OSGi включають в себе:

- специфікацію, в якій визначаються параметри;
- клас-активатор пакета;
- класи та ресурси, які реалізують функціонал.

Життєвий цикл зв'язки OSGI включає в себе наступні фази:

- встановлено – зв'язка знаходиться в системі;
- визначено – зв'язка готова до старту, визначені всі залежності.
- стартує – виконується метод старту;
- активний – зв'язка працює (метод старту повернув значення);
- зупиняється – виконується метод зупинки;
- вилучений – життєвий цикл зв'язки завершено.

Істотним недоліком даної специфікації є те, що сервіс запуститься на платформі тільки при наявності всіх встановлених залежностей, в той час як

клієнт до сервісу в цей час може працювати в обмеженому режимі (відсутня частина функціональності).

Архітектура мікросервісів, яка використовується в даній платформі, дозволяє зробити висновок про добру модульність, а також модифікованість і здатність до взаємодії рішення. Серед модулів платформи є сервіс виявлення, абстрактні загальні сервіси, файли конфігурації і інші тактики для зазначених параметрів якості.

Серед явно використовуваних тактик розробки, додатково використаних в проекті, були виділені:

- тактики, що адресують доступність: пульсація, обробка виключень, пасивна надмірність, транзакційність;
- тактики, що адресують модифікованість: узагальнення модулів, використання проміжних компонентів, реєстрація під час виконання;
- тактики для здатності до взаємодії: сервіс виявлення;
- тактики для інформаційної безпеки: авторизація та аутентифікація, обмеження впливу, перевірка доступності.

Зазначені групи тактик визначають додаткові параметри якості архітектури. Найбільша кількість явно простежуваних тактик розробки було адресовано до параметру доступності системи (що не є сильною стороною архітектури мікросервісів).

4 ТРУДОМІСТКІСТЬ РОЗРОБКИ СИСТЕМИ

Дані про трудомісткість розглянутого проекту були отримані виходячи з аналізу збільшень кількості вихідного коду в основну гілку сховища, а також кількості розробників і розподілу за часом частоти фіксацій змін.

На рисунку 4.1 показано кількість фіксацій, розподілене на часовому проміжку від 5 січня 2014 до 15 липня 2019 (час існування проекту).

Рисунок 4.1 – Розподіл фіксацій змін у часі з початку існування Проекту 1[45]

З огляду на відкритий характер вирішення, а також ступінь рівномірності роботи, для кожного з учасників були прийняті до уваги час першої і останньої фіксації змін коду в сховище, між якими вважалося, що учасник повністю працює над вирішенням.

При цьому не бралися до уваги учасники, які мають менше 10 фіксацій змін, за винятком тих, хто в цілому вніс більш ніж 1 KLOC.

4.1 Застосування методики проектування програмної архітектури та СППР EEDR

Перший етап верифікації методики полягав в знаходженні вектора

параметрів якості та перевірки роботи алгоритму для отримання трьох варіантів сценаріїв.

Застосовуючи вхідні дані до функціональних особливостей двох систем, отримуємо вектори параметрів якості систем, що складаються з наступних атрибутів:

$$\overline{П} = [П_i] = [\text{Інтероперабельність}, \text{Модифікованість}, \text{Доступність}, \text{ІБ}].$$

Отримані вектори параметрів якості використовується як відправна точка для запуску алгоритму підбору архітектури за трьома сценаріями для кожного з проектів та подальшого порівняння отриманого варіанту з реалізованим.

Приклад списку комбінацій архітектур і тактик проектування, які відповідають даному вектору, складається з вхідних елементів, (розглянутий шаблон Клієнт-Сервер).

З області визначення задачі підбору архітектур, підбирається початкова конфігурація рішення, що складається з комбінації шаблону і тактик розробки, необхідних для задоволення вектора параметрів якості. З огляду на кількість модулів (205), запуск рішення за трьома сценаріями привів до трьох різних шаблонів архітектури з різним набором тактик. Другий крок верифікації методики пов'язаний з перевіркою інтервального розрахунку трудомісткості, яка застосовується в даній роботі і порівнянням результату з фактичним значенням, отриманому при аналізі вихідного коду. Для інтервального підрахунку трудомісткості по описуваній методиці був прийнятий до уваги існуючий шаблон архітектури проєкту і реалізовані тактики розробки. Для подібної конфігурації був проведений розрахунок інтервальної трудомісткості. Межі інтервалів взяті в межах експертної оцінки подібного типу модулів.

Основні кроки розрахунку для Проекту.

Крок № 1. Розрахунок базової вартості шаблону:

$$\tilde{Q}_б = \alpha \cdot EAF \cdot (n \cdot \tilde{K}_k)^{1.09}.$$

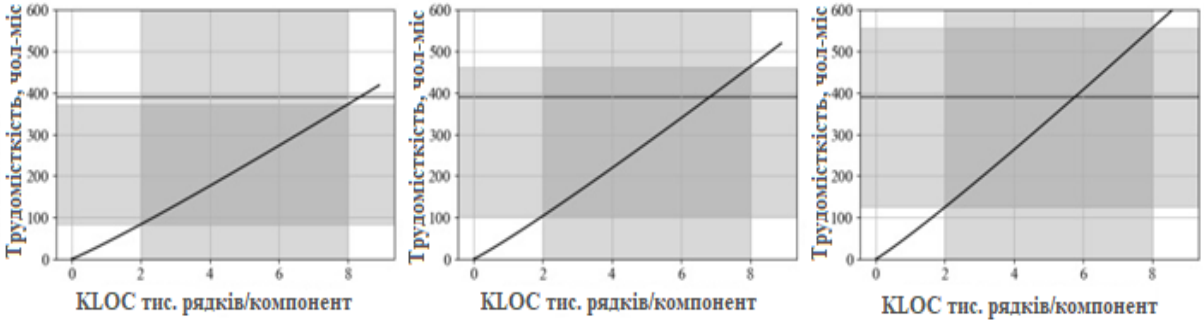
Значення коефіцієнта масштабу (B = 1,09) визначається усередненими показниками факторів.

Крок № 2. Визначення кількості модулів у проекті. Визначається за допомогою експертної оцінки, в даному випадку вибране значення збігається з реальною оцінкою. При кількості модулів, яка дорівнює кількості пакетів OSGi (враховуючи розширення), кількість модулів дорівнює 205.

Крок № 3. Визначення типів модифікації шаблону для реалізованих тактик проектування. Для списку тактик проектування Проекту типи модифікації шаблону архітектури мікросервісов вказані в таблиці 4.1. Залежно від величин інтервалів I_m і K_k, зміна загальної інтервальної трудомісткості рішення показана на рисунку 4.2.

Таблиця 4.1 – Типи модифікації і интервальная вартість

Шаблон/тактика	Тип модифікації	Інтервальна вартість
MVC + плагіни	-	$(7 \cdot \tilde{K}_k)^{1.09}$
Використання файлів конфігурації	Модифікація компонента	$(7 \cdot \tilde{I}_m \tilde{K}_k)^{1.07}$
Слабка зв'язаність	Модифікація компонента	$(7 \cdot \tilde{I}_m \tilde{K}_k)^{1.07}$
Аутентифікація	Додавання компонента в шаблон	$(\tilde{K}_k)^{1.07}$
Авторизація	Реалізація всередині компонента + додавання компонента в шаблон	$(\tilde{I}_m \tilde{K}_k)^{1.07} + (\tilde{K}_k)^{1.07}$



а) б) в)

Рисунрк 4.2 – Змінення інтервальної трудомісткості: а) мінімальна; б) усереднена; в) максимальна

4.2 Аналіз отриманих результатів

Аналіз результатів верифікації показує, що конфігурації архітектур, схожі з фактичними, були обрані при розрахунку максимального сценарію з тактиками розробки, що відповідають обраному вектору параметрів якості $\overline{\Pi} = [\Pi_i]$.

Також, порівнюючи отриманий результат з фактичною архітектурою проєкту, можна відзначити:

- реалізований варіант базового шаблону архітектури знаходиться серед запропонованих одним із сценаріїв методики базових шаблонів архітектури;

- збігається спрямованість запропонованих методикою і реалізованих тактик розробки. Дане спостереження верифікує вірне знаходження вектора параметрів якості для зазначеної системи;

- конкретна кількість і вид зазначених тактик варіюються в більшу сторону, що говорить про тенденції методики більш повно відповідати параметрам якості для розрахункової системи. Реальна архітектура проекту показує, що реалізація лише деяких тактик розробки із запропонованого списку в кінцевому підсумку є достатньою. Той факт, що присутність

зазначених тактик було отримано при виконанні максимального сценарію говорить про схильність саме даного сценарію розрахунку «перестраховуватися», і включати частково дублюючі функціонал тактики розробки.

ВИСНОВКИ

В результаті виконання магістерської атестаційної роботи було проведено огляд ключових особливостей систем IoT, який показав наявність найбільш важливих релевантних різним типам систем IoT параметрів якості. Зроблений аналіз процесу розробки програмної архітектури, в свою чергу, підтверджує можливість досягнення необхідних параметрів якості шляхом структурнопараметричному синтезу системи на основі необхідних характеристик.

Зазначені характеристики засновані на функціональних вимогах і типі системи IoT і можуть бути обрані через виявлені в роботі основні впливи, що діють на систему. Онтологічний опис процесу розробки програмної архітектури дозволив структурувати процес досягнення параметрів якості через застосування шаблонів програмних архітектур і супутніх тактик розробки, що залежать від необхідних параметрів якості системи.

Крім зазначених параметрів синтезу, в роботі була досліджена кількісна оцінка трудомісткості розробки комбінацій шаблонів і тактик шляхом застосування аналітичної методики оцінки вартості розробки на початковому етапі (COCOMO II).

Крім того, було виділено п'ять основних типів впливу тактик розробки на шаблони архітектури (типи зміни шаблонів тактикою), що дозволяє параметрично оцінити трудомісткість виходячи зі структури системи. Вищезазначені параметри лягли в основу методики вибору конфігурацій архітектури на основі рішення задачі комбінаторного синтезу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Токарев В.В. Разработка алгоритма мультиагентного управления группой мобильных «s-bot» / В. Н. Ткачев, В. В. Токарев, Г. И. Чурюмов // Реєстрація, зберігання і обробка даних. - 2019, Т. 21, № 1 – С.46-56.

2. Токарев В.В. Надширокосмугові технології в системах управління мобільними об’єктами / О. А. Серков, П. Є. Пустовойтов, І. В. Яковенко, Б. О. Лазуренко, Г. І. Чурюмов, В. В. Токарєв, Ванг Наннан // Сучасні інформаційні системи. - 2019, Т.3, №2 – С.22-27.

3. Volodymyr Tokariev. Ultra Wideband Signals in Control Systems of Unmanned Aerial Vehicles / Aleksandr Serkov, Valeri Kravets, Igor Yakovenko, Gennady Churyumov, Wang Nannan // The 10h IEEE International Conference on Dependable Systems, Services and Technologies, DESSERT’2019 5-7 June, 2019, Leeds, United Kingdom. - Pp.26 - 29.

4. Tokariev Volodymyr. Problem of self-organization of s-bot group movement in unorganized physical environment / Churymov Gennadiy, Tokariev Volodymyr, Tkachov Vitalii // Комп’ютерні та інформаційні системи і технології: тези доповідей третьої міжн. наук. - техн. конф. 23 - 24 квітня 2019 р. - Харків, Україна. - С.16-17.

5. 5.Volodymyr Tokariev. Method for Ensuring Survivability of Flying Ad-hoc Network Based on Structural and Functional Reconfiguration / Genadiy Churyumov, Vitalii Tkachov, Volodymyr Tokariev, Vladyslav Diachenko // Selected Papers of the XVIII International Scientific and Practical Conference "Information Technologies and Security" (ITS 2018) / Kyiv, Ukraine, November 27, 2018. – Pp. 64-76.

6. Volodymyr Tokariev. Method of Data Collection in Wireless Sensor Networks Using Flying Ad Hoc Network / Vitalii Tkachov, Volodymyr Tokariev, Yana Dukh, Vadym Volotka // 2018 5th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology, October 9-

12, 2018 Kharkiv, Ukraine. – Pp.197 - 201.

7. Tokariev,V. Scenario of Interaction of the Mobile Technical Objects in the Process of Transmission of Data Streams in Conditions of Impacting the Powerful Electromagnetic Field / G. Churyumov, V. Tokarev, V. Tkachov, S. Partyka // 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP). – 21-25 Aug. 2018. – Pp. 183-186.

8. Токарев В.В. Темпоральная модель адаптации интегрированной информационной системы путем реконфигурации логической структуры / О.Г. Лебедев, В.Н. Ткачев, В.В. Токарев, Г.И. Чурюмов // Комп'ютерні та інформаційні системи і технології: тези доповідей другої міжн. наук. - техн. конф. 18 - 19 квітня 2018 р. - Харків, Україна. - С.6-7.

9. Tokarev V.V. SHORTEST PATH BRIDGING METHOD FOR THE GROUP OF MOBILE TECHNICAL OBJECTS/ V.M. Tkachov, V.V. Tokarev, G.I. CHURYUMOV//СУЧАСНІ НАПРЯМИ РОЗВИТКУ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЗАСОБІВ УПРАВЛІННЯ: матеріали VIII - межд. наук. - техн. конф., 26 - 27 квітня 2018 р. - Харків, 2018р. - С.18.

10. Volodymyr V. Tokarev. Provision of Survivability of Reconfigurable Mobile System on Exposure to High-Power Electromagnetic Radiation / Igor V. Ruban, Genadiy I. Churyumov, Volodymyr V. Tokarev, Vitaliy M. Tkachov // Selected Papers of the XVII International Scientific and Practical Conference on Information Technologies and Security (ITS 2017). – CEUR Workshop Processing. – Kyiv, Ukraine, November 30, 2017. – Pp. 105-111.

11. СТВОРЕННЯ НАУКОВО-МЕТОДИЧНИХ ОСНОВ ЗАБЕЗПЕЧЕННЯ ЖИВУЧОСТІ МЕРЕЖЕВИХ СИСТЕМ ОБМІНУ ІНФОРМАЦІЄЮ В УМОВАХ ЗОВНІШНЬОГО ВПЛИВУ ПОТУЖНОГО НВЧ ВИПРОМІНЮВАННЯ // Г.И. Чурюмов, В.В. Токарев, И.В. Рубан, В.Н. Ткачев и др. // ЗВІТ ПРО НАУКОВО-ДОСЛІДНУ РОБОТУ за договором від 20.09.2017 р. № Ф76/109-2017 (заключний). № держреєстрації 0117U003916. ХИРЭ. - 116с.

12. Спосіб передачі цифрових даних мультикоптерною системою між сегментами розподіленої сенсорної мережі та базовою станцією [Текст] : пат. 118921 Україна: МПК 2017.01, H04W 64/00, H04W 84/18 (2009.01), G06F 17/40 (2006.01) / Ткачов В.М., Токарєв В.В., заявник та патентовласник Харківський національний університет радіоелектроніки. – u2017 04085; заяв. 24.04.2017; опубл. 28.08.2017, бюл. № 16. – 2017. – 5 с.

13. Токарев В.В. Мобильная подсистема «Мультикоптер-сенсорная сеть» в компьютерной системе хранения BIG DATA / В.О. Радченко, Д.А. Руденко, В.Н. Ткачов, В.В. Токарев // Системи управління, навігації та зв'язку - 2017. - №4(44). – С.102-105.

14. Токарев В.В. Проблема передачі даних типу BIG DATA у мобільній системі «МУЛЬТИКОПТЕР - СЕНСОРНА МЕРЕЖА» / В.М. Ткачов, В.В. Токарев, В.О. Радченко, В.О. Лебедєв // Системи управління, навігації та зв'язку - 2017. - №2(42). – С.154-157.

15. Токарєв В.В., Філіпчік А.А., Модель вибору pattern-архітектури для розробки систем IoT. Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення: тези доповідей 53 міжн. наук. інтер. конф., м. Тернопіль, 16 листопада 2020р. Тернопіль, С. 103-104.