Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-24М
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Татаров Д.Г.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Минайленко Р.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Татаров Д.Г. Дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи визначення контексту користувача на основі архітектури CoDA.

Метою розробки є дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA.

Об'єктом дослідження є процес визначення контексту користувача на основі архітектури CoDA.

Предметом дослідження є методи визначення контексту користувача на основі архітектури CoDA.

Методи дослідження базуються на методах теорії комп'ютерних мереж, теорії захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи визначення контексту користувача на основі архітектури CoDA.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Visual C#.

Ключові слова: комп'ютерні науки, контекст користувача, CoDA

ABSTRACT

Tatarov D.G. Research and software implementation of a user context determination system based on the CoDA architecture. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed, which is intended for a user context determination system based on the CoDA architecture.

The purpose of the development is the research and software implementation of a user context determination system based on the CoDA architecture.

The object of the research is the process of determining the user context based on the CoDA architecture.

The subject of the research is the methods of determining the user context based on the CoDA architecture.

The research methods are based on the methods of computer network theory, information protection theory, methods of mathematical statistics, and methods of software development.

The result of the work is a software implementation of a user context determination system based on the CoDA architecture.

In the process of working on the software model, an analysis of existing hardware and software tools was performed. All components of the developed software are fully described.

A user-friendly user interface has been developed. Instructions for working with the software are provided.

The program can be used on a PC with Windows 10/11.

The program was developed in Visual C#.

Keywords: computer science, user context, CoDA

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АРМ	–	автоматизоване робоче місце
ЕЦП	–	електронний цифровий підпис
ЗКЦ	–	засвідчуючий й ключовий центр
ЗЦ	–	засвідчуючий центр
КЦ	–	ключовий центр
ЛОМ	–	локальна обчислювальна мережа
СКЗІ	–	система контролю та захисту інформації
ЦР	–	центр реєстрації
ЦУМ	–	центр управління мережею
SOA	–	Service-Oriented Architecture
SOAP	–	Simple Object Access Protocol

ВСТУП

Актуальність теми. Gartner визначає CoDA. (Context delivery architecture, Архітектура доставки контексту) як архітектурний стиль, який спирається на стилі взаємодії та розбиття сервіс-орієнтовані архітектури (SOA) та архітектури, керованої подіями (EDA), і додає формальні механізми елементів програмного забезпечення, які відкривають та застосовують контекст користувача в реальному часі. CoDA. надає структуру архітекторам рішень, що дозволяє їм визначати та реалізовувати компоненти технології, інформації та процесів, які дозволяють службам використовувати контекстну інформацію для підвищення якості взаємодії з користувачем. Технології можуть включати контекстних брокерів, моніторів стану, датчиків, аналітичних движків та хмарних двигунів обробки транзакцій.

До того, як з'явився CoDA, існувала SOA (Service-Oriented Architecture), яка являє собою набір принципів та методологій для проектування та розробки програмного забезпечення у формі сумісних служб.

Реалізація SOA покладається на сітці неузгодженого програмного забезпечення, що є вільно пов'язаними одиницями функціональності, які не мають прямого зв'язку один з одним. Кожна служба керує конкретним та незалежним фрагментом функціональності та використовує визначені протоколи, які описують, як передавати інформацію між собою.

SOA покладається на сервісну орієнтацію як на основний принцип проектування. Якщо служба представляє собою простий інтерфейс, який вичерпує його основну складність, користувачі можуть отримати доступ до незалежних служб без знання реалізації платформи служби, включаючи основну операційну систему, мову програмування або апаратне забезпечення.

SOA допомагає підприємствам реагувати швидше та економічно ефективніше на мінливі ринкові умови шляхом заохочення повторного

використання на рівні сервісу, а не на мікро (кодовому) рівні.

CoDA або Архітектура доставки контексту забезпечує контекстні послуги, які з'єднуються в рамках SOA. В результаті постачальники послуг, постачальники послуг мережі та інтегратори програмного забезпечення можуть розвивати контекст.

Розглядаючи CoDA, як один з елементів SOA, повинен відмітити, що одним з важливих питань побудови архітектури SOA є її адміністрування й забезпечення безпеки.

При наявності сотень сервісів, готових до застосування, у користувача виникає потреба в централізованій системі адміністрування, за допомогою якої можна було б здійснювати їхній моніторинг і захист.

Існує безліч систем адміністрування SOA (на базі агентів і без них). Продукти SOAPStation компанії Actional, Management Foundation, Service-Level Manager і Exception Manager фірми AmberPoint, Gateway від Reactivity і Service Manager компанії SOA Software є найбільш популярними засобами, призначеними для адміністрування сервісів у рамках усього підприємства. Компанії Computer Associates і Hewlett-Packard зі своїми продуктами Unicenter і OpenView відповідно також пропонують програмні засоби для оперативного адміністрування й моніторингу SOA.

Адміністрування архітектури SOA означає як моніторинг працездатності окремих сервісів, так і реєстрацію й перевірку необхідних функцій відповідно до різних законодавчих актів, що регламентують правила й строки зберігання бухгалтерської звітності й інших ділових документів.

Програмні продукти по адмініструванню SOA також полегшують розподіл сервісів, автоматично відслідковуючи ненадійні з'єднання й проблеми взаємодії корпоративних мереж за допомогою засобів захвата повідомлень, по своїх можливостях набагато переважаючі традиційні аналізатори мережних протоколів. Залежно від потреб організації ви можете забезпечити безпеку архітектури SOA за допомогою звичайних продуктів управління на базі протоколу SOAP або

інших спеціальних продуктів. Програмні продукти, відповідальні за безпеку, такі, як XS40 компанії DataPower, Sentry і Xwall від ForumSystems, Gateway від Reactivity і XML Guardian від Sarvega, являють собою єдину точку доступу, через яку повинні проходити всі сервісні запити.

Безпеку цілком припустимо забезпечувати й на рівні сервера додатків, на якому розміщаються сервіси, але це є нездійсненним завданням, якщо ваш партнер по бізнесу буде взаємодіяти з вашою архітектурою SOA.

Зовнішні Web-сервіси або шлюз XML є більше гнучкими засобами для надання доступу партнерові (або клієнтові) до вашої SOA, значно полегшуюче життя розроблювачам додатків.

Якщо взяти до уваги той факт, що архітектура SOA не має чіткого набору поєднуваних систем, те немає нічого страшного в тому, щоб розділити функції забезпечення безпеки й адміністрування сервісів.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем визначення контексту користувача на основі архітектури CoDA.

– Дослідження системи визначення контексту користувача на основі архітектури CoDA.

– Програмна реалізація системи визначення контексту користувача на основі архітектури CoDA.

Об’єктом дослідження є процес визначення контексту користувача на основі архітектури CoDA.

Предметом дослідження є методи визначення контексту користувача на основі архітектури CoDA.

Методи дослідження базуються на методах теорії комп'ютерних мереж, теорії захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод визначення контексту користувача на основі архітектури CoDA.

– Розроблено вітчизняний продукт визначення контексту користувача на основі архітектури CoDA, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі визначення контексту користувача на основі архітектури CoDA.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2025 р.), основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Кожна команда безпеки стикається з одним і тим самим фундаментальним компромісом: швидко виявляти загрози або глибоко їх розуміти. Системи виявлення в режимі реального часу забезпечують мілісекундний час реагування, але можуть мати проблеми з контекстом, що призводить до сповіщень, які не враховують ширшу історію. Історичний аналіз забезпечує глибоке розуміння поведінки та контексту, але може надходити занадто пізно, щоб зупинити активні загрози. Сучасні архітектури SIEM-озер даних, що підтримують аналітику майже в режимі реального часу та історичну аналітику, надають можливості інженерам з виявлення. Якою має бути наша основа для визначення того, які правила куди йдуть? Розглянемо підозрілий вхід: аналіз у режимі реального часу негайно ідентифікує подію автентифікації на основі певних атрибутів, таких як місцезнаходження або привілейована команда, але не має контексту, щоб визначити, чи це відповідає нормальній поведінці користувача, чи скомпрометованому обліковому запису. Історичний аналіз може забезпечити цей контекст через поведінкові базові показники та порівняння груп однорангових користувачів, але на момент завершення цього аналізу активний зловмисник, можливо, вже досяг своїх цілей. Проблема стає ще гострішою, якщо врахувати вимоги до якості, викладені в нашій структурі для високоякісних правил SIEM. Ефективні правила повинні відстежувати поведінку з високою достовірністю, одночасно контролюючи відповідні тактики та методи для вашої конкретної моделі загрози. Системи реального часу чудово виявляють атомарну поведінку – окремі події, які є однозначно шкідливими або доброякісними, – але мають труднощі з контекстним аналізом, необхідним для виявлення поведінки та багатоетапної кореляції атак.

Швидкість без контексту створює шум; контекст без швидкості пропускає активні загрози. Ефективні програми безпеки розгортають обидва методи

стратегічно, використовуючи виявлення в режимі реального часу для високонадійних сценаріїв з високим рівнем впливу, а також використовуючи історичний аналіз для складного виявлення поведінки та багатоетапної кореляції атак.

1.2 Область застосування

Виявлення в режимі реального часу перевершує сценарії з трьома характеристиками: висока впевненість у логіці виявлення, негайний вплив на бізнес та чіткі шляхи виправлення. Коли ці характеристики узгоджуються, перевага в швидкості переважує контекстуальні обмеження. Ці сценарії зазвичай включають атомарні кореляції – окремі поведінкові ситуації, які об'єктивно є небезпечними або однозначно вказують на компрометацію.

Розуміння того, коли слід розгортати обробку в реальному часі, вимагає вивчення методів кореляції, які роблять виявлення ефективним. Як ми досліджували в нашому аналізі методів кореляції SIEM , атомарні кореляції відстежують окремі методи, такі як заплановані завдання або зміни політик, які можна оцінювати окремо. Ці атомарні моделі поведінки ідеально підходять для обробки в реальному часі, оскільки вони не потребують базових поведінкових показників або порівнянь груп однолітків для підтвердження їхньої значущості.

Перш ніж розглядати конкретні випадки використання, розгляньте цей лакмусовий папірець: якщо ви можете однозначно класифікувати подію як шкідливу, не знаючи нічого про історичну поведінку користувача, стандартні бізнес-моделі чи організаційний контекст, вона, ймовірно, є сильним кандидатом для обробки в режимі реального часу. Логіка виявлення має бути бінарною – поведінка або прийнятна, або ні, з мінімальною сірою зоною, яка потребує додаткового дослідження.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Сервіс-орієнтована архітектура (SOA) означає для різних людей різне. З погляду керівників бізнесу, вона надає набір бізнес-сервісів, які можна запропонувати клієнтам і партнерам. Для IT-архітектора SOA є архітектурним стилем і парадигмою, що дозволяє створювати модульні й слабозв'язані сервіси, які можна становити й організовувати в бізнес-процеси, які представляють діюче підприємство.

Для побудови працюючої SOA-системи необхідні гарна концептуалізація й визначення тих бізнес-сервісів, які складуть цю систему. Архітектор повинен вивчити безліч факторів, щоб визначити, які сервіси моделювати й будувати. Основними серед цих факторів є наступні характеристики сервісу:

- Можливість настроювання сервісу під потреби бізнесу.
- Можливість його компонування з іншими сервісами.
- Його потенціал для повторного використання.
- Його технічна реалізуємість.

Це деякі з базових концепцій, на підставі яких моделюються й проектуються сервіси в заснованій на SOA-архітектурі.

Сервіс у системі SOA відповідає чіткому життєвому циклу. Для кожної фази життєвого циклу SOA потрібні інструменти проектування й розробки, а також сполучні програмні засоби для виконання дій у конкретній фазі. У цій статті представлені ключові інструменти й продукти від компанії IBM, що дозволяють запровадити в життя конкретні фази життєвого циклу SOA.

Життєвий цикл SOA

Життєвий цикл сервісу починається з моделювання вимог до нього, потім відбувається проектування й розробка сервісу, після чого випливають комбінування й аранжування сервісу. Потім сервіс розгортається й виконується. Різні сполучні продукти часу виконання системи не тільки надають середовище виконання для сервісу й реалізованих їм бізнес-процесів, але й надають дають механізм контролю й керування сервісом.

– Концепція SOA Foundation задає логічні кроки по реалізації системи SOA. Компанія IBM ретельно відібрала з великого портфеля програмного забезпечення IBM програмні інструменти й продукти, що допомагають у реалізації всіх чотирьох фаз життєвого циклу SOA (для запам'ятовування їхньої послідовності використовують абревіатуру MADM):

1. Моделювання (Model).
2. Складання (Assemble).
3. Розгортання (Deploy).
4. Керування (Manage).

Принципи керівництва й передові методики визначають загальну стратегію, що управляє всіма фазами життєвого циклу.

На високому рівні узагальнення розташовуються загальні дії, що відбуваються в кожній фазі:

1. **Моделювання**. Збираються вимоги, бізнес-процеси в цілому моделюються, аналізуються, проектуються, а потім оптимізуються для формування майбутніх стандартних бізнес-процесів підприємства.

2. **Складання**. Реалізуються сервіси. Потім здійснюється складання впроваджених сервісів: їх знаходять, компонують і аранжують для реалізації корпоративних бізнес-процесів, які перевіряють на предмет відповідності як функціональним, так і іншим вимогам.

3. **Розгортання**. Скомпоновані бізнес-процеси розгортають у діючому середовищі виконання.

4. **Керування**. Проводять моніторинг і аналіз сервісів і бізнес-процесів, виконуваних під час виконання, для забезпечення їхньої безперебійної роботи. Також виміряються такі IT-показники, як безпека, продуктивність і експлуатаційна готовність. Надалі сервіси й бізнес-процеси також контролюють для забезпечення сумісності з бізнес-метриками й угодами про рівень сервісу (SLA), яким вони повинні відповідати.

5. **Керівництво (governance) і передові методики**. Це головний алгоритм, що контролює всі аспекти в кожній фазі життєвого циклу MADM. У зв'язку з їх різнобічним позитивним впливом на розробку SOA керування й передові методики розглядаються як невід'ємні компоненти системи нормативів, обов'язкових для будь-якої серйозної реалізації SOA.

Далі в даному розділі розглядаються спеціалізовані інструменти й продукти від IBM, що підтримують кожну з фаз життєвого циклу SOA.

Інструменти й продукти для життєвого циклу SOA

У цьому розділі представлені продукти IBM, які можна використовувати для виконання операцій у кожній фазі життєвого циклу SOA.

Етап моделювання

– WebSphere® Business Modeler

– Rational® RequisitePro®

У фазі моделювання бізнес моделюють навколо ключових процесів, що забезпечують функціонування підприємства. Спочатку моделюють поточні бізнес-процеси. Ця фаза допомагає задокументувати поточний стан бізнесу й навчити працюючий і знову прийнятий персонал тому, як функціонує компанія. Далі наявні змодельовані бізнес-процеси можна з’імітувати для виявлення можливих проблемних місць в існуючих операціях. Оптимізація бізнес-процесів допомагає виправити знайдені проблемні місця й сформувати більше гнучкі й продуктивні бізнес-процеси, а також усунути точки ризику й запобігти можливому невдоволенню користувачів.

Результатом цієї оптимізації є моделі для цільових бізнес-процесів. У свою чергу, їх імітують математично з різними вхідними параметрами, щоб оцінити можливі результати. Можна проводити їхню верифікацію на предмет відповідності бізнес-вимогам, як функціональним, так і іншим. При імітації процесів підприємство може прогнозувати, як вони будуть функціонувати при різних сценаріях; наприклад, при неоднаковому навантаженні в різні сезони. Тепер підприємство не зштовхнеться з неприємними сюрпризами, а це дозволяє перейти від стратегії реагування до прогнозування результатів різних бізнес-подій. Це значний крок уперед до становлення динамічного бізнесу.

Програма моделювання бізнесу WebSphere Business Modeler

Підприємству необхідний ефективний інструмент для моделювання бізнес-процесів, простий у використанні й доступний не тільки підготовленим IT-фахівцям. Цей інструмент повинен переважно використовувати звичайний бізнес-словник, а не мудру IT-термінологію. Повинна бути можливість додавати до елементів процесів бізнес-параметри у формі ключових показників ефективності (KPI), ресурсів, показників витрат вартістю й продуктивності так, щоб їх можна було виміряти в наступних фазах життєвого циклу SOA. WebSphere Business Modeler надає функціональні можливості такого роду. Це потужний, ґрунтовно продуманий, стабільний і простий у використанні продукт, якому можна застосовувати для моделювання бізнес-процесів.

WebSphere Business Modeler:

– Може бути корисний бізнес-аналітикам.

– Дозволяє фіксувати структуру бізнес-процесів.

– Забезпечує наочне подання бізнес -процесів, організаційної структури, ресурсів і показників продуктивності.

– Надає інструмент імітаційного моделювання для аналізу й тестування процесів.

– Дозволяє експортувати бізнес-моделі для використання під час операцій у наступних фазах життєвого циклу SOA.

– Може видавати вичерпну інформацію про економію витрат, часу й ресурсів.

– Дозволяє оптимізувати бізнес-процеси шляхом виявлення вузьких місць і диспропорції завантаження до фактичного внесення змін у процеси й сервіси.

Це далеко не повний перелік функцій WebSphere Business Modeler, але вже він показує, як цей інструмент можна використовувати у фазі моделювання життєвого циклу SOA. На рисунку 2.1 представлений скріншот моделі бізнес-процесів, створеної за допомогою WebSphere Business Modeler.

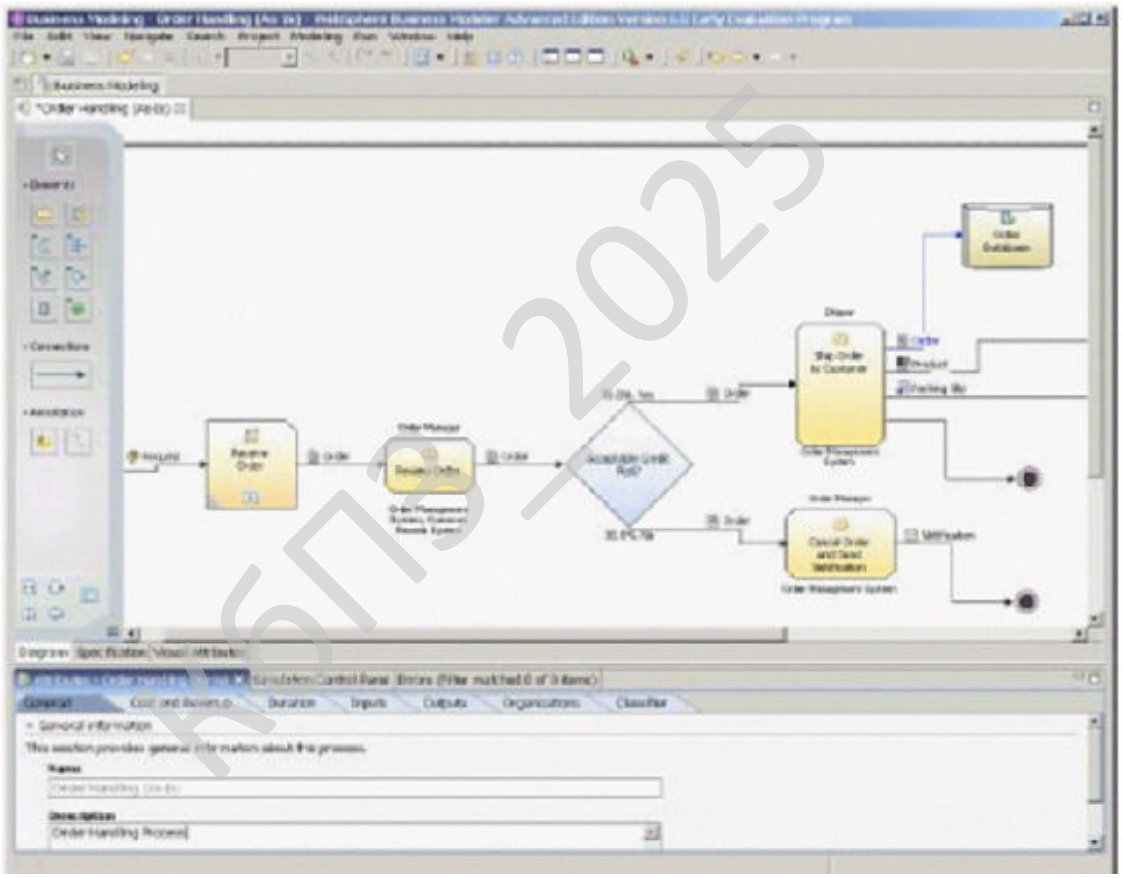

Рисунок 2.1 – Приклад моделі бізнес-процесів в WebSphere Business Modeler

Rational RequisitePro

Збір інформації про вимоги є ще одним важливим завданням, що вирішується у фазі моделювання. Рішення Rational RequisitePro допомагає команді розроблювачів створювати й спільно використовувати власні вимоги, застосовуючи знайомі методи документування, поряд з можливостями бази даних, такими як відслідковуємість вимог і аналіз впливів. Результатом є поліпшення взаємодії й керування вимогами й підвищення ймовірності своєчасного завершення проектів у рамках бюджету й відповідно до проектних показників. Серед ключових функцій Rational RequisitePro:

– Удосконалена інтеграція з Microsoft® Word.

– Потужна інфраструктура бази даних.

– Що налаштовуються, фільтруємі параметри вимог.

– Глибокий аналіз відслідковуємості й охвату.

– Докладний аналіз змін і /впливу з контрольними слідами й повідомленням по електронній пошті.

– Створення й порівняння базових параметрів проекту.

– Web-доступ для розосереджених колективів.

– Гнучкі опції звітності.

– Функції імпорту, що налаштовуються.

– Інтеграція з різноманітними інструментами пакета Rational Software Development Platform.

– Обумовлені користувачем види вимог.

– Конфігуруємі шаблони й проекту й документів.

На рисунку 2.2 представлений скріншот з вимогами, зібраними з використанням RequisitePro.

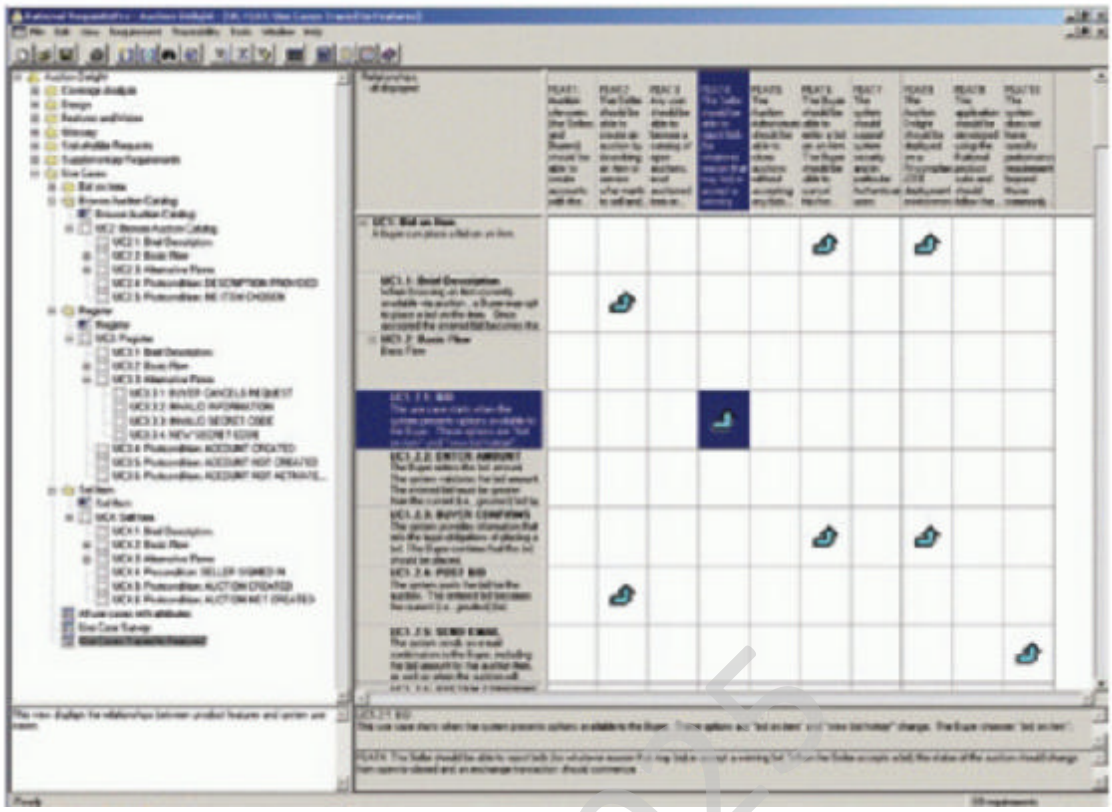

Рисунок 2.2 – Застосування інструмента RequisitePro для збору потреб

Етап складання:

– Rational Software Architect.

– Rational Application Developer.

– Rational Data Architect.

– WebSphere Integration Developer.

Етап складання складається із трьох основних процедур:

– Проектування сервісів.

– Побудова сервісів.

– Інтеграція сервісів.

Якщо ви подумки перенесетеся на пару років назад в розробці програмного забезпечення, то, імовірно, як і я, пригадаєте моделювання процесів в Visio і згадаєте про існуючий тоді розрив між моделюванням і проектуванням.

Артефакти, створені в Visio, не були сумісні з наступними інструментами для проектування й розробки компонентів і сервісів.

WebSphere Business Modeler у цьому змісті є ідеальним рішенням. Моделі, спроектовані за допомогою цього інструмента, можна експортувати як мінімум у трьох форматах: Business Process Execution Language (BPEL), Unified Modeling Language (UML) 2.0 і XML Schema Definition (XSD). Якщо BPEL використовується для аранжування й комбінування сервісів, то сумісні з UML 2.0 вихідні формати застосовують для проектування й реалізації сервісів за допомогою таких технологій, як Java™ 2 Platform, Enterprise Edition (J2EE). XSD-визначення бізнес-одиниць, знайдені на підставі входів і виходів кроків процесу, можуть бути відмінною відправною точкою для логічної моделі даних, а також для інформаційної моделі.

Елементи бізнес-процесу, тобто дії, що входять у цей процес, можна реалізовувати або як бізнес-сервіси, або як компоненти багаторазового використання. У кожному разі процес необхідно спроектувати й реалізувати, наприклад, використовуючи технологію J2EE. Потім реалізований сервіс стає доступним для складання й реалізації бізнес-процесу, що потім запускають у середовищі виконання.

Проектування, специфікація й побудова сервісу

Rational Software Architect і Rational Data Architect використовуються для проектування сервісів, їхніх елементів даних і повідомлень. Rational Application Developer застосовується для реалізації сервісів з використанням технології J2EE.

Rational Software Architect

Rational Software Architect – це інтегрований інструмент проектування й розробки, що використовує принцип керованої моделями розробки на базі UML. Нові ресурси проектування SOA у версії 6.0.1 спрощують трансформацію бізнес-процесів у сервіс-орієнтовані додатки. Повна інтеграція між WebSphere Business Modeler і Rational Software Architect дозволяє архітекторам програмного забезпечення наочно представити моделі бізнес-процесів у поданні UML 2.0,

рятуючи архітекторів від необхідності вручну імпортувати моделі процесів. Така інтеграція також дозволяє напряму зіставити моделі процесору з моделями проекту. Моделі процесу в WebSphere Business Modeler можна експортувати у форматі UML 2.0 і імпортувати в Rational Software Architect, де вони стають діаграмами дій.

UML 2.0 Profile для Software Services (підтримуваний інтеграцією Rational Software Architect-WebSphere Business Modeler) розширює можливості UML і надає загальна мова для опису сервісів, дозволяючи архітекторам моделювати й генерувати Web-сервіси. Цей профіль можна застосувати до будь-який нового або вже існуючої моделі, він також додає нові концепти й позначення, що ставляться до аналізу й проектування SOA. Профіль дає архітекторам можливість планувати сервіси з використанням логічної класифікації, що описує весь корпоративний портфель сервісів, а також розробляти специфікації сервісу – як структурні, так і поведінкові, – виступаючі в якості договори між клієнтами сервісу й конструкторами. Таким чином, Rational Software Architect дозволяє проектувати й описувати сервіси.

Rational Application Developer

Коли сервіси спроектовані й описані, їх потрібно сконструювати й реалізувати. Rational Application Developer можна використовувати під час конструювання й впровадження сервісів. Цей інструмент являє собою сервіс-орієнтоване середовище розробки, що автоматизує багато хто із завдань, звичайно виконуваних при конструюванні й використанні Web-сервісів. Розроблювачі можуть сфокусуватися на написанні логічного коду бізнесу, поклавши на Rational Application Developer автоматичну обробку всієї інформації з файлу Web Service Description Language (WSDL) і генерації коду для одержання клієнта, а також перевірки відповідності вимогам Web Services-Interoperability (WS-I).

Можна також використовувати Rational Application Developer для створення, побудови, застосування, тестування, розміщення й публікації Web-

сервісів – або з нуля, або модифікуючи існуючі додатки для відповідності WS-I. Rational Application Developer спрощує процес створення Web-сервісів по спадному принципі, заснованому на специфікаціях потреб і UML-Моделях. Цей інструмент дозволяє навіть створювати каркаси Enterprise JavaBeans (EJB), які можна використовувати для реалізації Web-сервісів. Rational Application Developer також допомагає при створенні Web-сервісу по висхідному принципі, від існуючих класів Java або EJB. Він надає майстер, що автоматично генерує WSDL-визначення з існуючих класів Java або EJB. Rational Application Developer також містить інструменти J2EE Connector Architecture, що дозволяють легко інтегрувати існуючі корпоративні інформаційні системи (EIS), такі як IBM CICS® або IBM IMS™, в SOA-рішення. Використовуючи ці інструменти, розроблювачі можуть швидко створити Web-сервіс із існуючої EIS-операції.

Rational Data Architect

Rational Data Architect – це корпоративний інструмент для моделювання даних і проектування інтеграційних рішень; він допомагає архітекторам даних проектувати реляційні й інтегровані бази даних, розбиратися в ресурсах даних і відносинах між ними, а також упорядковувати проекти баз даних. Rational Data Architect використовує XSD-експорт із WebSphere Business Modeler для моделювання схем реляційної бази даних (RDB). Схеми RDB допомагають при створенні відображення даних RDB. Моделювання даних, виконуване в Rational Data Architect, можна далі експортувати в UML.

Rational Data Architect використовують для створення логічної моделі й відображення даних, які можна експортувати з Rational Data Architect і імпортувати в Rational Software Architect у форматі UML. Це розширює моделі бізнес-процесу, експортовані з WebSphere Business Modeler і імпортовані в Rational Software Architect у форматі UML 2.0. Rational Software Architect також застосовується для проектування й специфікації сервісів з використанням RSA-плагіна UML 2.0 Profile для Software Services. Потім спроектовані й описані сервіси конструюють в Rational Application Developer, використовуючи технологію J2EE.

Компонування й аранжування сервісів

Після реалізації сервісів їх використовують для компонування й аранжування бізнесу-процесу. Моделі бізнес-процесів, створені в WebSphere Business Modeler, необхідно скорегувати й зібрати за допомогою BPEL. У фазі складання існуючі й новоспоруджені сервіси використовуються для реалізації всіх дій бізнесу-процесу.

WebSphere Integration Developer

IBM WebSphere Integration Developer надає розроблювачам інтеграції візуальні інструменти розробки для специфікації, тестування й розгортання виконуваних бізнес-процесів. WebSphere Integration Developer допомагає інтегрувати Web-сервіси, корпоративні додатки, завдання користувачів і інші компоненти сервісу в ефективні, засновані на SOA бізнесу-рішення.

Моделі бізнес-процесів з WebSphere Business Modeler експортують у вигляді BPEL; далі їх можна імпортувати в WebSphere Integration Developer. Потім WebSphere Integration Developer представляє бізнес-процес в WS-BPEL. Кожна процедура в потоці BPEL реалізується одним із двох можливих способів:

– Як сервіс, що уже розроблений (наприклад, сконструйований раніше за допомогою Rational Application Developer) або як зовнішній сервіс (наприклад, сервіси, надавані партнерами по взаємодії в режимі " бізнес-бізнес").

– Як компонент сервісу, що використовує конструкції й функції Service Component Architecture (SCA), підтримувані WebSphere Integration Developer, або як розширення IBM для BPEL.

WebSphere Integration Developer також допомагає розроблювачам компонувати бізнес-процеси шляхом об'єднання простих і складених сервісів у бізнес-процеси. Далі ці процеси можна запускати в середовищі виконання.

Етап розгортання:

– WebSphere Application Server.

– WebSphere Process Server.

Реалізовані бізнес-процеси тепер готові до розгортання в середовищі виконання, що підтримує виконання динамічних бізнес-процесів. Середовище виконання повинна бути заснована на відкритих стандартах, щоб одні сервіси могли безперешкодно викликати інші. Середовище виконання повинна підтримувати як мінімум три базові функції, наведені нижче:

– Трансляція протоколів між викликами різних сервісів.

– Маршрутизація між відповідними постачальниками сервісів.

– Медіація для забезпечення таких функцій, як безпека, аудит, протоколювання й т.д.

Артефакти реалізації (наприклад, компоненти J2EE, класи, EJB), які реалізують сервіси, також мають потребу в стабільному, масштабованому, високопродуктивному сервері додатків для забезпечення очікуваних рівнів сервісу і якості.

WebSphere Application Server

WebSphere Application Server виконує в інфраструктурі SOA дві ролі. Це безпечне, масштабоване, високопродуктивне й відказостійке хостингове середовище для основних бізнес-сервісів SOA – у першу чергу тих, які реалізовані з використанням EJB. Такі сервіси можна експонувати за допомогою WSDL і інтегрувати через стандартні протоколи й кодування Web-сервісів. Їх також можна інтегрувати більше сильнозв’язаним образом через зв'язування Remote Method Invocation/ Inter-ORB Protocol (RMI/IIOP). WebSphere Application Server також виступає як базова платформа виконання для WebSphere Portal, WebSphere Process Server, WebSphere Enterprise Service Bus, а також безлічі інших пропозицій у портфелі IBM. Сконструйовані сервіси й побудовані для них компоненти J2EE можна запускати на сервері додатків WebSphere Application Server.

WebSphere Process Server

Хоча WebSphere Application Server надає середовище виконання сервера додатків для виконання сервісів, воно не забезпечує сполучних засобів для

виконання динамічних бізнес-процесів. Він також не обробляє активізацію сервісів, перетворення протоколів, медіацію й маршрутизацію запитів. WebSphere Process Server, Version 6.0.1 виконує бізнес-процеси захищеним образом, узгоджено дотриманням транзакційної цілісності. Він підтримує як для засновані на BPEL потоки процесів, так і кінцеві бізнес-автомати. WebSphere Process Server також підтримує інтеграцію бізнес-правил у вибір процесів і сервісів. Цей сервер процесів є першим продуктом IBM із прямою підтримкою моделі програмування Service Component Architecture (SCA) SOA. WebSphere Process Server також інтегрується з WebSphere Portal Server для підтримки завдань користувача в бізнес-процесах.

Бізнес-процеси, зібрані в WebSphere Integration Developer, можна розгорнути в середовищі виконання WebSphere Process Server. WebSphere Process Server побудований на таких стандартах, як BPEL, Web-сервіси, Java Message Service (JMS), XML і багатьох інших – це забезпечує максимальну сумісність і гнучкість будь-яких сервісів, що є частиною системи SOA. WebSphere Process Server містить WebSphere Enterprise Service Bus (ESB), що служить проміжною ланкою між різнорідними ресурсами, максимізуючи повторне використання ресурсів, незалежно від їхнього місцезнаходження, постачальника, платформи й від того, чи є вони додатками власної розробки або комерційних додатків.

Разом з WebSphere Process Server і WebSphere ESB, WebSphere Application Server утворить стійке, масштабоване, високопродуктивне середовище розгортання для виконання критично важливих бізнес-процесів і бізнес-додатків. У той час як WebSphere Application Server надає середовище виконання J2EE для запуску компонентів J2EE (необхідних для побудови корпоративних додатків) і підтримку документів WSDL (для експонування бізнес-сервісів як Сервіс-Web-сервісів), WebSphere Process Server надає середовище для аранжування й комбінування складання бізнес-сервісів і запуску бізнес-процесів. WebSphere Process Server містить WebSphere ESB, що реалізує функції й можливості

корпоративної шини даних. І WebSphere ESB, і WebSphere Process Server працюють поверх WebSphere Application Server.

Етап керування:

– WebSphere Business Monitor.

– Tivoli® Composite Application Manager для SOA.

– Tivoli Composite Application Manager для Response Time Tracking.

– Tivoli Federated Identity Manager.

У фазі розгортання бізнес-процеси й додаток^-додатки-бізнес-додатки розміщають у відповідному середовищі виконання SOA. У фазі керування команді розроблювачів необхідно стежити за тим, щоб додатка, сервіси, бізнес-процеси, а також інфраструктура SOA і сполучні додатки, працювали безвідмовно й ефективно. Конструкції й артефакти системи SOA необхідно контролювати й перевіряти на предмет відповідності бізнес-метрикам і відповідним угодам SLA. Також необхідно вимірювати окупність інвестицій (ROI) для бізнесу. Таким чином, фаза керування призначена не тільки для керування корпоративної SOA (сервіси, процеси й інфраструктура), але й для контролю й виміру її ефективності й продуктивності.

У фазі керування можна виділити як мінімум чотири основних ділянки, що мають безпосереднє відношення до системи SOA:

– Керування бізнес-процесами.

– Керування рівнем сервісу.

– Керування виконанням транзакцій.

– Керування безпекою сервісів.

У даному розділі всі ці дії й відповідні їм інструменти розглядаються окремо. Ми приводимо далеко не повний перелік всіх продуктів, які застосовуються для проведення всіх процедур у фазі керування SOA. Однак за допомогою описаних тут продуктів можна здійснити більшість запитуваних дій у типовій фазі керування життєвого циклу SOA.

Керування бізнес-процесами – WebSphere Business Monitor

За часом виконання WebSphere Business Monitor здійснює моніторинг бізнес-процесів у реальному часі, забезпечуючи наочне відображення їхнього статусу. WebSphere Business Monitor інтегрується з WebSphere Business Modeler (застосовуваним при моделюванні), так що бізнес-параметри (наприклад, витрати, доходи, продуктивність) бізнес-процесів, змодельованих в WebSphere Business Modeler можна було відслідковувати в WebSphere Business Monitor і перевіряти на сумісність. Для цілей керування й виміру бізнес-процесів в WebSphere Business Monitor передбачені наступні можливості:

– контроль бізнес-процесів у реальному часі й надання інформації через панелі керування бізнесом, які надалі можна настроїти для різних користувачів (керівника бізнесу, бізнес-аналітика, адміністратора IT-систем). Панелі керування бізнесом надають у вигляді порівняльних таблиць ключові показники ефективності, такі як витрати, доходи, час і ресурси;

– оповіщення основних користувачів, що полегшує постійне поліпшення бізнес-процесів і допомагає в керуванні підприємством шляхом виявлення відхилень для наступного усунення;

– інструменти аналізу даних для проведення різних вимірів і пошуку в інформації про процеси закономірностей, які допоможуть підвищити їхню ефективність;

– підтримка швидкого реагування на критичні ситуації для задоволення запитів споживачів;

– підтримка постійного моніторингу й перевірки систем для дотримання встановлених рівнів дефектів і інших показників якості сервісу;

– збір у реальному часі даних про виконувані процеси для уточнення майбутніх моделей процесів.

– повернення даних в WebSphere Business Modeler для імітаційного моделювання процесів, що допомагає оптимізувати майбутні реалізації процесу.

Керування на рівні сервісів: Tivoli Composite Application Manager для SOA

Поява в багаторівневій архітектурі додатків рівня абстракції сервісів виникає потреба в механізмі для моніторингу, керування й виміри сервісів, які є основними конструкціями SOA. IBM Tivoli Composite Application Manager для рішень SOA (ITCAM для SOA) забезпечує відстеження Web-сервісів на всіх рівнях для виявлення й усунення несправностей, які можуть виникнути при виклику й виконанні сервісів.

ITCAM для SOA створений як комплексне управлінське рішення для підприємств, що розгортають додатки SOA, засновані на Web-сервісах. ITCAM для SOA Version 6.0 може виявляти, відслідковувати, діагностувати й контролювати Web-сервіси, реалізовані з використанням SOAP/HTTP, SOAP/JMS. Він допомагає ідентифікувати й розв'язувати проблеми, що виникають при розміщенні сервісів. Це здійснюється шляхом поглибленого аналізу інформації починаючи із сервісів аж до компонентів додатків і IT-ресурсів, для виявлення джерела несправності або проблемної зони. Передбачено убудовані оповіщення, повідомлення й послідовності дій для автоматизації керування сервісами під час виконання. Показники продуктивності й оповіщення сервісу також доступні в якості портлетів, які можна легко інтегрувати в корпоративні портали.

Керування виконанням транзакцій: Tivoli Composite Application Manager for Response Time Tracking

IBM Tivoli Composite Application Manager for Response Time Tracking (ITCAM для RTT) забезпечує наскрізне відстеження транзакцій для швидкого виявлення й локалізації несправностей. Цей продукт дозволяє вимірювати час відгуку розподілених транзакцій за допомогою синтетичних і реальних кінцевих методик виміру. ITCAM для RTT дозволяє простежувати шляхи транзакції користувача в бізнес-інфраструктурі шляхом деталізації кожного кроку, здійсненого при транзакції під час проходження через різні системи. На кожному

кроці також виміряється внесок кожного компонента транзакції в підсумковий час відгуку. Також як і ITCAM for SOA, ITCAM for RTT надає звіти про транзакції у формі, яку можна представити у вигляді портлетів у корпоративному порталі.

Керування безпекою сервісів: Tivoli Federated Identity Manager

Для сучасних організацій принцип "працювати краще" означає "працювати разом", і це ставить нові проблеми в IT-безпеки. SOA уможливила участь підприємств у такий екосистемі, де бізнес може бути як постачальником сервісу, так і його споживачем, створюючи в такий спосіб умови для формування заснованої на SOA мережі створення цінності. Участь в екосистемі руйнує традиційні границі корпоративних додатків. Бізнес-транзакція може запускати послідовність сервісів, кожний з яких може бути від іншого провайдеру. Вихід транзакцій за межі традиційних корпоративних границь, породжує додаткові труднощі відносно безпеки скомпонованих сервісів.

Безпека здобуває ще більшу важливість у зв'язку з безліччю нормативних вимог, прийнятих промисловими й споживчими групами, урядами й самими підприємствами. Надійна об'єднана система ідентифікації й керування безпекою є обов'язковою для успіху SOA, особливо в ситуаціях, коли корпоративні границі розширюються в більшу мережну прикордонну систему.

IBM Tivoli Federated Identity Manager (ITFIM) підтримує централізоване керування ідентифікаційною інформацією й полегшує користувачам довірчий доступ до інформації й сервісів. ITFIM забезпечує об'єднане, на основі заданих правил керування безпекою для інтегрованих Web-сервісів. Обмін достовірною ідентифікаційною інформацією й політиками полегшує переміщення користувачів по інтегрованих вузлах, що бере участь в екосистемі SOA. ITFIM підтримує відкриті стандарти й специфікації, включаючи Liberty, SAML, WS-Federation, WS-Security і WS-Trust, які спрощують інтегрування партнерських сервісів.

Керівництво й передові методики: WebSphere Service Registry і Repository

Керівництво SOA будується поверх середовища вибору рішень і керування й забезпечує керівництво (governance) і передові методики для ефективного керівництва діями в кожній фазі життєвого циклу SOA. Якщо розглянути життєвий цикл сервісу ще докладніше, ніж ми робили дотепер, ми побачимо, що в ньому можна виділити наступні ділянки:

- ідентифікація сервісу;
- визначення сервісу;
- розгортання сервісу;
- міграція сервісу;
- керування версіями сервісу;
- приналежність сервісу;
- моніторинг сервісу;
- тестування сервісу;
- безпека сервісу;
- політики сервісу.

Необхідно, щоб дії на кожній із цих ділянок ретельно управлялися середовищем, що встановлювала б упорядкований, заснований на процесі підхід до корпоративного SOA- трансформації. Як програмні засоби не можуть служити інструментом державного керування, так і багато аспектів керівництва SOA не підтримуються продуктами IBM і інших виробників. Однак певні аспекти керівництва й дотримання передових методик можна ефективно автоматизувати за допомогою підходящих програмних продуктів.

Одним з таких продуктів є WebSphere Service Registry and Repository . Він використовується для зберігання метаданих про зберігання, доступ і керування інформацією, що ставиться до сервісів. Він зберігає інформацію усередині організації або в зовнішніх системах про сервіси, які вже використовуються, плануються до використання або будуть коли-або відкриті для користувачів

сервісів. WebSphere Service Registry and Repository створює основу для керівництва сервісами протягом усього життєвого циклу. Він забезпечує інтероперабельність між сервісами за допомогою реєстру й сховища, побудованих на відкритих стандартах. Підтримка відкритих стандартів забезпечує можливість інтеграції й обмін інформацією про сервіси між WebSphere Service Registry and Repository і іншими заснованими на стандартах реєстрами й сховищами.

SOA-керівництво займається не тільки метаданими сервісів. Як уже говорилося, воно займається всіма аспектами життєвого циклу сервісу й об'єднання людей, процесів і технологій для взаємозалежної роботи. Таким чином, продукти, що підтримують інші аспекти життєвого циклу сервісів, також допомагають організовувати керівництво сервісами шляхом інтеграції трьох основ бізнесу – його кадрів, процесів, інструментів і технологій. Прикладом цього є інструмент для моніторингу й керування сервісом: WebSphere Business Monitor і ITCAM для SOA. Лінійки IBM Tivoli і Rational пропонують цілий ряд інструментів і продуктів, що допомагають у керівництві на фазах проектування й розробки сервісу.

Об'єднання інструментів

Отже, от як проходить робота над сервісом SOA з погляду інструментів і життєвого циклу:

– Виконується збір і збереження вимог за допомогою Rational RequisitePro.

– Виробляється моделювання бізнес-процесів з використанням WebSphere Business Modeler.

– Результати моделювання використовуються в якості вхідних даних в Rational Software Architect для проектування й специфікації сервісу.

– Вихідні дані також використовуються в WebSphere Business Monitor для виявлення бізнес-метрик, які необхідно вимірювати й контролювати під час виконання.

– Сервіси, спроектовані й задані з використанням Rational Software Architect, потім реалізуються в Rational Application Developer.

– Результати моделювання також використовують для подання бізнес-процесу в якості BPEL в WebSphere Integration Developer.

– Розроблені й експоновані сервіси потім використовують для реалізації бізнес-процесів в WebSphere Integration Developer.

– Реалізовані сервіси й конструктивні блоки компонентів J2EE розміщають на сервері WebSphere Application Server.

– Бізнес-процеси, скомпоновані в WebSphere Integration Developer, розміщають на сервері WebSphere Process Server. Сервер процесів використовує WebSphere ESB для медіації й маршрутизації сервісу.

– Поточний бізнес-процесами управляють і контролюють їх за допомогою WebSphere Business Modeler.

– Самими сервісами управляють і контролюють їх за допомогою ITCAM для SOA.

– Безпекою сервісів управляють за допомогою Tivoli Federated Identity Manager.

– Розподілені комплексні транзакції відслідковують на всьому шляху через стек IT-продуктів за допомогою ITCAM для Response Time Tracking.

Кроки 1 і 2 вдносяться до фази керування, кроки з 3 по 7 – до фази складання, кроки 8 і 9 – до фази розгортання, а кроки з 10 по 13 є частиною фази керування.

Дані реального часу, зібрані з WebSphere Business Monitor, потім знову використовуються для імітаційного моделювання й оптимізації процесів в WebSphere Business Modeler, і весь процес повторюється заново з нової фази моделювання. Удосконалення сервісу знову моделюють, результати нової фази моделювання вводять у наступну фазу складання, після чого оптимізований сервіс заново розгортають і переводять у режим керування. У процесі оптимізації використовують інформацію, при моніторингу процесу й виявленні проблемних місць.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об'єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних додатків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські додатки Windows, веб-служби XML, розподілені компоненти, додатки типу " сервер-клієнт", додатки баз даних і багато яких інших. В Visual C# є розширений редактор коду, конструктори зі зручним користувальницьким інтерфейсом, вбудований відладник і багато інших засобів, покликані спростити розробку додатків мовою Visual C# версії 5.0 і .NET Framework версії 4.5.

Синтаксис Visual C# дуже виразний, але простий у вивченні. Усі, хто знаком з мовами C, C++ або Java з легкістю взнають синтаксис із фігурними дужками, характерний для мови Visual C#. Розроблювачі, що знають кожну із цих мов, як правило, зможуть домогтися ефективної роботи з мовою Visual C# за дуже короткий час. Синтаксис Visual C# робить простіше те, що було складно в C++, і забезпечує потужні можливості, такі як типи значень Nullable, перерахування, делегати, лямбда-вираження й прямий доступ до пам'яті, чого немає в Java. Visual C# підтримує універсальні методи й типи, забезпечуючи більше високий рівень безпеки й продуктивності, а також ітератори, що дозволяють при реалізації колекцій класів визначати власне поводження ітерації, що може легко використовуватися в клієнтському коді. В Visual C# 5.0 вираження LINQ (Language-Integrated Query) роблять строго-типізований запит першокласною конструкцією мови.

Як об'єктно-орієнтована мова, Visual C# підтримує поняття інкапсуляції, спадкування й поліморфізму. Всі змінні й методи, включаючи метод `Main` – точку входу додатка – інкапсулюється у визначення класів. Клас може

успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово override, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримуюче спадкування.

На додаток до основних описаних об’єктно-орієнтованих принципів, мова Visual C# спрощує розробку компонентів програмного забезпечення завдяки декільком інноваційним конструкціям мови, у число яких входять наступні:

– Інкапсульовані підписи методів, називані делегатами, які підтримують строго-типізовані повідомлення про події.

– Властивості, що виступають у ролі методів доступу для закритих змінних-членів.

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

– LINQ (Language-Integrated Query), що пропонує вбудовані можливості запитів у різних джерелах даних.

Якщо буде потрібно забезпечити взаємодію з іншим програмним забезпеченням Windows, таким як об'єкти COM або власні бібліотеки DLL Win32, у мові Visual C# можна використовувати процес, що називається "Interop". Процес Interop дозволяє програмам на Visual C# виконувати практично будь-які дії, які може виконувати вихідний додаток на C++. Мова Visual C# підтримує навіть покажчики й поняття "небезпечного" коду для тих випадків, коли прямий доступ до пам'яті має вкрай важливе значення.

Процес побудови Visual C# у порівнянні з C і C++ простий і є більше гнучким, чим в Java. Немає окремих файлів заголовка, а методи й типи не потрібно повідомляти в певному порядку. У вихідному файлі Visual C# може бути визначене будь-яке число класів, структур, інтерфейсів і подій.

Архітектура платформи .NET Framework

Програма мовою Visual C# виконується в середовищі .NET Framework – інтегрованому компоненті Windows, що містить віртуальну систему виконання (середовище CLR) і уніфікований набір бібліотек класів. Середовище CLR являє собою комерційну реалізацію корпорацією Майкрософт інфраструктури CLI, що є міжнародним стандартом, який лежить в основі створення середовищ виконання й розробки, у яких забезпечується тісна взаємодія між мовами й бібліотеками.

Вихідний код, написаний мовою Visual C#, компілюється в проміжну мову (IL) у відповідності зі специфікацією CLI. Код IL і ресурси, такі як растрові зображення й рядки, зберігаються на диску у файлі, що виконується, називаному складанням, з розширенням EXE або DLL у більшості випадків. Складання містить маніфест із відомостями про типи складання, версії, мови й регіональні параметри та вимоги безпеки.

При виконанні програми на Visual C# складання завантажується в середовище CLR залежно від відомостей у маніфесті. Далі, якщо вимоги безпеки дотримані, середовище CLR виконує JIT-компіляцію для перетворення коду IL в інструкції машинного коду. Середовище CLR також надає інші служби, що відносяться до автоматичного збору сміття, обробки виключень і керуванню ресурсами. Код, виконуваний середовищем CLR, іноді називають "керованим кодом" у протиставлення "некерованому коду", що компілюється в машинний код, призначений для певної системи. Далі показані відносини під час компіляції й час виконання між файлами з вихідним кодом Visual C#, бібліотеками класів .NET Framework, складаннями й середовищем CLR.

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж 20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних

на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як якби вони були написані на одній мові.

Крім служб часу виконання, в.NET Framework також є велика бібліотека, що складається з більш ніж 4000 класів, організованих по просторах імен, які забезпечують різноманітні корисні функції для будь-яких дій, починаючи від введення й виведення файлів для керуванням рядками для розбивки XML, і закінчуючи елементами керування Windows Forms. У звичайному додатку мовою Visual C# бібліотека класів .NET Framework інтенсивно використовується для "устрою" коду.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи визначення контексту користувача на основі архітектури CoDA.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Семантика важлива в будь-якій області, а особливо – у сервіс-орієнтованій архітектурі (Service-oriented architecture – SOA). Оскільки SOA зв'язує між собою команди й організації, погоджена термінологія надзвичайно важлива. У цій серії статей ми проводимо ознайомлювальний тур по SOA, визначаючи терміни й варті за ними ідеї. Тут ми визначаємо термінологію, достатню для спілкування в області SOA. Визначемо, чому кожний з термінів важливий для SOA, що він значить у даному контексті, з якими стандартами він зв'язаний і чим відрізняється від інших термінів.

Перераховані нижче терміни не вибудувані за алфавітом, так само як і по ступені важливості. Вони організовані скоріше в блоковому стилі. Ми починаємо з "сервісу", оскільки це фундаментальне поняття в рамках SOA. На цьому понятті ми будуємо визначення таких термінів, як "архітектура", "керування" і "бізнес". У багатьох випадках ми розбиваємо об'ємні терміни на їхні складені компоненти.

Сервіс

Визначити поняття сервісу на початку статті непросто, оскільки воно тягне за собою багато інших визначень. Наприклад, вам може здатися, що наведене вище визначення занадто технічне. Однак ураховуйте, що важливо не зациклюватися на формальному визначенні сервісу, а сфокусувати увага на ключових ідеях, які за ним коштують, таких як:

– **Орієнтація на бізнес**: сервіси орієнтуються не на можливості IT, а на потреби бізнесу. Орієнтація сервісів на бізнес підтримується аналізом сервісу й технікою проектування.

– **Інструкції**: сервіси самодостатні й описуються в термінах інтерфейсів, операцій, семантики, динамічних характеристик, політик і властивостей сервісу.

– **Повторне використання**: повторне використання сервісів забезпечується їхнім модульним плануванням.

– **Угоди**: сервісні угоди полягають між сутностями, іменованими постачальниками й користувачами. Ці угоди ґрунтуються на сервісних інструкціях і не впливають на реалізацію самих сервісів.

– **Розміщення й видимість**: протягом свого життєвого циклу сервіси розміщаються й стають видимі завдяки сервісним метаданим, реєстрам і сховищам.

– **Агрегація**: на слабко зв'язаних сервісах будуються об'єднуючий бізнес-процеси й складні додатки для одного або декількох підприємств.

Ці характеристики в сукупності показують, що SOA має справу не тільки з "технологіями", але й з потребами й потребами бізнесу.

Також важливо враховувати, що не все є сервісом. Є такі IT-функції, які розміщати як сервіси змісту немає. Такі аналітичні техніки, як IBM Service-Oriented Modeling and Architecture (SOMA), допомагають визначати відповідність сервісів перерахованим вище ідеям. Всі ці моменти (включаючи всі виділені терміни з даного розділу) ми докладно розглянемо в даному розділі.

Архітектура

Як і для сервісів, для архітектури теж важко підібрати задовольняюче всіх визначення. Однак, на відміну від сервісів, коли люди говорять про SOA, про архітектуру часто забувають, і зрячи! По суті, архітектура підприємства й сервіс-орієнтована архітектура мають загальну мету – підтримка бізнесу за допомогою інтегрованої IT-стратегії. Архітектори масштабу підприємства, наприклад, є ключовим елементом для успіху SOA, оскільки саме вони розраховують стратегічну еволюцію IT-систем підприємства, ґрунтуючись на потребах, що розвиваються, і потребах бізнесу.

На форумі Open Group Architecture Forum (TOGAF) є два визначення архітектури залежно від контексту:

– "Формальний опис або докладний план системи на рівні компонентів для керівництва в процесі її створення.

– Структура компонентів, їхнього взаємозв'язку, принципи й напрямки розвитку, що визначають їхню розробку й еволюцію."

Ці два визначення критично важливі для розуміння "А" (архітектури) з абревіатури SOA. Заглянувши глибше, ми бачимо також, що архітектура необхідна для наступні:

– Розробка й моделювання на різних рівнях абстракції

– Відділення інструкції від реалізації

– Побудова гнучких систем

– Перевірка на відповідність бізнес-вимогам

– Аналіз обсягу змін з появою нових вимог

– Перевірка на відповідність правилам

Архітектура підприємства

От визначення з Вікіипедії: "Архітектуру" можна віднести на рівень проектів, а "архітектуру підприємства" – на рівень організацій. Відзначте також згадування про процеси, інформаційні системи, персонал, цілях, стратегії й бізнес-орієнтації IT. Головна мета створення архітектури підприємства – узгодження бізнес-стратегії й вкладень у сектор IT. Таким чином, архітектура підприємства дозволяє від загальної бізнес-стратегії перейти на рівень нижче, до лежачій у її основі технології."

Ви можете віднести "архітектуру" на рівень проектів, а "архітектуру підприємства" – на рівень організацій. Відзначте також згадування про процеси, інформаційні системи, персонал, цілях, стратегії й бізнес-орієнтуванню IT.

Сервіс-орієнтована архітектура

Оскільки сама модель сервіс-орієнтована, вона дозволяє здійснювати поступове прийняття SOA при виникненні в бізнесу нових потреб, починаючи з маленьких проектів і згодом інтегруючись у всю організацію. Більше докладну інформацію про SOA foundation можна знайти в Ресурсах.

SOA має наступні характеристики:

– Вона поліпшує взаємозв'язок між архітектурою підприємства й бізнесом.

– Вона дозволяє з наборів інтегрованих сервісів створювати складні додатки.

– Вона створює гнучкі бізнес-процеси.

Структура рішень SOA

Структура складається з 5 функціональних шарів (знизу нагору):

– **Експлуатаційні системи**: представляють існуючі IT-рішення, показують цінність і важливість вкладень в IT для SOA і можливість їхнього використання.

– **Сервісні компоненти**: реалізують сервіси, при цьому можуть використовувати одне або більше додатків з рівня експлуатаційних систем. Як ви бачите з моделі, у користувачів і бізнес-процесів, на відміну від сервісів, немає прямого доступу до компонентів. Існуючі компоненти можуть бути повторно використані або, якщо вони для цього підходять, впроваджені в SOA.

– **Сервіси**: представляють розміщені в середовищі сервіси. Ці сервіси є керованими видимими сутностями.

– **Бізнес-процес**: представляє операційні програми, що створюють бізнес-процеси у вигляді хореографій сервісів.

– **Користувачі**: представляють канали, використовувані для доступу до бізнес-процесам, сервісам і додаткам.

Керування

Для успішного прийняття SOA керування необхідно, зокрема, через крос-організаційної природи SOA, де власники, розроблювачі, програмісти, технічний персонал і користувачі можуть перебувати в різних організаціях, бізнесах, IT-департаментах, галузях бізнесу, відділах або департаментах.

Цей розділ містить визначення з IBM Rational Method Composer Plug-in for SOA Governance. Тут визначаються терміни "керування", " IT-керування", " SOA-

керування", а також визначається різниця між керуванням, керівництвом і відповідністю. Відразу описуються проблеми, що коштують перед SOA-керуванням. Більше докладну інформацію про Rational Method Composer ви знайдете в Ресурсах.

Керування

"Під керуванням ми маємо на увазі:

– Установлення ланцюжків відповідальності, повноважень і зв'язків, щоб уповноважити людей (права на прийняття рішень)

– Установлення механізмів для вимірів, розрахунків і контролю для того, щоб люди могли виконувати свої обов'язки в рамках своєї відповідальності

– Керування стежить за призначенням прав на прийняття рішень і вирішує, якими критеріями й правилами керуватися при прийнятті цих рішень. Правами на прийняття рішень наділяються певні посади, а не особистості. На відміну від керування, керівництво включає призначення персоналу на посади й спостереження за виконанням правил.

– Частиною будь-якого управлінського рішення є відповідність організаційним правилам відповідності. Відповідність – це документований доказ того, що керування є й реалізується: рішення документуються, а правила прийняття рішень дотримуються."

IT-керування

IT-керування стосується тих моментів керування, які ставляться до процесів у сфері інформаційних технологій в організації й того, наскільки ці процеси відповідають цілям бізнесу.

IT-керування може бути описане як призначення прав на прийняття рішень і засобів оцінки IT-процесів.

SOA-керування

"SOA-керування – це надбудова над IT-керуванням, орієнтована на сервіси та інші продукти життєвого циклу SOA."

Зокрема, SOA-керування фокусується на методах і процесах, що стосується ідентифікації сервісів, фінансування, прав власності, розробки,

програмування, розміщення, повторного використання, виявлення, доступу, моніторингу, керівництва й вилучення з обігу.

"SOA-керування призначене для рішення наступних проблем:

– Які нові організаційні посади й структури потрібні для ідентифікації, розробки й спільного використання сервісів?

– Які метрики підтримують вкладення засобів, обслуговування, життєздатність і спільне використання сервісів?

– Як бізнесу вирішити питання про інвестиції в створення й обслуговування сервісів?

– Що таке зрілість виробництва в області сервісу-орієнтованості?

– Які необхідні утворення, навчання або наставництво?"

Життєвий цикл SOA

У визначенні життєвого циклу SOA IBM SOA foundation відзначає чотири фази:

– Модель складається з бізнес-аналізу й розробки (вимоги, процеси, мети, ключові показники продуктивності) і IT-аналізу й розробки (визначення й специфікація сервісів).

– Складання складається із програмування сервісів і побудови складних додатків.

– Розміщення складається з розміщення додатків і засобів часу виконання, таких як Enterprise Service Buses (ESB).

– Керівництво складається з підтримки операційного середовища, моніторингу продуктивності сервісів і спостереження за дотриманням сервісних політик.

3.2 Розробка структурної схеми

Безпека продукту

CoDA пропонує функції безпеки продуктів корпоративного рівня для більшого контролю, прозорості та гнучкості:

– Автентифікація: CoDA пропонує широкий спектр методів автентифікації, таких як SSO з SAML 2.0, вхід за допомогою Google, Microsoft, Apple, Magic Links та базова автентифікація з 2-факторною автентифікацією. Корпоративні клієнти можуть встановлювати політики автентифікації та керувати користувачами за допомогою SCIM.

– Шифрування: Дані клієнтів шифруються під час передачі та зберігання за замовчуванням.

– Авторизація: керування доступом до документів, папок, пакетів та робочих просторів. Спільний доступ із групами Google. Корпоративні клієнти також можуть ділитися файлами з групами SCIM та встановлювати розширені правила спільного доступу для форм, документів та пакетів.

– Аудит: Ми надаємо API аудиту для корпоративних клієнтів, щоб отримувати журнали аудиту за останні 12 місяців. CoDA також надає пакет адміністратора для перегляду подій аудиту.

– Політики підприємства: підприємства можуть встановлювати політики для керування користувачами, документами та пакетами. До них належать політики для автентифікації користувачів, зовнішнього та вхідного спільного доступу, публікації, створення спільних папок, експорту даних, завантаження файлів та тривалості сеансу.

– Панелі керування Enterprise: CoDA пропонує корпоративним клієнтам розширені адміністративні панелі керування для легкого керування їхніми робочими процесами в CoDA. До них належать панелі керування для перегляду та керування ліцензіями, загальнодоступними документами, активністю користувачів та документами, що належать користувачам з деініціалізацією.

– Конфігурації та схвалення пакетів: CoDA пропонує високорозвинені засоби контролю безпеки для інтеграцій сторонніх розробників. Підприємства мають повний контроль над тим, які дані можна вносити до CoDA, хто може їх вносити та хто може до них отримувати доступ.

Безпека застосунків

Зобов'язання CoDA щодо безпеки починаються з процесів, інструментів та практик для безперервного проектування та розробки безпечного програмного забезпечення:

– Життєвий цикл розробки безпеки: Наша програма життєвого циклу розробки безпеки інтегрована в кожен етап процесу розробки програмного забезпечення для безперервного створення безпечного програмного забезпечення. Прикладами є щорічні тренінги з безпеки для всіх співробітників, моделювання загроз як частина процесу проектування та інструменти статичного аналізу коду.

– Щорічне тестування на проникнення: CoDA проводить щорічне тестування на проникнення, яке проводять авторитетні дослідницькі фірми з безпеки. Сфера цього тестування включає: веб-застосунки, інфраструктуру пакетів CoDA, хмарну інфраструктуру та мобільні застосунки.

– Програма публічної винагороди за виявлені помилки: CoDA запускає публічну програму винагороди за виявлені помилки через HackerOne.

Безпека інфраструктури

CoDA створена з нуля з використанням найкращих практик безпеки AWS:

– Хмарна інфраструктура: CoDA побудована з використанням усталених принципів безпеки, включаючи глибоко ешелонований захист, мінімальні привілеї та зменшення площі поверхні атаки. CoDA дотримується найкращих практик AWS щодо мережевої безпеки, використовуючи такі сервіси, як AWS CloudFront, AWS WAF, групи безпеки AWS та VPC.

– Безпека операцій: Ми використовуємо багатофакторну автентифікацію, контроль доступу на основі ролей (RBAC) та гранти доступу «точно вчасно» для

безпечного керування нашим сервісом. Крім того, ми реєструємо події аудиту та інформацію про безпеку на кожному рівні нашої інфраструктури та відстежуємо їх на наявність підозрілої активності.

– Резервні копії: CoDA щодня створює резервні копії баз даних для всіх даних, що зберігаються в AWS. Резервні копії зазвичай зберігаються щонайменше 35 днів. Ми також проводимо регулярні перевірки цілісності резервних копій та проводимо навчання з аварійного відновлення на робочому столі щонайменше раз на рік.

– Синхронізація годинника: інформаційні ресурси CoDA використовують протокол NTP для синхронізації із затвердженими джерелами NTP.

– Безпека пакетів: пакети виконуються в ізольованих захищених пісочницях. Розробник пакетів ніколи не торкається облікових даних авторизації, які використовуються користувачами в пакетах. Ми гарантуємо, що пакети обмінюються даними лише з тими веб-сайтами, про які вони заявляють.

– Шифрування: CoDA використовує службу керування ключами Amazon (KMS) для створення, обслуговування та ротації ключів шифрування. CoDA також використовує протокол Transport Layer Security (TLS 1.2) для шифрування даних користувачів під час передачі та симетричний алгоритм шифрування AES-256 для шифрування даних клієнтів у стані спокою.

Состав Комплексу

Комплекс складається з наступних компонентів:

1. Набір шлюзів кодування.
2. Центр генерації ключів.
3. Центр розподілу ключів.
4. Центр реєстрації мобільних клієнтів.
5. Центр підготовки електронних ключів мобільних клієнтів.
6. Мобільний клієнт.
7. Центр моніторингу.
8. Програма контролю цілісності.

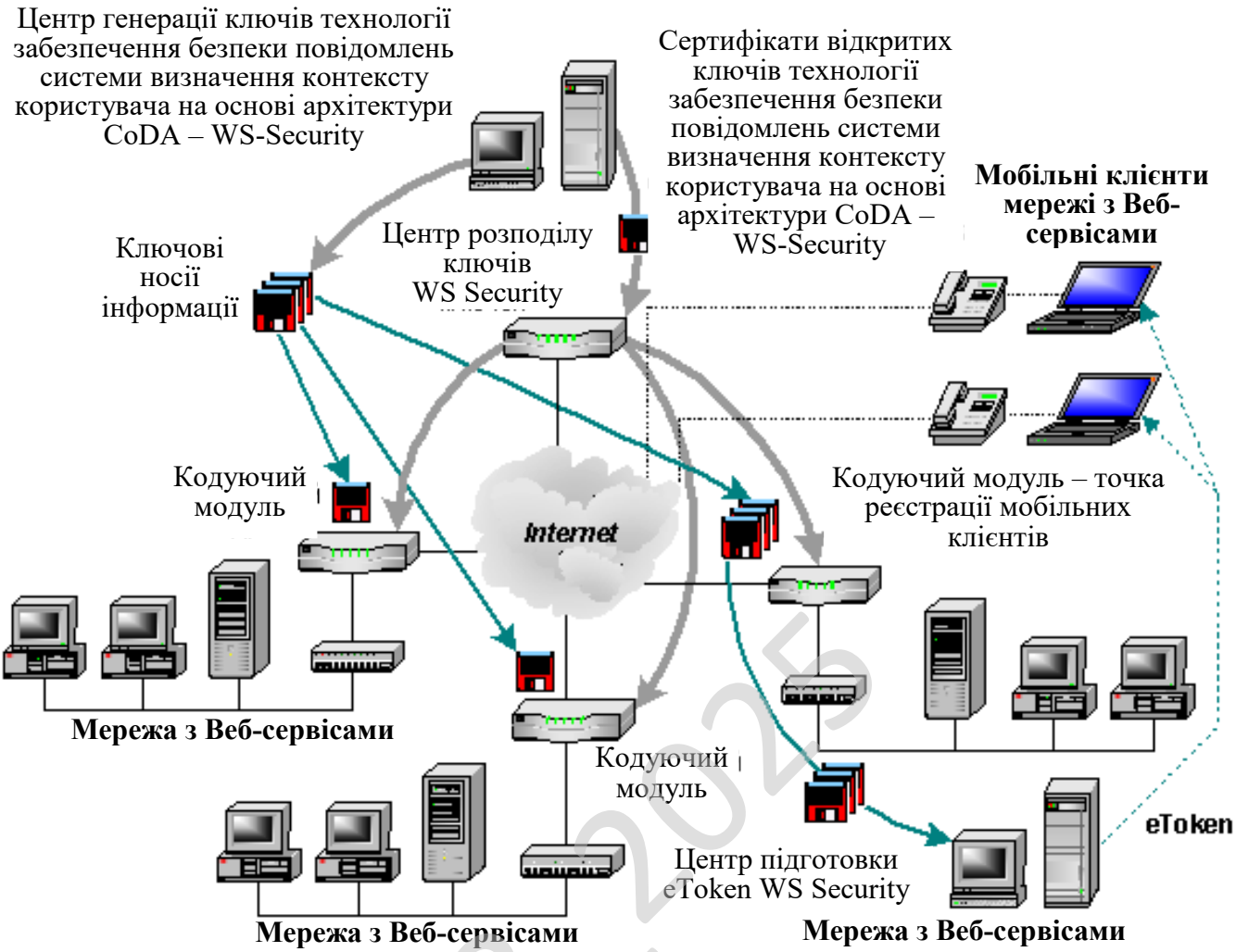

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

Розглянемо функціональну схему розробленої системи. Вона зображена на рисунку 3.2.

Як видно з рисунку система складається з наступних елементів:

1. Блок визначення алгоритму для формування ключа. Він може бути для симетричних та асиметричних алгоритмів. У першому випадку формується один ключ, який таємно розсилається учасникам зашифрованого обміну даними, у другому випадку генерується пара ключів: відкрий ключ – таємний ключ. Цей блок призначений для визначення по алгоритму шифрування параметрів та

обмежень при формуванні ключа.

2. Генератор випадкових чисел. Призначений для генерації випадкових чисел, які використовуються як при генерації ключів шифрування/дешифрування, так й для формування іншої інформації потрібної для авторизації та автентифікації учасників інформаційного обміну.

3. Блок генерації ключів. Призначений для генерації ключів для учасників інформаційного обміну.

4. Блок формування сертифікатів. Призначений для формування сертифікатів, які удостовірюють справжність ключів, та автентифікують користувачів.

5. База даних відкликаних сертифікатів, призначена для зберігання тих сертифікатів, які були зкмпроментовані.

6. Центр реєстрації.

Розглянемо ці блоки та їх функціональність більш детальніше.

Центр розподілу ключів

Центр розподілу ключів (ЦРК) призначений для обслуговування наступних запитів: на видання сертифікатів ЕЦП, на відкликання, призупинення й поновлення припиненої дії сертифікатів абонентів, сформованих на мережних вузлах або в Центрах Реєстрації для зовнішніх користувачів.

ЦРК забезпечує наступну функціональність:

1. Перше видання сертифіката підпису абонента відбувається в ЦРК разом з генерацією секретного ключа для нього. Подальше перевидання сертифіката може відбуватися як у ЦРК одночасно з формуванням нового секретного ключа (для завдань, що вимагають централізованої генерації й розподілу ключів), так і по запиту користувача корпоративної мережі, сформованого на його мережному вузлі.

2. Видання й реєстрація сертифікатів ЕЦП по запиту абонентів мережі. Запит на сертифікат являє собою шаблон сертифіката, що містить інформацію про абонента, його новий відкритий ключ підпису, передбачуваний

термін дії сертифіката, а також інші параметри, що відповідають стандарту X.509. Запит може бути зареєстрований або автоматично або в результаті дій адміністратора ЦРК. Запит може бути відхилений. Після заповнення полів сертифіката сертифікат через Центр керування відправляється до користувача на комп'ютер.

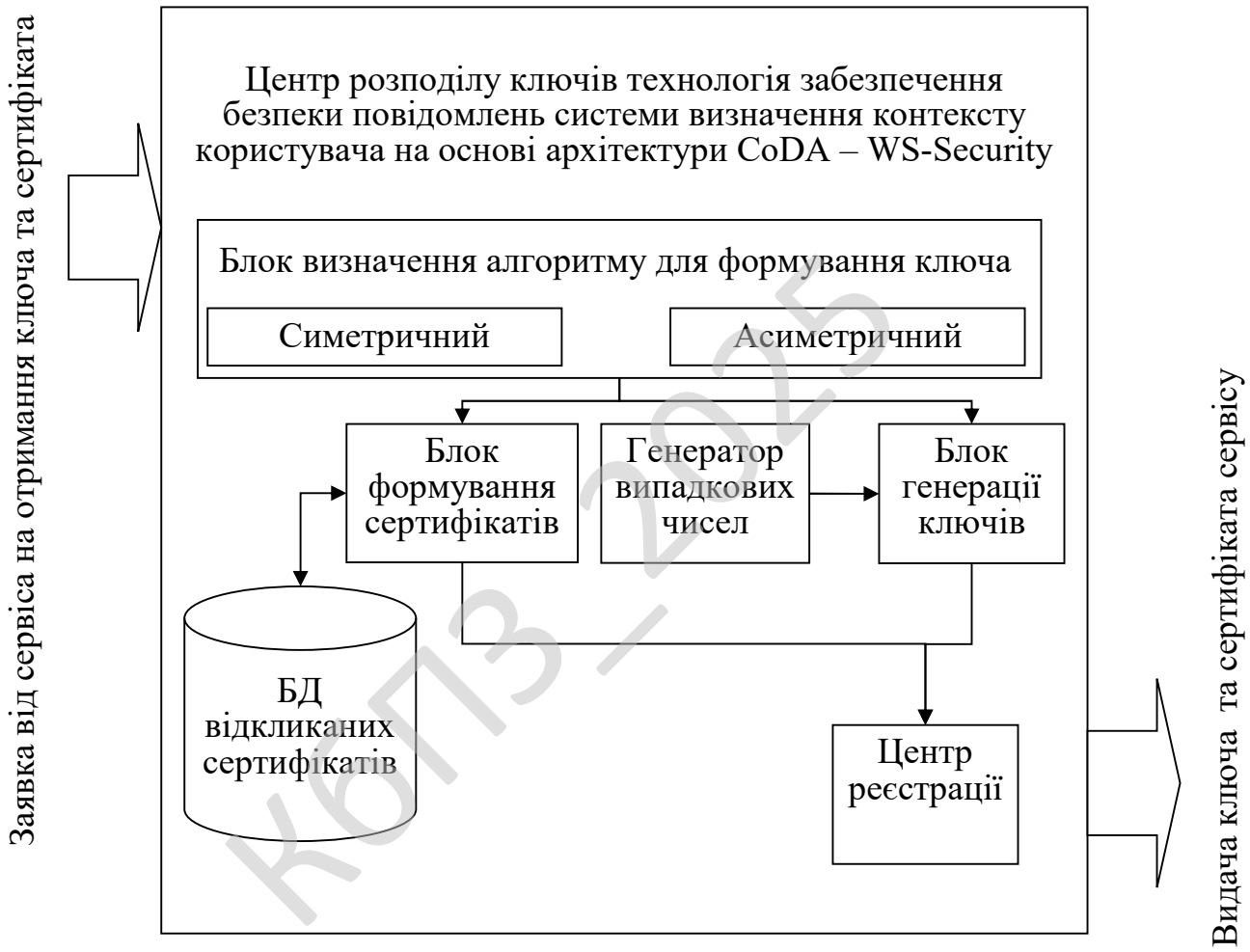

Рисунок 3.2 – Функціональна схема системи

3. Відкликання сертифікатів, призупинення дії сертифікатів, поновлення дії сертифікатів ЕЦП абонентів мережі. Ці дії виконуються адміністратором ЦРК. Довідник відкликаних сертифікатів розсилається абонентам мережі.

4. Реєстрація довідників сертифікатів ЕЦП головних абонентів інших ЦРК. Після перегляду сертифікатів головних абонентів інших ЦРК і прийняття їх виробляється підпис такого довідника своїм головним абонентом (крос сертифікація). Завірений довідник розсилається по мережі у відповідності зі зв'язками своїх абонентів, і використовуються при перевірці сертифікатів ЕЦП абонентів інших ЦРК, що надіслали підписану інформацію на який-небудь вузол своєї мережі.

5. Аналогічним чином виконується імпорт, крос сертифікація й розсилання сертифікатів ЦРК інших виробників на основі сертифікованих СКЗІ**.** Документи, підписані з використанням зазначених СКЗІ, будуть перевірені тільки при наявності на комп'ютері сертифіката відповідного ЦРК, завіреного Головним абонентом свого ЦРК. По міркуваннях безпеки ЦРК вимагає крос сертифікації навіть для сертифікатів інших ЦРК, у ланцюжку сертифікатів яких утримується сертифікат, якому довіряє ЦРК.

6. Реєстрація довідників відкликаних сертифікатів ЕЦП із інших ЦРК. Такі довідники надходять із інших мереж автоматично, засвідчуються головним абонентом і розсилаються по мережі у відповідності зі зв'язками абонентів мережі. Імпорт довідників відкликаних сертифікатів зі ЦРК інших виробників не виробляється. Доступ до них здійснюється в процесі перевірки підпису по шляху, зазначеному в ЕЦП.

7. Обслуговування запитів зовнішніх користувачів. Зовнішній користувач реєструється на одному з пунктів реєстрації. Адміністратор створює запит на сертифікат ЕЦП для зовнішнього користувача й відсилає його в ЦРК для видання сертифіката. Запит на сертифікат перед відправленням у ЦРК підписується ключем підпису цього Адміністратора. Після введення в дію сертифіката зовнішній користувач зможе користуватися ним (підписувати документи) на будь-якому вузлі мережі із установленим ПЗ. Адміністратор може створювати запити на відкликання, призупинення дії, поновлення дії припиненого сертифіката ЕЦП зовнішніх користувачів.

8. Видання й реєстрація сертифікатів ЕЦП для зовнішніх користувачів виконується тільки по запиту із Центра реєстрації (ЦР). Запит може бути зареєстрований або відхилений. Вторинні запити на сертифікати, якщо в них немає змін, і видача сертифікатів можуть оброблятися й видаватися автоматично. Сертифікати відправляється в ЦРК.

9. Відкликання сертифікатів, призупинення дії сертифікатів, поновлення дії сертифікатів ЕЦП зовнішніх користувачів може відбуватися по запиту зі ЦР, або самим Адміністратором ЦРК без запиту зі ЦР. Довідники відкликаних сертифікатів розсилаються по вузлах мережі.

10. Перегляд запитів і сертифікатів ЕЦП.

11. Розбір конфліктних ситуацій і експертизи правочинності й дійсності електронних документів, підписаних ЕЦП, виробляється відповідно до Документа "СКЗІ. Порядок розбору конфліктних ситуацій, пов'язаних із застосуванням ЕЦП".

12. Сервісні функції ЦРК.

ЦРК інформує адміністратора про витікання термінів дії різних сертифікатів за задане число днів до цього строку шляхом формування відповідних списків.

Автоматично формує архіви інформації через задані інтервали часу при наявності змін, що забезпечує можливість відновлення актуальної інформації.

Ключовий центр

Ключовий центр (КЦ) забезпечує захист інфраструктури ЕЦП за допомогою створення системи шифрування інформації у всіх процедурах керування інфраструктурою ЕЦП на основі симетричної схеми розподілу ключів:

– Захист секретних ключів ЕЦП, що розподіляються централізовано.

– Захист сертифікатів абонентів мережі.

– Захист транспортного рівня системи, що забезпечує роботу процедур запитів і одержання сертифікатів, доставки інших файлів інфраструктури ЕЦП.

– Генерацію паролів для захисту секретних ключів від несанкціонованого доступу. Тип пароля може бути – випадковий (пароль буде створений випадковим образом з різних слів, що утворять випадкову фразу, що запам'ятовується легко), власний (можна задати за бажанням, але не менш 5 символів), випадковий цифровий (сформується випадковим образом з різних цифр).

– Формування інших ключів, що забезпечують відновлення симетричної ключової інформації й роботу іншого ПЗ.

Первісний набір симетричних ключів видається користувачеві в складі файлу ключового дистрибутива, зашифрованого на паролі, і який користувач одержує при його первинній реєстрації. До складу ключового дистрибутива входить персональний ключ зв'язку з УКЦ, ключ зв'язку зі своїм координатором, первинний секретний ключ підпису й сертифікат, сертифікати головних абонентів ЦРК. Інша ключова інформація надходить на комп'ютер після інсталяції ПЗ і в процесі відновлень.

Центр керування мережею

Центра керування забезпечує:

– Реєстрацію вузлів і абонентів корпоративної мережі, реєстрацію "Центрів реєстрації" зовнішніх користувачів.

– Взаємодія зі ЦРК і користувачами при керуванні сертифікатами.

– Формування захищених довідників доступу для вузлів мережі й довідників зв'язків вузлів і абонентів для УКЦ при штатній експлуатації й компрометації ключів абонентів.

– Інші функції.

Взаємодія абонентів зі ЦРК виробляється тільки через Центр керування мережею.

Центр реєстрації

Центр Реєстрації призначений для реєстрації зовнішніх користувачів і одержання для них цифрових сертифікатів. У Центрі Реєстрації при пред'явленні

документів зовнішнім користувачем, що підтверджує його повноваження, створюється запит на сертифікат, виробляється відправлення його в ЦРК і здійснюється запровадження в дію виданого в ЦРК сертифіката. У Центрі Сертифікації сертифікат буде або задоволений, або відхилений. Тільки запит на сертифікат зі статусом "задоволений" стає сертифікатом підпису, і цей сертифікат може бути уведений у дію. Після введення в дію сертифіката, зовнішній користувач зможе користуватися ним (підписувати документи) на будь-якому вузлі із установленим ПЗ.

Центр Реєстрації виконує наступні функції:

– Генерація секретного ключа підпису й збереження його на персональному ключовому носії зовнішнього користувача.

– Уведення персональних даних для сертифіката зовнішнього користувача.

– Формування запиту на сертифікат.

– Підпис запиту на сертифікат ключем діючого адміністратора ЦР.

– Відправлення завіреного запиту в ЦРК

– Прийом сертифікатів зі ЦРК (відбувається автоматично).

– Перегляд запитів і прийнятих сертифікатів.

– Запровадження в дію сертифіката (збереження на персональному ключовому носії зовнішнього користувача).

– Формування запиту на відкликання сертифіката.

– Формування запиту на призупинення сертифіката.

– Формування запиту на поновлення сертифіката.

Крім того, Центр Реєстрації виконує експорт сертифікатів у різних кодуваннях.

Секретний ключ зовнішнього користувача і його сертифікат заносяться на його персональний носій. Це може бути дискета, компактний диск (CD), E-Token, смарт-карта, Touch memory і інші.

Секретний ключ зашифровується на паролі, вироблюваному програмою. Тип пароля може бути один з наступних:

– Власний пароль.

– Випадкова фраза, що запам'ятовується легко.

– Власний цифровий пароль.

– Випадковий цифровий пароль.

Розподіл відкритих ключів

Розглянемо основні алгоритми розподілу ключів.

На сьогоднішній день відомі наступні методи розподілу відкритих ключів:

– індивідуальне публічне оголошення відкритих ключів сервісами;

– використання привселюдно доступного каталогу відкритих ключів;

– участь авторитетного джерела відкритих ключів;

– сертифікати відкритих ключів.

Розглянемо кожний з перерахованих методів.

При *індивідуальному публічному оголошенні відкритих ключів* будь-яка сторона, що бере участь в обміні повідомленнями (X), може надати свій відкритий ключ (K_O) будь-якій іншій стороні. Недоліком даного підходу є неможливість забезпечити автентифікацію відправника відкритого ключа (K_O). Тобто, при даному підході в порушника з'являється можливість фальсифікації сервісів.

Використання привселюдно доступного каталогу відкритих ключів дозволяє домогтися більше високого ступеня захисту інформації й мережі побудованої з використанням сервіс-орієнтованої архітектури. У цьому випадку за ведення й поширення публічного каталогу повинна відповідати надійна організація (уповноважений об'єкт). При цьому повинні дотримуватися наступні правила.

1. Сервіси повинні реєструвати свої відкриті ключі в публічному каталозі, що веде вповноважений об'єкт.

2. Реєстрація повинна проходити по заздалегідь захищених каналах зв'язку.

3. Уповноважений об'єкт повинен періодично публікувати каталог відкритих ключів.

Недоліком даного підходу є наступне. Якщо порушникові вдасться змінити записи, що зберігаються в каталозі відкритих ключів, то він зможе авторитетно видавати фальсифіковані відкриті ключі й, отже, виступати від імені кожного з учасників обміну даними й читати повідомлення, призначені будь-якому сервісу.

Участь авторитетного джерела відкритих ключів. Обов'язковою умовою даного варіанта розподілу відкритих ключів сервісів є умова, що авторитетне джерело відкритих ключів має свій секретний ключ, і кожний сервіс знає його відкритий ключ. При цьому виконується наступний порядок дій:

1. Сервіс 1 надсилає запит авторитетному джерелу відкритих ключів про поточне значення відкритого ключа сервісу 2. При цьому вказується дата й час запиту (д.вр.).

2. Авторитетне джерело, використовуючи свій секретний ключ $K_{\text{С}}^{\text{А}}$, шифрує й передає повідомлення сервісу 1 $Y_1 = E_{K_{\text{С}}^{\text{А}}}\left(K_{\text{О}}^{\text{П2}}, \text{д.вр.}\right)$, у якому втримується наступна інформація:

– $K_{\text{О}}^{\text{П2}}$ – відкритий ключ сервісу 2;

– буд. вр. – дата й час відправлення повідомлення.

3. Сервіс 1, використовуючи $K_{\text{О}}^{\text{П2}}$, шифрує й передає сервісу 2 шифроване повідомлення $Y_2 = E_{K_{\text{О}}^{\text{П2}}}\left(ID_1, N_1\right)$, що містить:

– ID_1 – ідентифікатор відправника (сервіс 1);

– N_1 – унікальну мітку даного повідомлення.

4, 5. Сервіс 2, одержавши шифроване повідомлення $Y_2 = E_{K_{\text{О}}^{\text{П2}}}\left(ID_1, N_1\right)$, дешифрує його за допомогою свого секретного ключа $K_{\text{С}}^{\text{П2}}$ $\left(ID_1, N_1\right) = D_{K_{\text{С}}^{\text{П2}}}\left(Y_2\right)$ й відповідно до ідентифікатора ID_1, аналогічно з пунктами 1 і 2 вище

перерахованих дій одержує від авторитетного джерела відкритий ключ сервісу 1 $K_{\text{О}}^{\text{П1}}$.

6. Сервіс 2, використовуючи $K_{\text{О}}^{\text{П1}}$, посилає сервісу 1 шифроване повідомлення $Y_4 = E_{K_{\text{О}}^{\text{П1}}}(N_1, N_2)$, де N_2 – унікальна мітка даного повідомлення.

7. Сервіс 1 шифрує за допомогою відкритого ключа $K_{\text{О}}^{\text{П2}}$ повідомлення Y, призначене сервісу 1 і передає $Y_5 = E_{K_{\text{О}}^{\text{П2}}}(Y, N_2)$.

Наведений варіант розподілу відкритих ключів має деякі недоліки:

– щораз, коли сервіс має намір передати інформацію новому адресатові, то він повинен звертатися до авторитетного джерела з метою одержання відкритого ключа;

– каталог імен і відкритих ключів, підтримуваний авторитетним джерелом, є привабливим місцем для порушника передачі інформації сервісів.

Сценарій розподілу відкритих ключів із застосуванням *сертифікатів відкритих ключів*. Обов'язковою умовою даного варіанта розподілу відкритих ключів сервісів є умова, що авторитетне джерело сертифікатів має свій секретний ключ $K_{\text{С}}^{\text{А}}$, і кожний сервіс знає його відкритий ключ $K_{\text{О}}^{\text{А}}$. При цьому виконується наступний порядок дій:

1. Сервіс 1 генерує пари ключів $\left[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}\right]$ (відповідно, відкритий і секретний) і по захищеному каналу зв'язку звертається до авторитетного джерела сертифікатів з метою одержання сертифіката.

2. Авторитетне джерело шифрує за допомогою свого секретного ключа $K_{\text{С}}^{\text{А}}$ сертифікат $C_{\text{П1}} = E_{\text{Кс}}^{\text{А}}\left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]$ і видає його сервісу 1. Сертифікат містить:

– $K_{\text{О}}^{\text{П1}}$ – відкритий ключ сервісу 1 (даний ключ сервіс 1 сам згенерував і передав авторитетному джерелу для сертифікації);

– $ID_{\text{П1}}$ – ідентифікатор сервісу 1;

– $T_{\text{П1}}$ – термін дії сертифіката сервісу.

3. Сервіс 1 пересилає свій сертифікат $C_{\text{П1}} = E_{\text{Кс}}^{\text{А}}\left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]$, отриманий від авторитетного джерела, сервісу 2. Останній, знаючи відкритий ключ

авторитетного джерела сертифікатів $K_{\text{О}}^{\text{А}}$, має можливість прочитати й упевнитися, що отримане повідомлення є сертифікатом $D_{K\text{о}}^{\text{А}}[C_{\text{П1}}] = D_{K\text{о}}^{\text{А}}\left[E_{K\text{с}}^{\text{А}}\left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]\right] = \left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]$.

4. 5, 6. Сервіс 2 виконує аналогічні дії, які були виконані сервісом 1 у пунктах 1, 2 і 3. Тобто одержує від авторитетного джерела сертифікат $C_{\text{П2}} = E_{K\text{с}}^{\text{А}}\left[K_{\text{О}}^{\text{П2}}, ID_{\text{П2}}, T_{\text{П2}}\right]$. Пересилає його сервісу 1. Останній, знаючи відкритий ключ авторитетного джерела сертифікатів $K_{\text{О}}^{\text{А}}$, має можливість прочитати й упевнитися, що отримане повідомлення є сертифікатом

$$D_{K\text{о}}^{\text{А}}[C_{\text{П2}}] = D_{K\text{о}}^{\text{А}}\left[E_{K\text{с}}^{\text{А}}\left[K_{\text{О}}^{\text{П2}}, ID_{\text{П2}}, T_{\text{П2}}\right]\right] = \left[K_{\text{О}}^{\text{П2}}, ID_{\text{П2}}, T_{\text{П2}}\right].$$

У результаті перерахованих дій сервіси обмінялися відкритими ключами й готові до передачі й прийому сервісних повідомлень.

Застосування криптосистеми з відкритим ключем для розподілу секретних ключів

На сьогоднішній день існує кілька підходів застосування криптосистеми з відкритим ключем для розподілу секретних ключів [7]. Розглянемо деякі з них.

Простий розподіл секретних ключів складається у виконанні наступних дій:

1. Сервіс 1 генерує пари ключів $\left[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}\right]$, відповідно, відкритий і секретний.

2. Сервіс 1 передає сервісу 2 повідомлення $X_{\text{П1}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{П1}}\right]$, де $ID_{\text{П1}}$ – ідентифікатор сервісу 1.

3. Сервіс 2, одержавши повідомлення $X_{\text{П1}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{П1}}\right]$ від сервісу 1, так само генерує свою пару ключів $\left[K_{\text{О}}^{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$.

4. Сервіс 2, використовуючи відкритий ключ $K_{\text{О}}^{\text{П1}}$ сервісу 1, шифрує й передає повідомлення $Y_{\text{П2}} = E_{K_{\text{О}}^{\text{П1}}}\left[ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$ сервісу 1.

5. Сервіс 1 знищує свій секретний ключ $K_{\text{С}}^{\text{П1}}$, а сервіс 2 знищує відкритий ключ сервісу 1 $K_{\text{О}}^{\text{П1}}$.

Таким чином, обидва сервіси мають сеансовий (секретний) ключ $K_{\text{С}}^{\text{П2}}$ і можуть використовувати його для передачі інформації, захищеної традиційним шифруванням. По закінченні сеансу передачі інформації ключ $K_{\text{С}}^{\text{П2}}$ знищується. Однак даний підхід уразливий для активних порушень. Дійсно, якщо порушник має можливість впровадження в з'єднання між сервісами, те, виконуючи наступні дії, він буде мати можливість знати секретний (сеансовий) ключ.

1. Сервіс 1 генерує пари ключів $\left[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}\right]$ і передає сервісу 2 повідомлення $X_{\text{П1}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{П1}}\right]$.

2. Порушник перехоплює повідомлення $X_{\text{П1}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{П1}}\right]$, створює власну пару ключів $\left[K_{\text{О}}^{\text{Н}}, K_{\text{С}}^{\text{Н}}\right]$ і передає сервісу 2 повідомлення $X_{\text{Н}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{Н}}\right]$.

3. Сервіс 2, одержавши повідомлення $X_{\text{Н}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{Н}}\right]$, генерує свою пару ключів $\left[K_{\text{О}}^{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$, шифрує (використовуючи відкритий ключ порушника $K_{\text{О}}^{\text{Н}}$) і передає повідомлення $Y_{\text{П2}} = E_{K_{\text{о}}^{\text{н}}}\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor$ сервісу 1.

4. Порушник перехоплює повідомлення $Y_{\text{П2}} = E_{K_{\text{о}}^{\text{н}}}\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor$, дешифрує його $\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor = D_{K_{\text{с}}^{\text{н}}}\left[Y_{\text{П2}}\right]$, визначає сеансовий ключ $K_{\text{С}}^{\text{П2}}$ і передає сервісу 2 повідомлення $Y_{\text{П2}} = E_{K_{\text{о}}^{\text{П1}}}\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor$.

У результаті обидва сервіси мають сеансовий ключ $K_{\text{С}}^{\text{П2}}$, однак не будуть підозрювати, що він теж відомий і порушникові.

Сценарій *розподілу секретних ключів із забезпеченням конфіденційності й автентичності* зображений на малюнку 2.10 і складається у виконанні наступних дій.

1. Сервіси генерують пари ключів, відповідно $\left[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}\right]$, $\left[K_{\text{О}}^{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$, і обмінюються між собою відкритими ключами $K_{\text{О}}^{\text{П1}}$ й $K_{\text{О}}^{\text{П2}}$.

2. Сервіс 1, використовуючи $K_{\text{О}}^{\text{П2}}$, передає сервісу 2 повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}\left\lfloor ID_{\text{П1}}, N_1 \right\rfloor$, що містить: свій ідентифікатор – $ID_{\text{П1}}$; N_1 – унікальна мітка даних повідомлень.

3. Сервіс 2, використовуючи $K_{\text{О}}^{\text{П1}}$, передає сервісу 1 повідомлення $Y_{\text{П2}} = E_{K_{\text{О}}^{\text{П1}}}\left\lfloor N_2, N_1 \right\rfloor$, що містить N_1 і N_2 - унікальні мітки даного повідомлення. Наявність мітки N_1 переконує сервісу 1 у тім, що тільки сервіс 2 міг дешифрувати повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}\left\lfloor ID_{\text{П1}}, N_1 \right\rfloor$.

4. Сервіс 1, використовуючи $K_{\text{О}}^{\text{П2}}$, передає сервісу 2 повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}\left\lfloor N_2 \right\rfloor$, що містить унікальну мітку N_2. Дане повідомлення виконує функцію підтвердження для сервісу 2, що його респондентом є сервіс 1.

5. Сервіс 1 генерує секретний (сеансовий) ключ $K_{\text{С}}$, що двічі шифрується з використанням: свого секретного ключа $E_{K_{\text{С}}^{\text{П1}}}[K_{\text{С}}]$ й відкритого ключа сервісу 2 $E_{K_{\text{О}}^{\text{П2}}}[\, E_{K_{\text{С}}^{\text{П1}}}[K_{\text{С}}]\,]$. Після виконання процедури шифрування повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}[\, E_{K_{\text{С}}^{\text{П1}}}[K_{\text{С}}]\,]$ передається сервісу 2. Останній, маючи відкритий ключ сервісу 1 і свій секретний ключ, дешифрує отримане повідомлення.

У результаті перераховані дії обидва сервіси мають секретний (сеансовий) ключ $K_{\text{С}}$.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

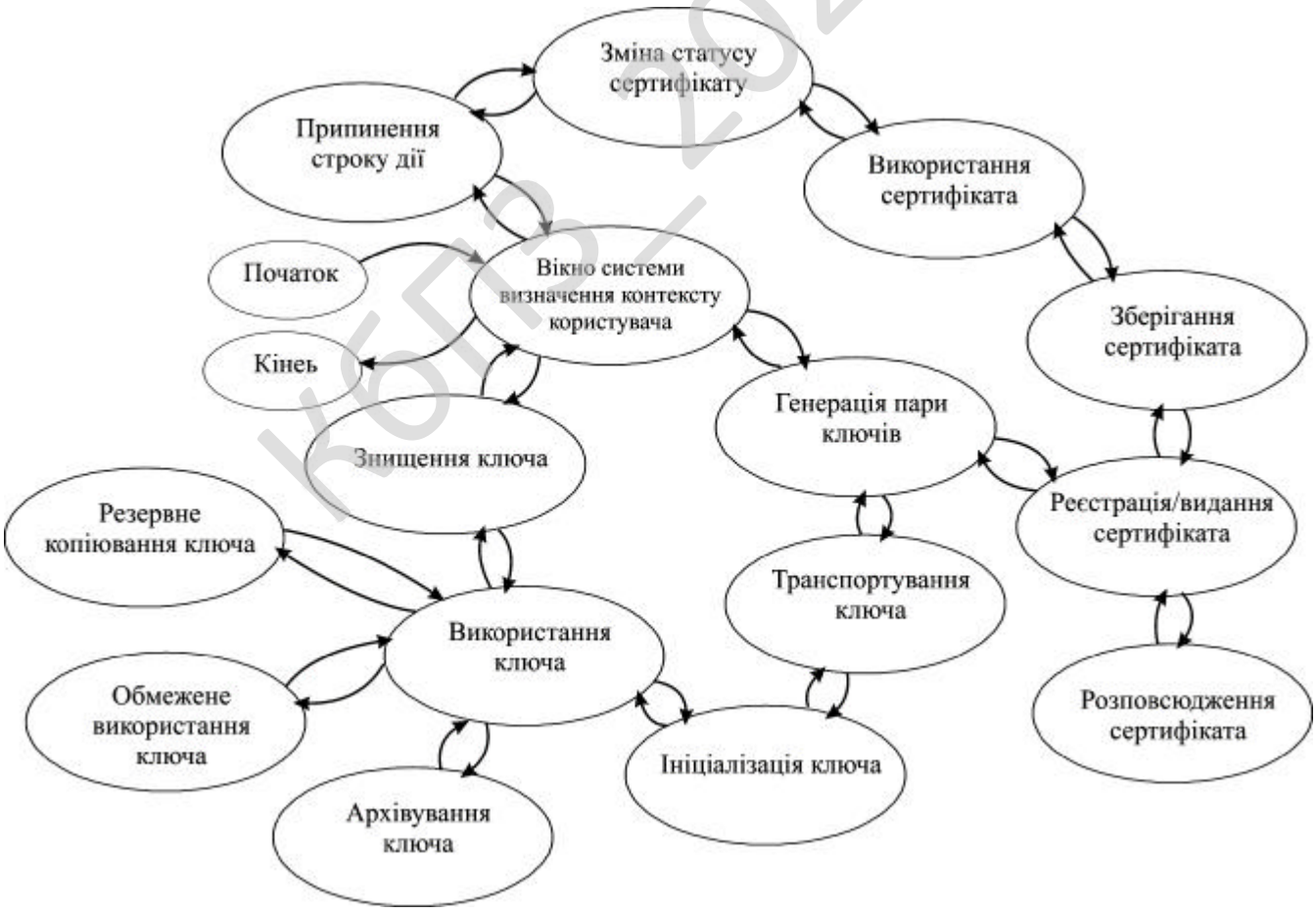

Рисунок 3.3 – Діаграма взаємодії процесів

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Під час роботи над магістерською роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Рисунок 4.1 – Блок-схема основної програми

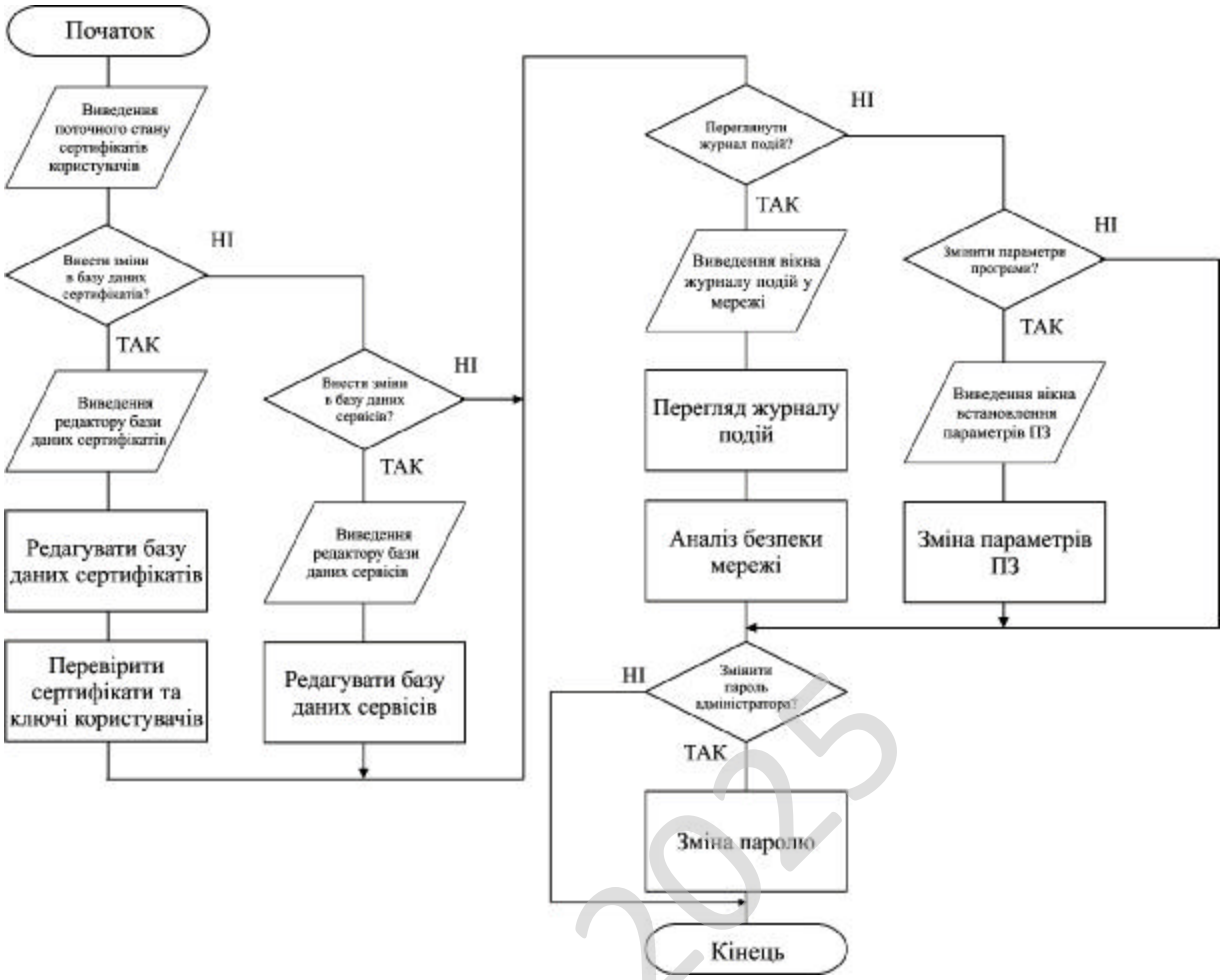

Рисунок 4.2 – Блок-схема роботи підпрограми

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю визначення контексту користувача на основі архітектури CoDA.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення,

візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Також при розробці магістерської роботи було використано наступні підходи UML: Діаграма класів; Діаграма діяльності (діаграми поведінки типу); Діаграма прецедентів (діаграми поведінки типу); Діаграма компонент; Діаграма об'єктів; Діаграма розгортання.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж

класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами

моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма компонент в UML це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

Модуль програмного забезпечення може бути представлено в якості компоненти. Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми.

Діаграма компонент відображає лише структурні характеристики, для відображення окремих екземплярів компонент слід використовувати діаграму розгортування.

Компоненти об'єднуються разом використовуючи структурні зв'язки (assembly connector) щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

Структурна взаємодія – «зв'язок двох компонент, який передбачає, що один з них надає послуги, потрібні іншому компоненту».

При використанні діаграми компонент щоб показати внутрішню структуру компонента, клієнтські та серверні інтерфейси можуть утворювати пряме з'єднання з внутрішніми. Таке з'єднання називається з'єднанням делегації.

Діаграма об'єктів в UML це діаграма, що відображає об'єкти та їх зв'язки в певний момент часу. Діаграма об'єктів може розглядатись як окремий випадок діаграми класів, на якій можуть бути представлені як класи, так і екземпляри (об'єкти) класів. Схожою за змістом є діаграма взаємодії (collaboration diagram).

Діаграми об'єктів не мають власної нотації. Оскільки діаграми класів можуть відображати об'єкти, то діаграма класів, на якій відображено лише об'єкти, та не відображено класи, може вважатись діаграмою об'єктів.

Діаграма об'єктів відображає об'єкти та зв'язки в певний момент роботи програми. Об'єкти можуть містити інформацію про власні значення а не про описання. Для відображення загальних шаблонів об'єктів та зв'язків, що можуть багаторазово створюватись під час роботи програми, слід використовувати діаграму взаємодії, яка може відображати характеристики об'єктів та зв'язків. Екземпляр діаграми взаємодії створює діаграму об'єктів.

Діаграма об'єктів не відображає еволюцію системи під час роботи. Натомість, слід використовувати діаграми взаємодії з повідомленнями, або діаграми послідовності.

Діаграма розгортання (deployment diagram) це діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму FEAL – блоковий шифр, запропонований Акіхіро Симідзу і Седзі Міягуті.

У ньому використовуються 64-бітовий блок і 64-бітовий ключ. Його ідея полягає і в тому, щоб створити алгоритм, подібний DES, але з більш сильною

функцією етапу. Використовуючи менше етапів, цей алгоритм міг би працювати швидше. На жаль, дійсність виявилася далекою від цілей проекту.

Як вхід процесу шифрування використовується 64-бітовий блок відкритого тексту. Спочатку блок даних підлягає операції XOR з 64 бітами ключа. Потім блок даних розщеплюється на ліву і праву половини. Об'єднання лівої і правої половин за допомогою XOR утворює нову праву половину. Ліва половина і нова права половина проходять через N етапів (спочатку 4). На кожному етапі половина об'єднується за допомогою функції F[1] з 16 бітами ключа і за допомогою XOR – з лівою половиною, створюючи нову праву половину. Вихідна права половина (на початок етапу) стає новою лівою половиною. Після N етапів (ліва і права половини не переставляти після N-го етапу) ліва половина знову об'єднується з допомогою XOR з правою половиною, утворюючи нову праву половину, потім ліва і права об'єднуються разом в 64-бітове ціле. Блок даних об'єднується за допомогою XOR з іншими 64 бітами ключа і алгоритм завершується.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ визначення контексту користувача на основі архітектури CoDA яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: З’єднання; Ресурси; Журнал; Параметри; Довідка.

– Розділу виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

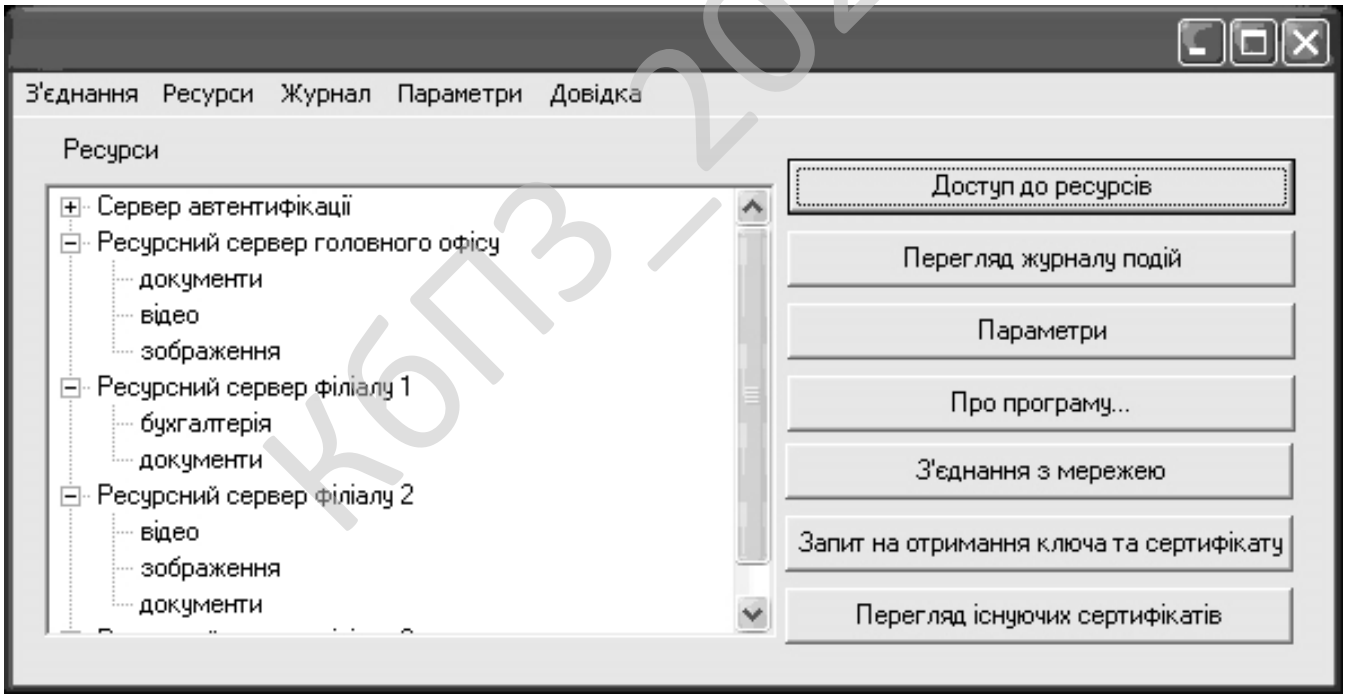

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і інтуїтивно зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним

середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

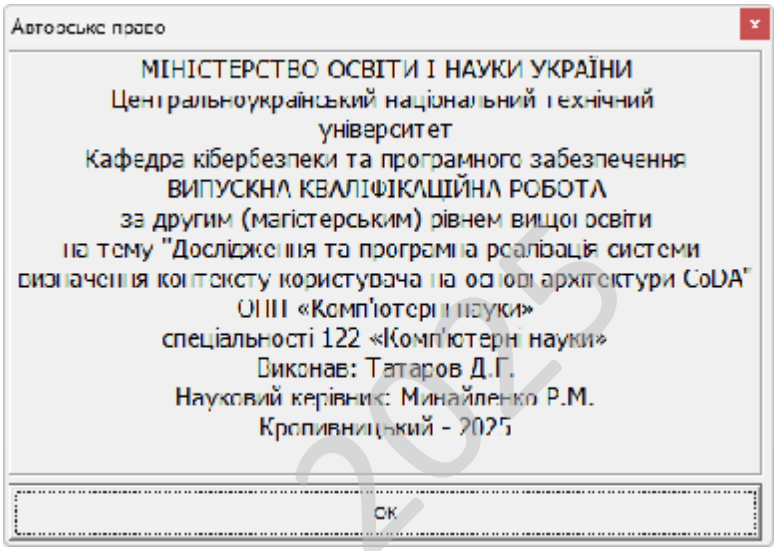

Рисунок 5.2 – Авторське право

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних

вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати ІТ рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене ІТ рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути

невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування чорної скриньки. Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

– Помилок характеристик (необхідна ємність пам'яті і т.д.);

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – proprietary software.

Програмне забезпечення, на яке зберігаються як немайнові, так і майнові авторські права. Отримавши або придбавши таке програмне забезпечення, користувач отримує обмежені права користування ним: може бути заборонено або закрито доступ до коду (вивчення), внесення змін, тиражування, розповсюдження та перепродаж. Програмне забезпечення вважається власницьким, якщо наявне хоча б одне з перелічених обмежень.

Найчастіше основним методом захисту майнових прав на власницьке ПЗ, поза ліцензійною угодою, власник обирає закриття сирцевого коду, захищаючи свій продукт від модифікації і вбудовуючи системи обмеження користування через авторизацію. Таке програмне забезпечення називається закритим. Проте, код власницького продукту може бути і відкритим, але власник може обмежити права користувача умовами користувацької ліцензії.

Власницьке програмне забезпечення та комерційне програмне забезпечення не є синонімами – власницьким може бути і безплатне (тобто, некомерційне) програмне забезпечення.

На противагу власницькому ПЗ існує вільне програмне забезпечення, автори і власники якого дозволяють вивчати, модифікувати і поширювати свій продукт. Саме визначення власницького програмного забезпечення виникло в результаті діяльності громадського руху вільного програмного забезпечення (представленого Фондом вільного програмного забезпечення та іншими організаціями) і осмислення умов свободи користування програмами. Визначенням власницького програмного забезпечення є не невідповідність хоча б одній з базових умов вільного програмного забезпечення. Сама назва власницьке ПЗ підкреслює визначальне значення власника у способі використання і можливостях розвитку цього програмного забезпечення.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи визначення контексту користувача на основі архітектури CoDA.

Метою розробки є дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA.

Об'єктом дослідження є процес визначення контексту користувача на основі архітектури CoDA.

Предметом дослідження є методи визначення контексту користувача на основі архітектури CoDA.

Методи дослідження базуються на методах теорії комп'ютерних мереж, теорії захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод визначення контексту користувача на основі архітектури CoDA.

– Розроблено вітчизняний продукт визначення контексту користувача на основі архітектури CoDA, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та розробки системи визначення контексту користувача на основі архітектури CoDA можуть бути насамперед корисними для підприємств, які мають розгалужену ІТ-інфраструктуру й використовують сервери, мережеве обладнання та корпоративні сервіси для підтримки своєї діяльності. Для таких компаній стабільність і безперебійність роботи інформаційних систем є критично важливими, тому можливість своєчасного виявлення несправностей або перевантажень стає суттєвою конкурентною перевагою. Саме система моніторингу допомагає контролювати роботу мережевих пристроїв у режимі реального часу, виявляючи проблеми ще до того, як вони вплинуть на користувачів.

Особливий інтерес до таких систем можуть проявити ІТ-компанії, які займаються наданням послуг хостингу, розробкою програмного забезпечення або підтримкою клієнтів. Для них швидкість реагування на інциденти та якість технічного обслуговування є показниками репутації, а отже, від роботи системи моніторингу залежить рівень довіри клієнтів і лояльність користувачів. Такі підприємства часто працюють у середовищі, де навіть хвилинна затримка чи зупинка сервера призводить до фінансових збитків, тому автоматизація контролю за станом мережі – це не розкіш, а необхідність.

Крім комерційних компаній, результати дослідження будуть актуальними для державних структур, освітніх установ і організацій, які мають внутрішні мережі та зберігають великі обсяги інформації. У таких установах впровадження системи моніторингу підвищує ефективність роботи ІТ-відділів, зменшує ризик втрати даних і допомагає раціонально використовувати наявні ресурси.

Не менш важливим є значення цієї розробки для навчальних і наукових закладів. Вони можуть використовувати систему як навчальну платформу для підготовки фахівців у сфері інформаційних технологій. Студенти отримають можливість не лише спостерігати за реальною роботою системи моніторингу, а й аналізувати дані, моделювати різні ситуації та вчитися реагувати на інциденти. Таким чином, результати дослідження мають універсальний характер і можуть бути впроваджені як у бізнесі, так і в освіті.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості програмного продукту було проведено експертне опитування серед фахівців у галузі IT-інфраструктури, адміністраторів систем і представників компаній, що мають досвід використання схожих рішень. Експертам було запропоновано оцінити систему за основними критеріями – функціональні можливості, надійність, простота впровадження, масштабованість, вартість експлуатації та потенційна економічна ефективність.

Більшість експертів високо оцінили саме інтелектуальну частину системи – можливість автоматичного сповіщення про інциденти, генерацію аналітичних звітів і прогнозування потенційних відмов обладнання. Особливо було відзначено, що система працює стабільно навіть при великому навантаженні й може адаптуватися до різних типів мережевої інфраструктури, що робить її універсальною.

За результатами оцінки середній рівень привабливості продукту склав 8,7 бала з 10 можливих. Експерти зазначили, що така система може мати великий попит серед середніх і великих підприємств, особливо якщо її вартість залишатиметься конкурентною. Також було підкреслено, що простота інтерфейсу та можливість кастомізації під конкретного користувача є суттєвими перевагами, які підвищують комерційний потенціал рішення.

Таким чином, метод експертних оцінок показав, що система має високу ринкову привабливість, відповідає актуальним потребам бізнесу та може стати успішним продуктом за умови належного маркетингового просування та підтримки користувачів.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості розробки системи визначення контексту користувача на основі архітектури CoDA доцільно використовувати витратний метод. Він передбачає визначення всіх фактичних витрат, які були понесені під час створення програмного продукту, включаючи оплату праці розробників, витрати на апаратне забезпечення, ліцензії, тестування та впровадження. Такий підхід дозволяє точно визначити базову собівартість проєкту, що є особливо важливим для невеликих команд і стартапів.

Однак, у випадку комерційного впровадження, доцільно поєднати цей підхід із дохідним методом. Дохідний метод дає змогу оцінити майбутні вигоди, які підприємство отримає після впровадження системи. Наприклад, скорочення простоїв серверів, підвищення ефективності роботи персоналу та зменшення витрат на ручну діагностику мережі є прямими джерелами економічної вигоди.

Такий комбінований підхід дозволяє не лише визначити початкову вартість розробки, а й обґрунтувати економічну доцільність проєкту. Він допомагає потенційним інвесторам побачити не просто витрати, а реальні фінансові перспективи, які відкриває впровадження системи.

У результаті використання комбінованої моделі оцінки можна отримати повну картину вартості та окупності проєкту, що стане основою для прийняття управлінських рішень щодо його реалізації чи масштабування.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Компанія має розгалужену IT-інфраструктуру, яка включає сервери, мережеве обладнання, робочі станції, системи зберігання даних і корпоративні сервіси. До впровадження системи моніторингу контроль за станом мережі здійснювався вручну: адміністратори виявляли проблеми лише після звернень користувачів або повного виходу сервісів із ладу. Це призводило до простоїв, затримок у роботі та фінансових втрат. Основна мета впровадження системи мережевого моніторингу – забезпечити цілодобове автоматичне відстеження стану обладнання, серверів і додатків, оперативне реагування на інциденти, зниження кількості простоїв і запобігання критичним збоям у роботі IT-інфраструктури. Вхідні дані зафіксовано в таблиці 7.1.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	До впровадження	Після впровадження	Економічний ефект
Кількість простоїв серверів на рік	20 випадків	5 випадків	–15
Середня тривалість простою одного сервера	4 години	1 година	–3 години
Середні втрати підприємства за 1 годину простою	25 000 грн	5 000 грн	–20 000 грн
Витрати на ручну діагностику й усунення збоїв	300 000 грн/рік	150 000 грн/рік	–150 000 грн
Вартість впровадження системи моніторингу	—	—	450 000 грн
Річні витрати на підтримку системи	—	—	100 000 грн

Розрахунок економічного ефекту демонструє наступне: зменшення збитків від простоїв – 1 975 000 грн/рік, економія на технічному обслуговуванні – 150 000 грн/рік, сукупний річний ефект – 2 125 000 грн/рік, чистий ефект – 2 025 000 грн/рік, термін окупності (Payback Period) – 0,22 року (~2,5 місяці), коефіцієнт ефективності (ROI) – 450%.

Додаткові (немонетарні) переваги: підвищення стабільності IT-інфраструктури завдяки ранньому виявленню збоїв, зменшення навантаження на IT-персонал через автоматизацію моніторингу, покращення SLA (Service Level Agreement) і задоволеності користувачів, прогнозування потенційних проблем через аналітику та звітність у реальному часі, зростання репутації підприємства, адже мінімізуються ризики затримок у наданні послуг або збою критичних бізнес-процесів.

Таким чином, моніторинг стає не лише технічним інструментом, а й важливою складовою операційної надійності та конкурентоспроможності підприємства.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Просування системи моніторингу має будуватися на поетапному підході, що включає як технічну демонстрацію, так і інформаційне просування. На першому етапі варто створити пілотний проєкт і запропонувати його впровадження у невеликій кількості підприємств для збору відгуків і реальних кейсів. Це дозволить перевірити ефективність системи в реальних умовах і створити довіру до продукту.

Далі важливо забезпечити інформаційну присутність продукту – через участь у галузевих конференціях, IT-форумах, онлайн-презентаціях і спеціалізованих публікаціях. Саме через публічну експертну комунікацію формується репутація розробника та усвідомлення цінності рішення на ринку.

Наступним етапом є розширення партнерських зв’язків. Доцільно співпрацювати з ІТ-компаніями, які займаються інтеграцією корпоративних систем, адже вони можуть пропонувати продукт своїм клієнтам як частину комплексного рішення. Водночас слід розробити гнучку цінову політику – наприклад, ліцензування за кількістю пристроїв або модель передплати, що зробить продукт доступнішим для малого та середнього бізнесу.

Просування має супроводжуватися технічною підтримкою користувачів, оновленнями та навчанням персоналу. Це створює позитивний досвід використання продукту та сприяє формуванню довгострокових відносин із клієнтами. У підсумку правильна стратегія просування допоможе не лише збільшити продажі, а й побудувати впізнаваний бренд на ринку ІТ-рішень.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту варто поєднати прямі продажі з цифровими платформами розповсюдження програмного забезпечення. Власний сайт компанії може стати не лише вітриною продукту, а й каналом комунікації з клієнтами, де вони зможуть отримати демо-версію, консультацію або підтримку. Це сприятиме зниженню витрат на маркетинг і збільшенню довіри.

Додатково ефективним буде впровадження партнерської програми для системних інтеграторів і реселерів, які вже мають доступ до корпоративних клієнтів. Така модель дозволяє розширити охоплення ринку без суттєвих додаткових інвестицій.

Також можна запропонувати гібридну форму реалізації: ліцензування для великих компаній і модель SaaS (Software as a Service) для малого бізнесу. Це підвищить доступність системи та дозволить гнучко реагувати на потреби різних сегментів ринку.

Ключовим напрямом оптимізації збуту є створення якісного сервісу після продажу – технічна підтримка, регулярні оновлення, аналітичні звіти. Усе це забезпечує стабільність роботи клієнта й стимулює його до подальшої співпраці.

7.7 Визначення ключових факторів успіху конкретного проєкту

Основним фактором успіху є стабільність і надійність системи. Якщо система моніторингу працює без збоїв і забезпечує реальну користь, вона швидко здобуває довіру користувачів. Технологічна якість продукту, його здатність масштабуватися й інтегруватися з іншими IT-рішеннями відіграють ключову роль у його життєздатності.

Другим важливим чинником є професійна команда розробників і технічної підтримки. Клієнти цінують не лише продукт, а й можливість отримати швидку допомогу у випадку проблем або питань. Від рівня компетенції фахівців залежить не лише якість обслуговування, а й довгострокові відносини з партнерами.

Не менш значущим є гнучкість системи – можливість адаптувати її під специфіку кожного клієнта. Різні компанії мають різну інфраструктуру, тому універсальне, але налаштовуване рішення стає перевагою.

І, нарешті, успіх будь-якого IT-проєкту визначається здатністю постійно вдосконалюватися. Регулярні оновлення, впровадження нових технологій і зворотний зв’язок із користувачами формують довіру й підтримують актуальність продукту на ринку. Саме ці чинники разом створюють основу для стабільного розвитку та комерційного успіху системи моніторингу.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Електронно-обчислювальна машина (ЕОМ) відіграє важливу роль у житті сучасної людини. Кожного дня мільйони людей використовують ЕОМ для пошуку необхідної інформації, спілкуванні у соціальних мережах, перегляду новин, роботи тощо. Багато людей користуються ЕОМ у професійних цілях, оскільки завдяки ЕОМ з'явилося багато нових професій. Тому для розробника хмарних сервісів так важливо розробити зручний інтерфейс для зручного сприйняття інформації, та необхідний функціонал, який буде відповідати необхідним вимогам та навантаженням. Все це вимагає багато часу та великого навантаження з боку розробників. Тому так важливо слідкувати за умовами праці, в яких відбувається робочий процес. Оскільки захворювання можуть бути спричинені надмірним фізичним або розумовим навантаженням, через велику нервово-емоційну напругу, або через виробниче середовище. В даному розділі магістерської роботи проведемо аналіз основних чинників при роботі програміста.

Законом України "Про охорону праці" регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машина (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, параметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

На роботу програміста впливають наступні фактори: невідповідний мікроклімат приміщення (температура, вологість), недостатня освітленість робочої зони, підвищений рівень шуму та електромагнітного випромінювання, порушення іонного складу повітря, неправильна ергономічна організація робочого місця, ризики, пов'язані із погіршенням зору, порушенням фізичного стану, стресом тощо.

Шкідливими факторами при роботі з персональним комп'ютером є неіонізуюче випромінювання промислової частоти, збільшене нервово- емоційне навантаження на оператора, збільшення навантаження на органи зору та дрібні стереостатичні рухи кінцівок. Ці фактори можуть викликати у працівника певні розлади здоров'я, зокрема підвищення артеріального тиску, кон'юктивіти, тендовагініти та інші захворювання.

Комп'ютер, як і будь-який електричний прилад, особливо при його неправильному підключенні, може бути джерелом ураження оператора електричним струмом. Саме тому всі працівники, які працюють з персональним комп'ютером, повинні мати першу (або другу) групу допуску з електробезпеки.

Через наявність зазначених факторів працівники, які працюють з персональними комп'ютерами, підлягають попередньому та періодичному медичному огляду згідно з пунктом 6.2.3 додатку 4 до наказу Міністерства охорони здоров'я України «Про затвердження Порядку проведення медичних оглядів працівників певних категорій» від 21 травня 2007 року № 246.

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Оптимальна температура в приміщенні для праці має становити 20-24°С, відносна вологість – 40-60 %, атмосферний тиск – 750 мм. рт. ст., запиленість не повинна перевищувати 10 мг/м^3, швидкість руху повітря – 0,1 м/с.

Через те, що обчислювальна техніка є джерелом тепловиділення, організація мікроклімату потребує додаткових зусиль: кондиціонування, провітрювання, використання систем опалення тощо. Об’єм приміщень повинен передбачатися з урахуванням як мінімум 20 м^3 /на особу [4].

Монітори комп’ютерів є джерелом випромінювання, яке може зашкодити здоров’ю людини. Для убезпечення роботи з комп’ютером відстань від монітора повинна становити не менше 50 см, бажано використовувати монітори зі зниженим рівнем, скорочувати час безперервної роботи за комп'ютером (робити п’ятнадцяти хвилинні перерви після кожних півтори години праці). Також в приміщенні необхідно встановлювати іонізатори повітря, використовувати нейтралізатори та зволожувачі.

Комп’ютери та периферійні пристрої є джерелами шуму, висока інтенсивність якого може призвести до проблем з органами слуху та негативно впливати на психологічний стан. Рівень шуму на робочому місці не повинен перевищувати 50 дБА [5]. Для зменшення рівня шуму можна використовувати звукопоглинальні пристрої, а стіни приміщень з комп’ютерами можуть бути покриті звукопоглинальними матеріалами. Поряд із шумом часто виникає вібрація. Для зменшення рівня вібрації в приміщенні на поверхні необхідно встановлювати віброізолятори.

Ергономічні показники робочого місця програміста мають бути наступними: висота робочої поверхні повинна складати 720 мм, розмір поверхні має становити 1600 х 1000 мм; під столом повинен бути простір з розмірами по глибині 650 мм; стіл повинен мати підставку для ніг, розташовану під кутом

15°до поверхні; відстань клавіатури від краю столу має бути не більше 300 мм; відстань між очима й екраном повинна складати 40 – 80 см; стілець повинен мати підйомно-поворотний механізм; висота сидіння має регулюватися в межах 400 – 500 мм, глибина – не менше 380 мм, а ширина – не менше 400 мм, висота опорної поверхні спинки має бути не менше 300 мм, ширина – не менше 380 мм. Кут нахилу спинки стільця до площини сидіння повинен змінюватися в межах 90 – 110° [6].

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер Prinics PicKit M1 Smartphone Photo Printer White, електродвигуни вентиляторів ЕОМ.

Робота програміста передбачає постійний візуальній контакт з моніторами комп'ютерів, та, як наслідок, значне навантаження на зір. Традиційно, це зорова робота високої або середньої точності. Для зорової роботи високої точності загальне освітлення (розподіл світла у всьому об'ємі приміщення) має становити 300 лк, комбіноване освітлення (поєднання загального і місцевого освітлення) – 750 лк. Штучне освітлення повинно бути рівномірним та використовуватися в світлий і темний час доби. Джерелами штучного освітлення можуть слугувати люмінесцентні лампи. Правильне освітлення передбачає уникнення відблисків на екранах.

З 2019 року діють Державні будівельні норми України "Природне і штучне освітлення" – ДБН В.2.5-28:2018 [4], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018 [4], можна віднести до роботи з малою точністю (найменший

розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [4], Крім того все поле зору повинно бути освітлено достатньо рівномірно – це основна гігієнічна вимога. Оскільки яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень;

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який

повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.5 Розрахункова частина

Проведемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 6 м, довжина – 7 м, висота – 2,9 м.

У зазначеному приміщенні працює 4 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою [1]:

$$F = E \cdot S \cdot K \cdot Z / n,$$

де:

F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа освітлюваного приміщення (у нашому випадку S=6 х 7 = 42 м^2);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників у процесі експлуатації (його значення

залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку $K = 1,5$);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, у нашому випадку $Z = 1,1$);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку від усіх ламп і обчислюється в долях одиниці [8]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін} = 50\%$ і $\rho_{стелі} = 50\%$.

Обчислимо індекс приміщення за формулою:

$$i = S/(h\cdot (A+B)),$$

де:

S – площа приміщення, $S = 42$ м2;

h – розрахункова висота підвісу, $h = 2,9$ м (співпадає з висотою стелі, оскільки лампи освітлення закріплюються на стелі);

A – ширина приміщення, $A = 6$ м;

B – довжина приміщення, $B = 7$ м.

Підставимо всі значення у формулу та визначимо індекс приміщення:

$$i=1,4.$$

Знаючи індекс приміщення, за знаходимо $n = 0,29$ (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [8]. Підставимо всі значення у формулу, визначимо світловий потік: F=71689 Лм.

Для розрахунку будемо використовувати світлодіодні стельові панелі Delux LED Panel 41 44 Вт, світловий потік яких $F_л = 3600$ Лм.

Число ламп визначається за формулою:

$$N=F/F_л$$

де:

F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначимо індекс приміщення:

$$N= 71689 / 3600 =19,9 \text{ шт.}$$

Приймаємо необхідну кількість світлодіодних світильників 20 шт.

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з умов поліпшення охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи визначення контексту користувача на основі архітектури CoDA.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів визначення контексту користувача на основі архітектури CoDA.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем визначення контексту користувача на основі архітектури CoDA.

– Досліджена система визначення контексту користувача на основі архітектури CoDA.

– На основі отриманих результатів досліджень створена програмна реалізація системи визначення контексту користувача на основі архітектури CoDA.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання визначення контексту користувача на основі архітектури CoDA.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм FEAL.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Татаров Д.Г. Дослідження та програмна реалізація системи визначення контексту користувача на основі архітектури CoDA // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

2. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

3. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

4. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

5. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

6. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

7. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

8. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

9. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

10. Петрик В.М., Присяжнюк М.М., Аль-Файюмі Халед та ін. «Системи інформаційної зброї та технології інформаційної війни»: підручник / Петрик В.М., Присяжнюк М.М., Аль-Файюмі Халед, Жарков Я.М., Смірнов О.А, Буравченко К.О., Давидюк А.В., Кононович В.Г., Корчинский В.В., Кудирко В.М., Фесенко А.О.; за заг. ред. В.М. Петрика, М.М. Присяжнюка.– К.: Видавничий центр “Кафедра”, 2025.– 320 с.

11. Усік, П.С., Смірнова, Т.В., Буравченко, К.О., Смірнов, О.А., Улічев, О.С., Смірнов, С.А. «Дослідження технологій забезпечення кібербезпеки банківських систем з використанням штучного інтелекту». *Кібербезпека: освіта, наука, техніка.* 2025. Том 1 № 29. С.704–716, 2025

12. Kuznetsov, O., Frontoni, E., Kuznetsova, K., Arnesano, M., Smirnov, O. «A secure biometric authentication architecture for blockchain-driven cyber-physical systems». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 193–224.

13. Kuznetsov, O., Atzeni, G., Arnesano, M., Randieri, C., Smirnov, O. «Secure IoT-based smart wheelchair system: From implementation to security enhancement strategy». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 225–257.

14. Kuznetsov, O., Smirnov, O., Kuznetsova, T., Shaikhanova, A., Svatowsky, I. «Privacy-utility trade-offs in IoT networks: A comparative analysis of differential privacy mechanisms for sensor data aggregation». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 589–622.

15. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

16. Kuznetsov O., Frontoni E., Kuznetsova Y., Smirnov O., Moskovchenko I. «Trust-Based Security Architecture for Edge Computing: A Simulation Study of Dynamic Trust Evolution and Attack Detection». *CEUR Workshop Proceedings,* 2024, 3909, pp. 227–241.

17. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

18. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous

Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

19. Ткаченко, О., Ільєнко, А., Улічев, О., Мелешко, Є., Смірнов, О. «Правові засади поширення інформаційних впливів в соціальних мережах». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 170–188.

20. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

21. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

22. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

23. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

24. Akhalaia, G., Iavich, M., Iashvili, G., Prysiazhnyy, D., Smirnova, T. «Secure Encrypted Connection on Georgian Website». *CEUR Workshop Proceedings,* 2023, 3550, pp. 313-320.

25. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56

26. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

27. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

28. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

29. Смірнова Т.В., Гнатюк С.О., Бердибаєв Р.Ш., Сидоренко В.М., Жигаревич О.К., «Система корелювання подій та управління інцидентами кібербезпеки на об'єктах критичної інфраструктури». *Кібербезпека: освіта, наука, техніка*, №*3*(19), 2023, С. 176-196.

30. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *ІІ Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

31. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

32. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя*

кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

33. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 2(72), С. 170-178.

34. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

35. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

36. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

37. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

38. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

39. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

40. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

41. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

42. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

43. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

44. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

45. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

46. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

47. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

48. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

49. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

50. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

51. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

52. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.