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
“____” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи інтелектуального Software-Defined Power для дата-центрів”

Виконав здобувач вищої освіти
IV курсу, групи КН-22
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Засядьвовк Б.Д.
«____» _____ 2026 р.

Керівник проекту
кандидат технічних наук
_____ Лисенко І.А.
«____» _____ 2026 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“Інформаційні технології”*_____

Спеціальність 122 *“Комп’ютерні науки”*

Освітньо-професійна (освітньо-наукова) програма _*“Комп’ютерні науки”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 06 » лютого 2026 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Засядьовку Богдану Дмитровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи інтелектуального Software-Defined Power для дата-центрів*

2. Керівник роботи *Лисенко Ірина Анатоліївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 116-02 від 06.02.2026 року

3. Строк подання студентом роботи до захисту *23.05.2026 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи інтелектуального Software-Defined Power для дата-центрів*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 06 » лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2026 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2026 р.	
3.	Розробка моделі компонента	20.03.2026 р.	
4.	Розробка структур даних	25.03.2026 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2026 р.	
6.	Програмування алгоритмів	10.04.2026 р.	
7.	Оформлення ПЗ	17.04.2026 р.	
8.	Попередній захист роботи	23.05.2026 р.	

Дата видачі завдання
« 06 » лютого 2026 р.

Підпис керівника
_____ Лисенко І.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 06 » лютого 2026 р.

Підпис здобувача
_____ Засядьвовк Б.Д.
(прізвище та ініціали)

АНОТАЦІЯ

Засядьвовк Б.Д. Програмне забезпечення системи інтелектуального Software-Defined Power для дата-центрів. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2026.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи інтелектуального Software-Defined Power для дата-центрів.

Метою розробки є програмне забезпечення системи інтелектуального Software-Defined Power для дата-центрів.

Результат роботи – програмна реалізація системи інтелектуального Software-Defined Power для дата-центрів.

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 Python.

Ключові слова: комп'ютерні науки, інтелектуальний Software-Defined Power, дата-центр

ABSTRACT

Zasiadvovk B.D. Software for the intelligent Software-Defined Power system for data centers. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the intelligent Software-Defined Power system for data centers.

The purpose of the development is the software for the intelligent Software-Defined Power system for data centers.

The result of the work is the software implementation of the intelligent Software-Defined Power system for data centers.

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

A convenient user interface has been developed. Instructions for working with software are provided.

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

The program was developed in the Python environment.

Keywords: computer science, intelligent Software-Defined Power, data center

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

EOM	–	електронно-обчислювальна машина
ATM	–	Asynchronous Transfer Mode – асинхронний спосіб передачі даних
BER	–	Bit Error Rate – імовірність ушкодження одного біта
CBWFQ	–	class based weighted fair queueing
DiffServ	–	Differentiated Services – механізм який залежно від вимог до якості обслуговування записує в IP заголовки пакетів спеціальні мітки
DSCP	–	DiffServ CoSde Point
ICMP	–	Internet CoSntrol Message Protocol – міжмережний протокол управляючих повідомлень
ISP	–	Internet Service Provider – провайдер
LLQ	–	low latency queueing
MPLS-TE	–	Multiprotocol Label Switching Traffic Engineering
PQ	–	priority queueing
RIO	–	RED with Input Output
RTT	–	Round Trip Time – час між відправкою запиту та отриманням відповіді
STM	–	Synchronous Transfer Mode – синхронний спосіб передачі даних
QoS	–	Quality of Service – якість обслуговування
WFQ	–	Weighted Fair Queuing – механізм планування пакетних потоків даних
WRED	–	Weighted random early detection – взвішане значення довжини черги, у якості фактора, визначаючого ймовірність відкидання пакета

ВСТУП

Актуальність теми. Забезпечення безперебійної й гарантованої подачі електроенергії до ІТ-устаткування є постійною турботою керівників ЦОД. Однак існуючі інфраструктури електропостачання нерідко неефективні й задіються лише частково, оскільки спроектовані й побудовані для задоволення пікових потреб. Рішення, що з'являються на ринку, які часто йменують Software-Defined Power, дозволяють істотно підвищити ефективність використання інфраструктури й ресурсів електроживлення.

ІТ-світ стає програмно-визначаємим (Software-Defined, SD). Свій переможний хід це поняття початло з мережних інфраструктур, що вступили на шлях відділення «перемелючих» бітів трафіку комутаторів від засобів контролю й керування цим процесом. Після появи програмно-визначаємих засобів зберігання (СЗД) стало можливим створення повністю програмно-визначаємих центрів обробки даних (Software-Defined Data Center, SD-DC). Якщо говорити коротко, у таких ЦОД реалізований якийсь рівень абстракції, що дозволяє гнучко використовувати наявні ІТ-ресурси (сервери, СЗД, мережа) відповідно до поточних потреб конкретних застосунків і бізнес-процесів. При цьому ресурси споживаються максимально ефективно, а застосунку одержують оптимальні для їхньої роботи сервіси, які адаптуються до завдань, що змінюються.

Слідом за ІТ-інфраструктурою принципи «програмної визначаємости» почали поширюватися й на інші системи. У контексті ЦОД у першу чергу цікаво їхнє застосування до інженерних систем, що забезпечують роботу ІТ. І от у професійному співтоваристві всі частіше стало згадуватися поняття Software-Defined Power, або програмно-визначаєме електроживлення.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи інтелектуального Software-Defined Power для дата-центрів.

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

– Огляд існуючих систем інтелектуального Software-Defined Power для дата-центрів.

– Дослідження системи інтелектуального Software-Defined Power для дата-центрів.

– Програмна реалізація системи інтелектуального Software-Defined Power для дата-центрів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі інтелектуального Software-Defined Power для дата-центрів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи інтелектуального Software-Defined Power для дата-центрів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

За аналогією з іншими системами SD, мова йде про формування рівня абстракції, що дозволяє ефективно управляти наявними ресурсами (у цьому випадку електроживлення) в інтересах кінцевих «користувачів», якими – як і для всього SD-DC – є різні додатки. Концепція Software-Defined Power реалізується на рівні ПЗ керування – наприклад, системи класу DCIM, що здатна задіяти наявні системи керування IT і інженерним устаткуванням, а також автоматизувати операційні процедури. Головний результат – оптимізація розподілу й виділення ресурсів електроживлення в одному ЦОД або в мережі з декількох ЦОД з урахуванням поточних вимог і пріоритету різних застосунків.

Важливою особливістю систем Software-Defined Power є координація «дій» IT- і інженерної інфраструктур. У випадку мережі з декількох територіально розподілених ЦОД така координація дозволяє оперативно переводити IT-навантаження (наприклад, переміщати віртуальні машини) між ЦОД для підвищення надійності, економії електроенергії або для виконання планових або аварійних процедур обслуговування. Це дозволяє гарантувати безперервну (24x7) роботу застосунків, навіть якщо локальні джерела електроживлення не самі надійні.

Дотримуючись принципу «слідом за місяцем», можна досягти істотної економії. Вартість електроенергії в більшості країн миру залежить від багатьох факторів, у тому числі від часу доби. Так, у Україні оптові ціни на електроенергію міняються щогодини: пікові значення доводяться на період з 9 до 13 год, а в нічні годинники її вартість істотно нижче. Тому, при наявності технічної можливості управляти споживанням ЦОД, можна істотно заощадити, скоротивши витрати в годинники максимальної вартості електричної енергії. Для

цього можна переводити IT-ресурси з одного ЦОД в інший «слідом за місяцем», відключаючи сервери в місцях, де в цей момент спостерігається пікова вартість електрики. Такі можливості також відносять до функціонала систем Software-Defined Power.

Багато експертів розглядають технології Software-Defined Power як розширення концепції SD-DC для зниження впливу можливих проблем з електроживленням на роботу застосунків. Завдання систем Software-Defined Power полягають у тому, щоб у кожний конкретний момент часу підключати IT-навантаження (застосунку) до найбільш надійному й економічно вигідного (знов-таки в цей момент) ресурсу електроживлення. Очевидно, цього можна домогтися як перемикаючи джерела електроживлення без фізичного переміщення IT-навантаження, так і переводячи IT-навантаження в оптимальне (з погляду забезпеченості електроживленням) місце: в іншу стійку, в інший зал або ЦОД.

Нове покоління програмних засобів дозволяє профілювати використання ресурсів електроживлення й виділяти їх з урахуванням пріоритетів IT-устаткування й застосунків. Це дає можливість нарощувати системи обробки й зберігання даних (сервери й СЗД) без збільшення потужності енергопостачання, що забезпечує значну економію засобів за рахунок відсутності необхідності підтримки надлишкової потужності.

1.2 Область застосування

Як показує аналіз доступної в Інтернеті інформації, першої концепцію й рішення Software-Defined Power запропонувала компанія Power Assure ще в 2007 році. Однак в 2014 році ця компанія закрилася. Згодом термін Software-Defined Power стосовно до своїх розробок стали використовувати багато хто, включаючи 3DFS, GridBridge, CUI, Virtual Power Systems і інші. Однак єдиного визначення й набору функціональності для таких систем поки не вироблено. Можливо, це

пов'язане з тим, що гіганти галузі, такі як Schneider Electric, поки не замічені в активному просуванні компанії Software-Defined Power. Разом з тим комплексні системи, що поставляються великими компаніями, електроживлення вкупі із системами керування категорії DCIM (Data Center Infrastructure Management) можуть багато чого (а іноді й значно більше) з того, що проголошують адепти Software-Defined Power.

Але повернемося до продуктів, позиціонуємим як рішення Software-Defined Power. Щоб більш детально зрозуміти їхні можливості, розглянемо як приклад апаратно-програмний комплекс ICE – спільне дітище компаній CUI і Virtual Power Systems. Як обіцяють постачальники, за рахунок оптимізації використання ресурсів електроживлення це рішення дозволяє знизити операційні витрати на 15-25%, але що ще більш цікаво – на 40–60% скоротити капітальні витрати.

Основою системи є розроблене компанією Virtual Power Systems ПЗ керування ICE Software Application Suite. Рішення припускає установку в ЦОД і додаткових апаратних елементів. Так, у кожній стійці, що буде охоплена системою Software-Defined Power, установлюється блок ICE Block висотою 4U (можлива установка таких блоків для всіх стійок в окремій шафі наприкінці ряду). Одне із завдань ICE Block – зняття пікових навантажень без необхідності збільшення загальної потужності системи енергопостачання ЦОД. Це досягається завдяки встановленим в ICE Block літій-іонним батареям.

У періоди підвищеного споживання з боку встановленого в стійку IT-устаткування додаткову електроенергію забезпечують локальні АКБ. Коли ж споживання падає, батареї підзаряжаються. Відповідно до твердження постачальників, таке рішення здатне забезпечити роботу IT-устаткування сумарною потужністю 16 кВт на стійку, при цьому централізовану систему електроживлення ЦОД досить спланувати з розрахунку навантаження всього 8-10 кВт на стійку.

Таким чином, загальну потужність підключення до енергосистеми можна знизити більш ніж у півтора разу. Крім того, загальна потужність, на яку розрахована електрична інфраструктура ЦОД, буде нижче, а виходить, і витрати на таку інфраструктуру виявляться значно менше. На іншій чаші ваг – необхідність покупки системи ICE, а також виділення чотирьох юнітів (4U) дорогоцінні простори в кожній стійці (або установки додаткової стійки під пристрої ICE Block наприкінці ряду).

Другий апаратний елемент системи ICE – невеликий перемикач ICE Switch, що, власне кажучи, і «диригує» підключенням і відключенням різних джерел електрики залежно від поточних потреб, вартості одержання енергії від конкретного джерела в цей момент і т.д. ICE Switch виконаний у вигляді компактного пристрою (висотою 1U) і складається із двох ідентичних модулів (схема резервування 2N) з можливістю їх «гарячого» відключення/підключення.

При підвищенні споживання в пікові періоди ICE Switch може, зокрема, включати дизель-генератори. Якщо звичайно ДГУ використовуються тільки як резервне джерело живлення у випадку аварії основного, то система ICE задіє їх для виробітку додаткової енергії в пікові періоди. Поряд з використанням АКБ у складі ICE Block це дозволяє знизити загальну потужність підключення ЦОД.

ICE Switch здатний підключати й альтернативні джерела електроживлення – наприклад, сонячні батареї або вітрові генератори (звичайно, у випадку їхньої наявності). Інтелектуальна система керування може прогнозувати, коли оптимально включити те або інше джерело: наприклад, сонячну батарею – у безхмарний літній день, а вітровий генератор – уночі. На даний момент альтернативні джерела, звичайно, не здатні повністю забезпечити енергетичні потреби ЦОД, тому їхнє використання варто розглядати тільки як доповнення до основного енергопостачання.

До речі, саме в жаркі літні дні підвищується споживання системи охолодження, а в такі дні й сонячні батареї генерують максимальну потужність, тому їхня допомога буде як не можна до речі.

Розроблювачі приводять безліч привабливих сценаріїв застосування системи Software-Defined Power. Один із них пов'язаний із пріоритизацією навантаження. Сьогодні більшість великих ЦОД будують за схемою резервування 2N, щоб гарантувати замовникам високий показник доступності. Це значить, що вся інфраструктура системи безперебійного гарантованого електроживлення ЦОД проектується й будується «з надлишком». Однак далеко не всім застосункам потрібно настільки високий рівень доступності. Комутатор ICE Switch, що працює у зв'язуванні з ПЗ ICE Software Application Suite, дозволяє задіяти надлишкові (зарезервовані для забезпечення схеми 2N) ресурси для підтримки систем, яким досить рівня 1N.

Приведемо приклад ЦОД із критичним навантаженням (2N) потужністю 400 кВт і менш важливим навантаженням (1N) потужністю 100 кВт. Щоб реалізувати схему 2N, необхідно організувати два незалежних промені електроживлення, причому на кожному забезпечити можливість подачі 400 кВт для критичного навантаження. При наявності систем керування електропостачанням частина потужності можна перерозподілити для живлення навантаження 1N. Тоді в штатному режимі по кожному промені буде подаватися по 250 кВт.

У випадку аварії на одному із променів низькопріоритетного навантаження прийдеться відключити, але критичне навантаження буде забезпечена повністю (400 кВт). Така схема дозволяє забезпечити нормальне функціонування ЦОД (з урахуванням пріоритету навантаження) без зайвої надмірності по електроживленню, що знов-таки означає зниження витрат.

У цілому в концепції Software-Defined Power, звичайно, утримується неабияка частка маркетингу. По суті, ця концепція означає подальший розвиток

систем керування електроживленням зі збільшенням обсягу збираємих даних (про поточну ситуацію), залученням більше ефективних засобів їхнього аналізу й прогнозування. І звичайно, як показано на прикладах вище, для реалізації Software-Defined Power необхідні досить «просунуті» (інтелектуальні) елементи інфраструктури електроживлення, без них ніяке ПЗ не буде здатно оптимізувати використання наявних ресурсів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи інтелектуального Software-Defined Power для дата-центрів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

NSX – платформа віртуалізації й забезпечення безпеки мережі

Рішення VMware NSX дозволяє повністю відтворити фізичну мережу програмним методом незалежно від базового устаткування. Програмний підхід до створення мереж допомагає досягати нових рівнів адаптивності, безпеці й економії, які були немислимі при використанні фізичних мереж

Сценарії використання VMware NSX

Безпека

Мікросегментація ЦОД і налаштування політики безпеки для кожного сегмента виключають горизонтальне поширення погроз.

Автоматизація

Автоматизація служб мережі й безпеки для миттєвої ініціалізації мереж і усунення «вузьких» місць, типових для фізичних мереж.

Безперервна робота застосунків

Розміщення й перенос робочих навантажень за пару мінут, без переривання роботи застосунків і без взаємодії з фізичною мережею.

Основні можливості NSX

– **Логічні комутатори** – всі можливості комутації на рівнях 2 і 3 у віртуальному середовищі, відділені від базового устаткування.

– **Шлюз NSX:** шлюз рівня 2 для оптимального підключення до фізичних робочих навантажень і існуючим віртуальним локальним мережам.

– **Логічні маршрутизатори:** розташовані між комутаторами для динамічної маршрутизації в різних віртуальних мережах.

– **Міжмережевий екран:** розподілений брандмауер з лінійною продуктивністю ядра, обліком ідентифікаційних даних і віртуалізації, а також можливостями моніторингу процесів.

– **Балансувальник навантаження:** всі можливості балансування навантаження пристроїв, що працюють по протоколі SSL.

– **Логічна мережа VPN:** підключення різних середовищ друг до друга й віддалений доступ через програмну мережу VPN.

– **API-інтерфейс NSX на базі REST:** інтеграція з будь-якою платформою керування хмарою.

Переваги NSX:

– Повне відділення від устаткування фізичної мережі й скорочення витрат.

– Скорочення часу ініціалізації мережі з декількох днів до декількох секунд.

– Підвищення експлуатаційної ефективності за рахунок автоматизації.

– Розміщення й перенос робочих навантажень незалежно від фізичної топології.

– Можливість розгортання на будь-якому гіпервізорі й використання на будь-якій платформі керування хмарою.

– Інтеграція сторонніх мережних рішень і засобів забезпечення безпеки через стандартні API-інтерфейси.

– Розгортання мережі без порушення роботи в існуючих фізичних мережах і топологіях наступного покоління.

VMware Horizon 7 – платформа віртуалізації робочих місць

VMware Horizon 7 надає кінцевим користувачам централізований і безпечний доступ до віртуальним або розміщеним у віддаленому середовищі комп'ютерам, застосункам і веб-службам через єдину робочу область. Це забезпечує безпеку, зручність і однаковість робітничого середовища користувачів незалежно від ОС або пристроїв.

Необхідні додатки й служби надаються на вимогу одним натисканням кнопки, що забезпечує високу продуктивність праці й допомагає підвищити мобільність і конкурентоспроможність бізнесу

Основні можливості VMware Horizon 7

Єдина цифрова робоча область

Horizon дозволяє об'єднати всі корпоративні додатки компанії, у тому числі застосунку, розміщені на вузлах RDS, і застосунку Citrix XenApp у єдину робочу область. Кінцеві користувачі одержують швидкий доступ до всіх необхідних інструментів з будь-яких пристроїв, а IT-Відділ може консолідувати, адмініструвати й захищати обчислювальні ресурси в реальному часі через єдину консоль керування

Повноцінне адаптивне середовище для однакових умов роботи

Horizon 7 включає набір технологій, що забезпечують оптимальні умови роботи кінцевих користувачів на різних пристроях, у будь-якому місці, при використанні будь-яких носіїв і підключень: від корпоративних ЛОМ до загальнодоступного wi-fi і мобільного Інтернету

Надання ресурсів по запиті з можливостями персоналізації

Віртуальні машини створюються для конкретного користувача одним натисканням кнопки й віддаляються після кожного сеансу для зниження ризиків безпеки. При цьому технологія миттєвого клонування й рішення VMware App Volumes і VMware User Environment Manager забезпечують надання персоналізованих віртуальних комп'ютерів зі збереженням застосунків, параметрів і профілів користувачів між сеансами, навіть якщо сам комп'ютер буде знищений після виходу із системи.

Підвищення рівня безпеки, надійності й керованості IT-середовищем

Застосування гнучких контекстно-контекстно-залежних політик на основі ролей, які поєднують відомості про користувача, пристрій і місце розташування. Дані політики перевіряються при вході в систему, виході з її, відключенні й повторному підключенні, а також у заздалегідь задані інтервали відновлення. IT-

Фахівці можуть вибірково, залежно від способу авторизації користувача, настроїти автоматичне включення або відключення таких можливостей, як перенапрямок буфера обміну, USB, друку й диска клієнта

Переваги Horizon 7**:**

– Співробітники можуть працювати, перебуваючи де завгодно, с будь-якого пристрою, без прив'язки до конкретного ПК, а також організовувати сеанси спільної роботи в реальному часі.

– Поліпшення умов роботи користувачів і підвищення адаптивності бізнесу.

– Автоматизація керування, підвищення рівня безпеки й надійності IT-середовища.

– Збільшення гнучкості інфраструктури й висока швидкість створення робочих місць.

– Зниження витрат на обслуговування й заміну локальних ПК і електроенергію.

– Масштабування опублікованих застосунків одним натисканням кнопки, а також прискорення їхнього розгортання в 5-10 разів.

– Автоматична синхронізація всіх змін і доповнень при наступному підключенні до мережі.

vSAN – програмне сховище даних

VMware vSAN – радикально спрощене сховище даних на основі стандартних серверів x86. vSAN дозволяє грамотно виділити обсяг пам'яті на жорстких дисках ваших серверів і не закуповувати дорогі СЗД, що сприяє зниженню сукупної вартості володіння до 50% у порівнянні із традиційними сховищами

Основні можливості vSAN:

– підвищення продуктивності – твердотільні накопичувачі (SSD) і технологія кластеризації забезпечують виконання до 6 млн. операцій уведення виводу в секунду без додаткового навантаження на ЦП і ресурси пам'яті;

– поліпшення ефективності зберігання – виключення дублювання, стиск на основі ПЗ й механізм Erasure Coding (RAID 5/6) оптимізують ємність сховища й в 7 разів скорочують обсяг даних;

– висока відказостійкість – розподілені RAID-масиви й дзеркалювання кешу для захисту від втрат даних у випадку збоїти диска, вузла, мережі або стійки;

– горизонтальне й вертикальне масштабування – збільшення ємності й продуктивності без переривання роботи за рахунок додавання нових вузлів у кластер або дисків в існуючі вузли;

– просте розгортання – сумісність із більшістю відомих ОС, з усіма мережними фабриками й маршрутизаторами. vSAN активується в парі клацань миші без установки додаткового програмного забезпечення;

– автоматизація керування – виділення ресурсів зберігання й керування рівнями обслуговування сховищ автоматизовані й виконуються на основі налаштовуємих політик, орієнтованих на віртуальні машини;

– безпека – убудована підсистема шифрування даних і підтримка дворівневої перевірки дійсності SecurID і CA (схвалено Агентством по оборонних інформаційних системах DISA).

Переваги vSAN:

– Високопродуктивне сховище даних корпоративного класу без інвестицій в «залізо».

– Незалежність від устаткування – vSan може бути розгорнуть на устаткуванні будь-яких виробників.

– Зниження сукупної вартості володіння до 50% – використовуйте існуючі сервера, купуйте диски а не масиви.

– Ініціалізація двома клацаннями миші й автоматизація виділення ресурсів і їхнього балансування.

– Проста горизонтальна й вертикальна масштабованість без простоїв у роботі.

vSphere – платформа віртуалізації **серверів**

Сценарії використання VMware vSphere:

– Консолідація обчислювальних ресурсів – підвищуйте продуктивність існуючої інфраструктури без додаткових інвестицій в «залізо».

– Ефективна віртуалізація серверів – віртуалізуйте сервери x86 і об'єднайте їх у логічні пули для виділення ресурсів декільком робочим навантаженням.

– Висока доступність – забезпечте максимальний час безперебійної роботи віртуалізованої інфраструктури, скоротите позапланові й виключите планові простої для обслуговування серверів і сховищ.

– Універсальна платформа для будь-яких застосунків – виконуйте як традиційні, так і контейнерні додатки в єдиній інфраструктурі й з існуючими робочими навантаженнями.

– Безперервність роботи застосунків – миттєво ініціалізуйте й розгортайте робочі навантаження, переносите працюючі віртуальні машини між серверами без переривання роботи користувачів і простою застосунків.

– Підтримка віддалених офісів і філій – забезпечте швидку ініціалізацію серверів і управляйте декількома IT-середовищами без розширення IT-персоналу.

Переваги vSphere:

– підвищення коефіцієнта корисного використання потужностей сервера;

– зниження витрат на керування IT-інфраструктурою на 53%;

– скорочення часу простою застосунків рівня1 на 54%;

– зниження витрат на устаткування й електроенергію;

– підвищення надійності, відказостійкості й гнучкості інфраструктури.

Оновлений постер VMware Validated Design for Software-Defined Data Center 4.2

Сам реліз VMware Validated Design for Software-Defined Data Center 4.2 відбувся 13 лютого 2018 року. Був опублікований оновлений постер по цьому рішенню.

VMware Validated Design – це сімейство рішень для проектування центрів обробки даних, які охоплюють обчислення, зберігання, мережна взаємодія й керування, які є основою для реалізації Software-Defined Data Center (SDDC).

VMware Validated Designs (VVDs) являють собою набір документів, що пропонують, які пояснюють, як планувати, розгортати й експлуатувати SDDC. VVD засновані на наборі окремих продуктів VMware (vSphere, vSAN, NSX, vRealize Suite), які об'єднані в загальний пакет. Набір окремих продуктів і номерів їхніх версій становлять VMware Validated Design Software Bill of Materials (BOM).

VMware надає три версії VMware Validated Design для Software-Defined Data Center:

– Стандартна реалізація VMware Validated Design для Software-Defined Data Center забезпечує архітектуру єдиного центра або центра даних для регіональних центрів обробки даних. Ви можете використовувати цю реалізацію як базу для додаткових реалізацій, таких як VMware Validated Design для віддаленого офісу й філії (ROBO).

– Реалізація ROBO розширює стандартну реалізацію, включаючи віддалені філії, що вимагають меншої кількості ресурсів. Ця реалізація підходить для розгортань, які поширюються географічно, але ви хочете управляти більш централізовано. Перед роботою із впровадженням ROBO у вас повинен бути встановлений первинний центр даних Центра обробки даних із програмним забезпеченням VMware.

– Реалізація консолідованої SDDC є альтернативою однокористувальницької конфігурації VMware Validated Design for Software-Defined Data Center, у якій ви використовуєте один консолідований модуль для керування робочими навантаженнями для керування й орендаря. Ця реалізація підходить для конфігурацій з меншими витратами на запуск і апаратне забезпечення.

Сам постер має шість основних розділів:

– Архітектура логічних компонентів. У цьому розділі представлена логічна архітектура й способи розгортання й інтеграції рішень у повнофункціональний двокомпонентний центр даних із програмним забезпеченням. Починаючи із серверів vCenter і контролерів платформ із балансуванням навантаження для cross-region і Cross-vCenter NSX, завдяки хмарним операціям і автоматизації за допомогою vRealize Suite цей розділ торкати всіх цих варіантів.

– Архітектура ядра й домену. Цей розділ ілюструє концентрацію областей робочого навантаження, використовуваних у проекті, і те, як вони створюються й розгортаються для забезпечення загального набору масштабованих будівельних блоків для вашого центра даних, обумовленого програмним забезпеченням.

– Віртуальні мережі з розподіленою логічною мережею й застосунками. У цьому розділі описується використання архітектури розподіленої логічної маршрутизації в dual-region Software-Defined Data Center, шляхом включення VMware NSX у стек керування. У ньому також описується розгортання повного пакета рішень SDDC за допомогою застосунків Virtual Networks і мережних сервісів, надаваних NSX.

– Storage – у цьому розділі показане використання в дизайні vSAN і NFS . VMware Validated Design розроблений і протован за допомогою VMware vSAN в області керування й загального доступу для первинного сховища й NFS для вторинного сховища (шаблони, архіви й резервні копії).

– Домени робочого навантаження. Цей розділ демонструє як мережа працює усередині доменів робочого навантаження.

– Region Protection and Disaster Recovery. У цьому розділі показано, як дизайн захищає vRealize Automation, vRealize Orchestrator, vRealize Operations і vRealize Business for Cloud за допомогою Site Recovery Manager разом з NSX.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Як мова програмування обрана Python. Python – високорівнева мова програмування загального призначення з акцентом на продуктивність розроблювача й читаність коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, у тому числі структурне, об'єктно-орієнтоване, функціональне, імперативне й аспектно-орієнтоване. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточні обчислень і зручні високорівневі структури даних. Код у Python організовується у функції й класи, які можуть поєднуватися в модулі (які у свою чергу можуть бути об'єднані в пакети).

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється вільно під дуже ліберальною ліцензією, що дозволяє використовувати його без обмежень у будь-яких застосунках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM і інших. Проект PyPy пропонує реалізацію Python на самому Python, що зменшує витрати на зміни мови й постановку експериментів над новими можливостями.

Python – мова програмування, що активно розвивається, нові версії (з додаванням/зміною язикових властивостей) виходять приблизно раз у два з половиною року. Внаслідок цього й деяких інших причин на Python відсутні ANSI, ISO або інші офіційні стандарти, їхня роль виконує CPython.

Python портований і працює майже на всіх відомих платформах – від КПК до мейнфреймів. Існують порти під Microsoft Windows, практично всі варіанти UNIX (включаючи FreeBSD і Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, AS/400 і навіть OS/390, Symbian і Android.

При цьому, на відміну від багатьох портуємих систем, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM/DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java – Jython, що дозволяє інтерпретаторові виконуватися на будь-якій системі, що підтримує Java, при цьому класи Java можуть безпосередньо використовуватися з Python й навіть бути написаними на Python. Також кілька проектів забезпечують інтеграцію із платформою Microsoft .NET, основні з яких – IronPython і Python.Net.

Python підтримує динамічну типізацію, тобто тип змінної визначається тільки під час виконання. Тому замість «присвоювання значення змінної» краще говорити про «зв'язування значення з деяким ім'ям». У Python є убудовані типи: бульові, рядки, Unicode-рядки, цілі числа довільної точності, числа із плаваючої коми, комплексні числа й деякі інші. З колекцій Python підтримує кортежі (*tuples*), списки, словники (асоціативні масиви) і, починаючи з версії 2.4, безлічі. Всі значення в Python є об'єктами, у тому числі функції, методи, модулі, класи.

Додати новий тип можна або написавши клас (class), або визначивши новий тип у модулі розширення (наприклад, написаному мовою C). Система класів підтримує спадкування (одиночне й множинне) і метапрограмування. Можливе спадкування від більшості убудованих типів і типів розширень.

Всі об'єкти діляться на посилальні й атомарні. До атомарного ставляться int, long, complex і деякі інші. При присвоюванні атомарних об'єктів копіюється їхнє значення, у той час як для посилальних копіюється тільки покажчик на об'єкт, таким чином, обидві змінні після присвоювання використовують те саме значення. Посилальні об'єкти бувають змінювані й незмінні. Наприклад, рядки й кортежі є незмінними, а списки, словники й багато інших об'єктів – змінюваними. Кортеж у Python є, по суті, незмінним списком. У багатьох випадках кортежі працюють швидше списків, тому якщо ви не плануєте змінювати послідовність, то краще використовувати саме їх.

Мова має чіткий і послідовний синтаксис, продуману модульність й масштабованість, завдяки чому вихідний код написаних на Python програм легко читаємий.

Python – стабільна й розповсюджена мова. Він використовується в багатьох проектах і в різних якостях: як основна мова програмування або для створення розширень і інтеграції застосунків. На Python реалізоване велика кількість проектів, також він активно використовується для створення прототипів майбутніх програм. Python використовується в багатьох великих компаніях.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи інтелектуального Software-Defined Power для дата-центрів.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) 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

Якщо переводити Software Defined на українську мову, то ми одержимо термін «програмно-визначаємий» або «програмно-конфігуруємий». Однак, тренд зсуву фокуса з «заліза» на «софт» краще описує перший варіант, тому як основний ми будемо використовувати його.

Отже, Software Defined Data Center або SDDC – це програмно-визначаємий центр обробки даних. Уперше це поняття було запропоновано Стивеном Херродом в 2012 році, що займав в той час посаду технічного директора VMware.

Настільки пізніше поява концепції SDDC пов'язане із процесами, що відбувалися в IT-індустрії. В 1950х роках вартість апаратного забезпечення була непорівнянна більше вартості програм, які вважалися лише доповненням до «заліза». Тобто першим завданням було розробити апаратне рішення, а вторинної – софт.

Поступово процес комодитизації або перехід до створення більше універсальних і взаємозамінних обчислювальних машин привів до розуміння того, що програми можна буде використовувати на будь-якому комп'ютерному устаткуванні. У результаті до 80 років розробка програмного забезпечення стає самостійною галуззю, а вартість софта спочатку наближається до вартості розробки «заліза», а потім у деяких випадках починає перевищувати. Це скасовує першість hardware на користь software. Весь спеціалізований функціонал переноситься в програмне забезпечення, а апаратна частина стає більше універсальною. Процес, поширюючись на всі елементи системи центрів обробки даних, створює передумови до створення програмно-визначаємих дата-центрів або SDDC.

Першими подіями, що відкрили дорогу до SDDC, стали концепції програмно-визначаємих мереж (Software Defined Network, SDN), а навіщо трохи пізніше програмно-визначаємих систем зберігання даних (Software Defined Storage, SDS). У сукупності з технологією віртуалізації серверів, що дозволяла емулювати на базі фізичних серверів віртуальні машини з необхідними характеристиками, віртуалізація мереж і СЗД дала можливість створити стек технологій, що тепер називається SDDC.

Такий спосіб організації інфраструктури підстьобнув розвиток хмарних технологій, яким були потрібні гнучкість і легка масштабованість. Хмара, як модель надання різних IT-послуг і ресурсів (XaaS) з віддаленим доступом через Інтернет, мало потребу в технології автоматизації дій по розгортанню, переконфігуруванню й моніторингу обчислювальних ресурсів без прив'язки до статичного апаратного забезпечення. Таку можливість дало програмне забезпечення, що входить у стек технологій для програмно-визначаємих ЦОДів.

Корпоративний хмарний провайдер Cloud4Y використовує для створення програмно-визначаємої інфраструктури своєї хмари (Software Defined Infrastucture, SDI) стек пропрієтарного програмного забезпечення від VMware. Незважаючи на високу вартість ПЗ, воно дає необхідну для Enterprise-клієнтів надійність і стабільність роботи.

В основі платформа VMware vSphere, що, на думку багатьох фахівців, є галузевим стандартом для корпоративних інфраструктур. vSphere містить у собі сервер керування ESXi-хостами й СЗД, так званий, vCenter Server. ESXi-хости – це фізичні сервери з гіпервізорами VMware ESXi, які відповідають за емуляцію віртуальних машин, кожний на своєму сервері. vCenter Server «підминає» під себе ESXi-хости і є «командним пунктом», що дозволяє не тільки управляти всіма можливостями кожного хоста із групи, але й створювати кластери з опціями балансування (DRS), «живої» міграції (vMotion) і швидкого відновлення віртуальних машин на резервних хостах (HA) для підвищення відказостійкості.

В особливій кластерній файловій системі зберігаються розділи з файлами віртуальних машин, які доступні для читання й запису всім ESXi-хостам кластера. Через зберігання в одному місці й незалежності віртуальної машини від фізичної платформи досягається швидке переміщення й відновлення за допомогою HA, DRS, FT, vMotion.

Для мінімальної реалізації SDDC залишається згадати про рішенні VMware для програмно-конфігуруємих мереж (Software Defined Network або SDN). VMware NSX відтворює модель мережі на програмному рівні. Мережа NSX – це бібліотека логічних мережних елементів і служб, таких як логічні комутатори, маршрутизатори, брандмауери, засоби балансування навантаження, VPN та інші. Завдяки цьому рішенню, що є частиною платформи vSphere, адміністратори провайдеру й орендарів хмари можуть у короткі строки поєднувати віртуальні машини в мережі з будь-якою топологією або створювати гібридні хмари.

Клієнти не використовують VMware vCenter Server. За керування кластерами й фізичним устаткуванням відповідає провайдер. Клієнти одержують значну кількість можливостей керування своїм віртуальним ЦОДом за допомогою зручного порталу самообслуговування VMware vCloud Director. Створення SDDC або vЦОДа для клієнта відбувається в найкоротший термін з однієї панелі vCloud Director, при цьому може бути створена необхідна кількість віртуальних машин з потрібними характеристиками й операційними системами, а потім вони можуть бути об'єднані у внутрішні або гібридні хмари. vCloud Director і vCenter це частина стека SDDC, які відповідають за автоматизацію й керування, багато в чому саме завдяки цьому, подібне рішення набирає популярність, рятуючи IT-Фахівців від одноманітної роботи з адміністрування парку віртуальних машин.

Також багато хто вважають за необхідне виділити окремим елементом стека засобу забезпечення інформаційної безпеки, які повинні бути віртуалізовані й надаватися у вигляді сервісів.

SDDC – це інтегрований рівень абстракції, що визначає весь ЦОД засобами програмного забезпечення, що складає з інфраструктурних компонентів – процесори, системи зберігання, мережне устаткування й засоби забезпечення інформаційної безпеки, автоматизації й керування, з яких формуються загальні пули віртуальних ресурсів, надавані споживачам у вигляді сервісів (XaaS).

Використання стека технологій VMware для реалізації SDI хмари Cloud4Y дозволяє підвищити якість обслуговування (Quality of Service, QoS) і знизити вартість у порівнянні з реалізацією необхідних варіантів інфраструктури замовника з використанням традиційних ЦОДів шляхом оренди фізичних серверів (bare metal) або покупки власних. Економічність досягається за рахунок відмови клієнтів від капітальних витрат на користь операційних витрат і оплати тільки за фактично використані ресурси з можливістю погодинної тарифікації. Незважаючи на комодитизацію устаткування для забезпечення рівня доступності від 99,982% на місяць, закріпленого в SLA, Cloud4Y використовує надійне серверне й мережне устаткування, що розташовано в мережі дата-центрів TIER III, з'єднаних оптичним кільцем з дублюванням каналів зв'язку. Це й інші організаційно-технічні міри дозволяють гарантувати клієнтам хмари високий рівень безпеки і якості послуг.

3.2 Розробка структурної схеми

Програмно-визначаємий центр обробки даних (ЦОД), елементом якого є Software-Defined Power, або програмно-визначаєме електроживлення, складається з повністю віртуалізованої інфраструктури, що швидко адаптується під мінливі потреби бізнесу за рахунок керування на програмному рівні й автоматизації більшості процесів.

Повна віртуалізація серверів, мереж і систем зберігання даних дозволяє консолідувати існуючі ресурси й оптимально розподіляти робочі навантаження,

що підвищує гнучкість і масштабованість IT-інфраструктури, а також вивільняє устаткування.

Переваги програмно-визначаємого ЦОД:

– підвищення доступності й збільшення коефіцієнтів використання існуючих ресурсів;

– зниження капітальних витрат – устаткування потрібно не модернізувати, а віртуалізувати;

– для віртуалізації й адміністрування використовується ПЗ, що швидше й гнучкіше, ніж устаткування;

– оперативна реконфігурація й перепрофілювання інфраструктури відповідно до бізнес-вимогами;

– миттєва масштабованість і прискорення розробки застосунків і виділення технологічних ресурсів;

– воля в плані проектування інфраструктури – різні варіанти вибору устаткування;

– забезпечення більше високого захисту віртуальних ресурсів, чим самі надійні фізичні середовища.

Програмно-визначаємий ЦОД поєднує рішення по віртуалізації серверів, сховища даних і мережних компонентів, а також включає інструменти для створення цифрової робочої області

Що ж дає використання DCIM? Дані, надавані DCIM, елементом якого є Software-Defined Power, дозволяють оптимізувати живлення, охолодження й фізичний простір у машинному залі, що, у свою чергу, дає можливість відстрочити капітальні вкладення в облаштованість нових площ. Наявний у багатьох систем інструментарій дозволяє моделювати сценарії «що, якщо», тобто зрозуміти, як різні дії, наприклад розміщення додаткових стійок або устаткування, вплинуть на ситуацію в ЦОД. Відповідно до розрахунків Gartner, за рахунок однієї тільки економії енергії впровадження DCIM, елементом якого є Software-Defined Power, окупається за три роки.

Разом з тим жодна із представлених на ринку систем поки не має ту функціональність, який повинна володіти DCIM, елементом якого є Software-Defined Power, та й саме питання про її обов'язкові компоненти залишається відкритим. До того ж повномасштабна реалізація цієї системи потрібно далеко не завжди. Просуваються під парасолькою DCIM, елементом якого є Software-Defined Power, продукти ведуть свій родовід від самих різних рішень. Відповідно, вони можуть бути сильні в одних областях (облік і інвентаризація, моніторинг енергопостачання, контроль СКС) і слабкіше в інші. Все це, поряд з відсутністю безпосереднього бенефіціара, утрудняє впровадження DCIM, елементом якого є Software-Defined Power, і ускладнює вибір.

Моніторинг мікроклімату ЦОД

Щоб домогтися значної економії енергії, зовсім не обов'язково впроваджувати повномасштабну дорогу систему DCIM, елементом якого є Software-Defined Power, для цієї мети підійде й система контролю мікроклімату ЦОД, наприклад Synapsense. Як свідчить досвід реалізації проектів цієї американської компанії, впроваджене рішення може окупитися за дуже короткий час, іноді буквально за місяць.

Synapsense призначена для постійного контролю кліматичних параметрів ЦОД у режимі реального часу. Її основу становлять бездротові датчики температури, вологості й тиску, які підключаються до шлюзів за технологією 802.15. Хоча система працює в тім же частотному діапазоні 2,4 Ггц, що й традиційний Wi-Fi, завдяки частотно-тимчасовому поділу каналів вона не буде створювати перешкод для розгорнутої в ЦОД мережі WLAN.

У невеликих ЦОД (від 100 до 1000 м^2) рекомендується використовувати два шлюзи для резервування. Оскільки мережа має ніздрювату топологію, у випадку виходу з ладу обох шлюзів датчики будуть по ланцюжку передавати інформацію один іншому, і в остаточному підсумку вона потрапить у програмне забезпечення, що є ядром даної системи. Якщо датчик «бачить», що канал зайнятий Wi-Fi, він регулярно посилає запити, очікуючи вільного вікна, і у

випадку його появи передає інформацію на базовий шлюз. При відсутності такої можливості дані переміщаються на сусідній датчик, що акумулює їх і передасть у той момент, коли радіоканал звільниться. Ця технологія робить систему відказостійкою і перешкодозахищеною.

Для кожної стійки рекомендується придбати комплект із семи датчиків, які встановлюються на фронтальну й тильну частини стійки, угорі, у середині й унизу, а опорний датчик температури розташовують під фальшполом. На кондиціонери датчики температури ставляться на вході й виході повітряного потоку. Кожне із цих пристроїв вимірює вологість. Систему можна розширити за рахунок датчиків контролю електроживлення (датчики Холу). На підставі отриманих даних Synapsense розраховує PUE і безліч інших показників, які допоможуть оптимізувати електроспоживання ЦОД, а також виявляти ще що тільки зароджуються проблеми – наприклад, коли при перемиканні кондиціонерів на резервні потужності термодинамічні процеси переходять із квазістаціонарних у перехідні.

Температуру й вологість, та й інші параметри, виміряти неважко. Основу Synapsense становить програмне забезпечення, що цю інформацію обробляє, відображаючи її в реальному часі на картах температури, тиску й вологості. Відомості, що зберігаються в базі даних, у будь-який момент можна відтворити для відновлення подій, які відбувалися й місяць, і рік назад. Так, при виникненні якихось проблем, можна подивитися, звідки почався перегрів, – якщо «винуватий» кондиціонер, на плані відразу будуть видні червоні області й перепади тиску в цих місцях.

У порівнянні з аналогами, що використовуються в обчислювальних центрах, система моніторингу Synapsense легко встановлюється у вже існуючому або знову споруджуваному ЦОД, оскільки її бездротові базові станції не вимагають зовнішніх ліній зв'язку або зовнішнього живлення (вони здатні працювати від батарейок до шести років при частоті опитування раз у п'ять мінут). Гнучкість налаштування інтерфейсу керуючого ПЗ дозволить надати

кожному користувачеві тільки ті дані, які відповідають його рівню доступу. Візуалізація даних, що збираються системою Synapsense, дозволяє оперативно відслідковувати зміни температури, вологості або тиску на побудовані для декількох рівнів картах з колірною індикацією, а сигнал про перевищення встановлених порогів значення миттєво надійде на пульт чергової зміни.

Як затверджується, система може працювати не тільки із програмними, але й з апаратними протоколами (SNMP, Modbus, BUCnet). Це дозволяє розширити її практично до повнофунціональної системи DCIM, елементом якого є Software-Defined Power.

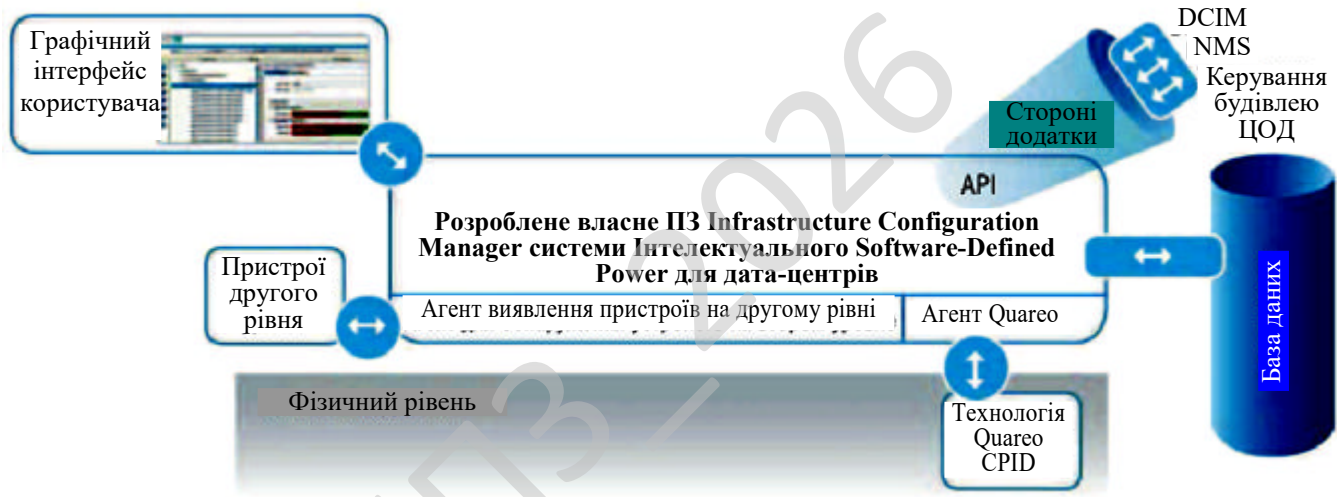

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

У першу чергу, мова йде про те, що вдало обране й настроєне DCIM-рішення дозволяє підвищити ефективність експлуатації як усього дата-центра, так і окремих його елементів. Внаслідок цього падає вартість IT-сервісу при перерахуванні на один користувача в одиницю часу. У перспективі, це дозволяє поліпшити надійність роботи ЦОД з одночасним поліпшенням сервісів,

надаваних користувачам. Ще одна приємна сторона – зниження операційних витрат на експлуатацію ЦОД.

Що стосується ключових можливостей DCIM, елементом якого є Software-Defined Power, які потрібні в першу чергу, те тут у даній роботі визначені наступні:

- Моніторинг енергоспоживання.
- Керування джерелами енергії.
- Моніторинг стану мікроклімату ЦОД.
- Звітність.
- Візуалізація даних.
- Керування ресурсами.
- Предиктивний аналіз.
- Моделювання й симуляція.
- Моніторинг вентиляції.
- Керування робочими процесами.
- Оцінка фізичного стану IT-устаткування.
- Оптимізація охолодження.
- Автоматизація IT-процесів.
- Планування навантаження.
- Керування робочим процесом/змінами.

На рисунку 3.2 відображена функціональна схема системи. Зафарбованими блоками відображені елементи Software-Defined Power (програмно-визначаєме електроживлення).

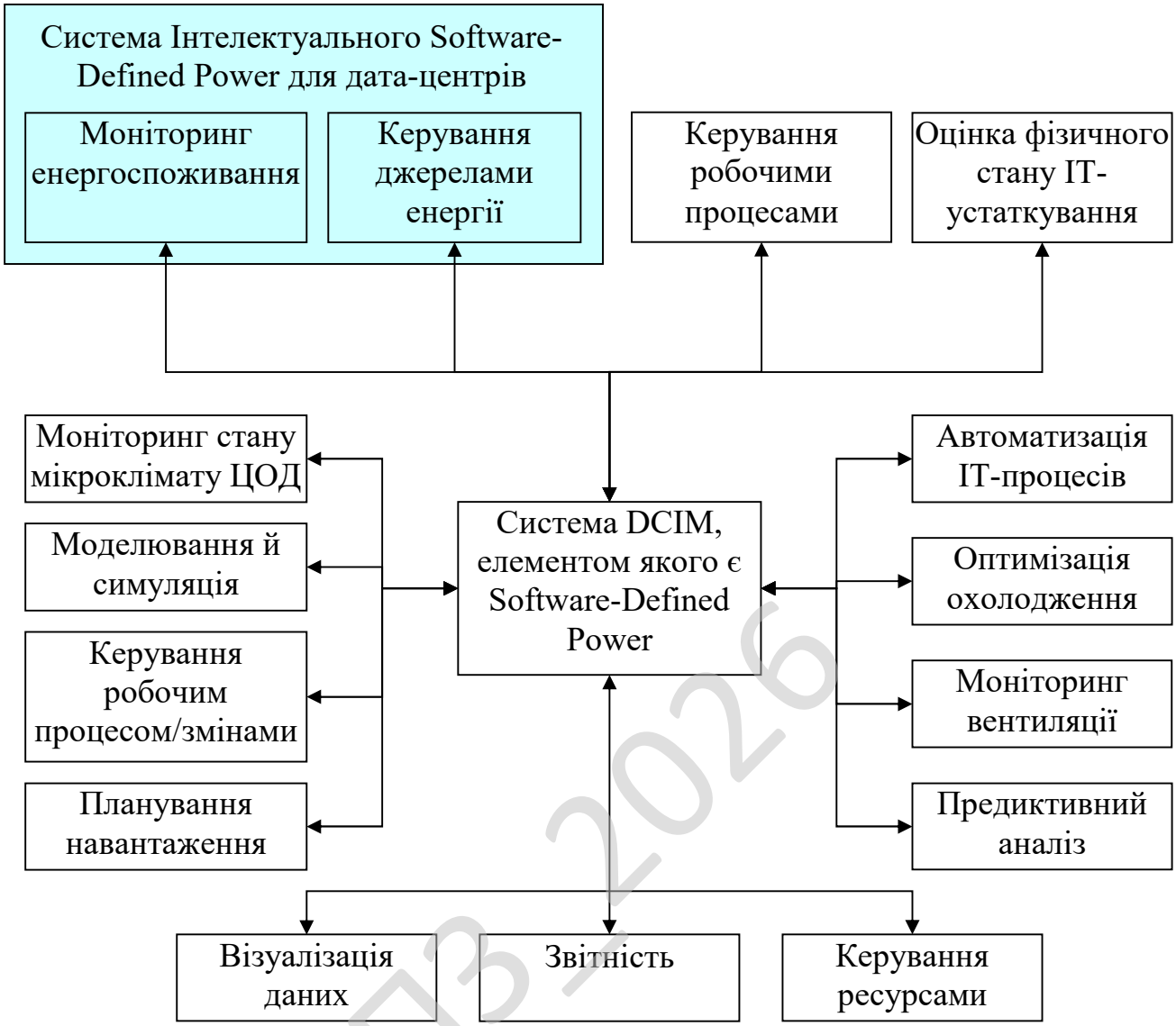

Рисунок 3.2 – Функціональна схема системи

Ми вважаємо, що наріжним каменем будь-якого DCIM-рішення повинне бути об'єднання потоків даних з максимального числа джерел. Цю інформацію можна збирати й обробляти режимі реального часу, що дає операторам ЦОД практично повну інформацію із всіх параметрів дата-центра, включаючи обчислювальну потужність у будь-який момент часу, надійність інфраструктури, керування активами, термін служби устаткування й багато чого іншого.

DCIM-рішень зараз досить багато, а загальний обсяг ринку цих рішень експерти оцінюють приблизно в $1 млрд. Звичайно в компанії є два сценарії впровадження рішень такого роду. Перший сценарій – це створення дата-центра з

інтеграцією готового рішення на старті проектування. Другий – об'єднання різних автономних систем різних виробників у єдине ціле.

При роботі з будь-яким сценарієм обране рішення повинне працювати з усією структурою дата-центра, включаючи електричні, механічні, електронні елементи, системи зберігання даних і все інше. Це потрібно для того, щоб оператори ЦОД безперешкодно цілодобово одержували всю доступну інформацію, що може виявитися корисною. При цьому DCIM-система повинна вміти працювати як зі старими, так і з новими пристроями, протоколами й стандартами.

Щоб одержати більше інформації, така система повинна бути оснащена як основними, так і допоміжними датчиками. Найчастіше використовуються системи, здатні:

– Зчитувати інформацію про рівень споживання електроенергії в енергетичній інфраструктурі;

– Зчитувати інформацію про температуру й вологість на стійках, у системах охолодження й усередині воздуховодів;

– Радіомітки на всьому устаткуванні або найважливіших елементах, що необхідно для спрощення їхньої інвентаризації;

– Сенсори для виявлення витоків рідини.

Відразу варто сказати, що велика кількість інформації, що збирається подібними системами – не самоціль. Головне, щоб всі ці дані були корисними для оператора ЦОД.

Вибираємо DCIM, елементом якого є Software-Defined Power

Якщо враховувати все, що вже сказано вище, при виборі DCIM-рішення для свого ДЦ варто зробити от що:

– Розрахувати бюджет проекту з обліком всіх можливих схованих витрат.

– Провести кілька консультацій зі своїми ж фахівцями – брати участь повинні всі зацікавлені сторони.

– Скласти список усього того, що потрібно враховувати й за чим варто спостерігати. Список повинен бути пріоритетним – спочатку найважливіші елементи, потім – менш важливі.

– Переглянути, які протоколи й засоби комунікації підтримуються інфраструктурним устаткуванням.

– Додати додаткові датчики в тому випадку, якщо це потрібно.

– Вибір підходящих DCIM-рішень, які можуть виконувати всі поставлені завдання.

– Зконфігурувати сервер під DCIM-рішення, що було вирішено вибрати в остаточному підсумку.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників.

Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

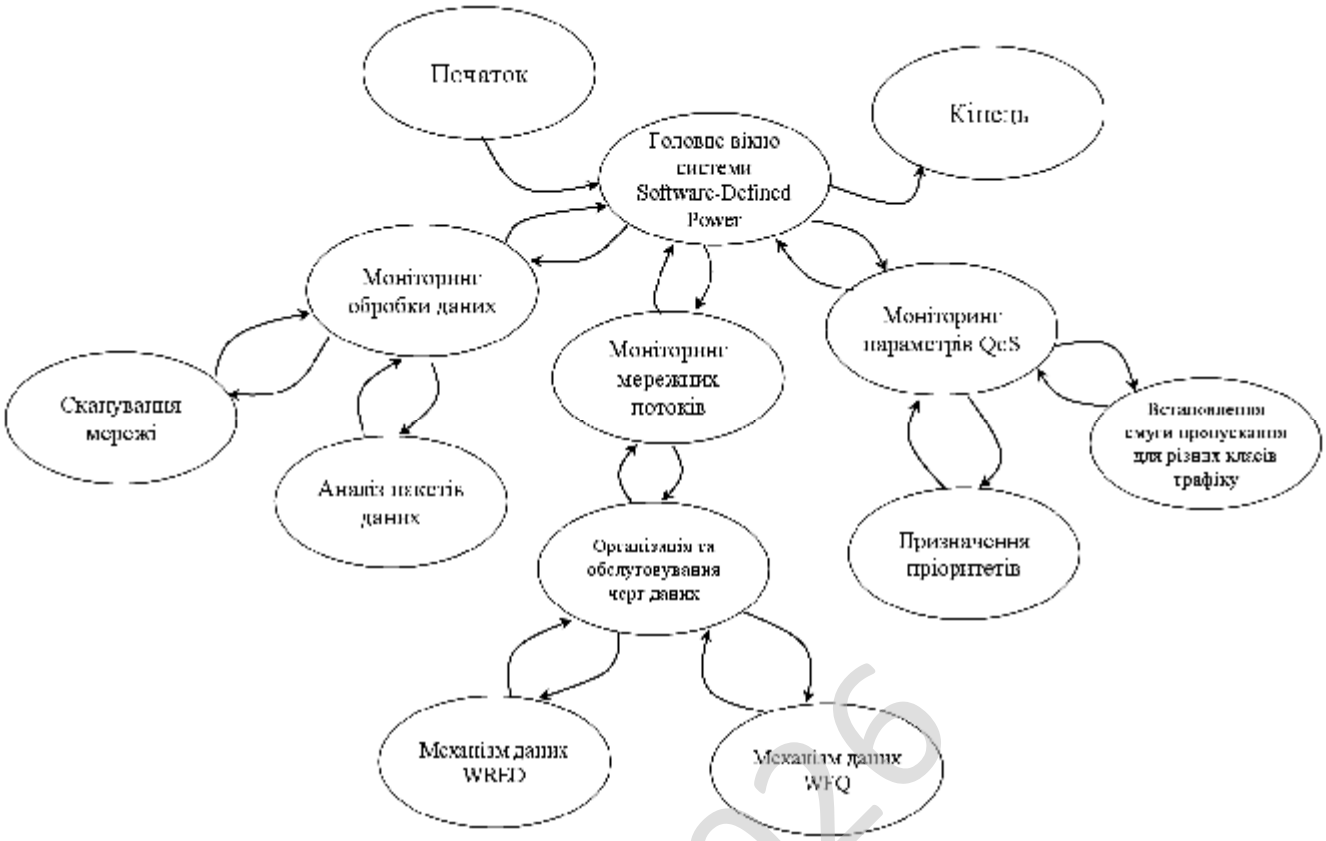

Рисунок 3.3 – Діаграма взаємодії процесів

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.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграми потоків даних містять чотири типи елементів:

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

– Сховища даних (репозиторії).

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо реалізацію бакалаврської дипломної роботи. Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень.

Блок-схеми показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. Тому від точності і детальної блок-схеми залежить результат всієї програми.

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 високого рівня декомпозиції задач на класи.

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 підсистеми 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.

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.

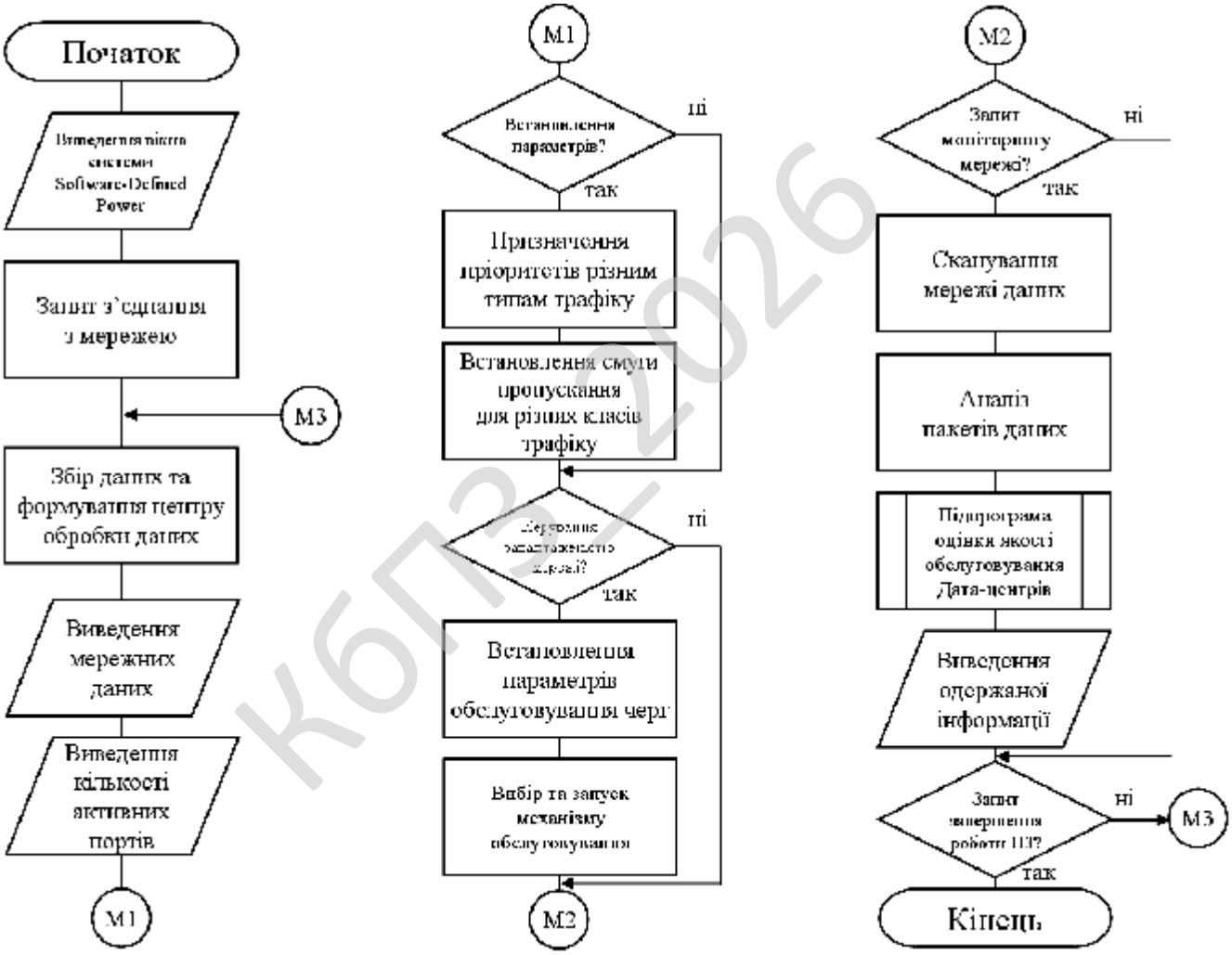

Рисунок 4.1 – Блок-схема основної програми

Було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу).

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.

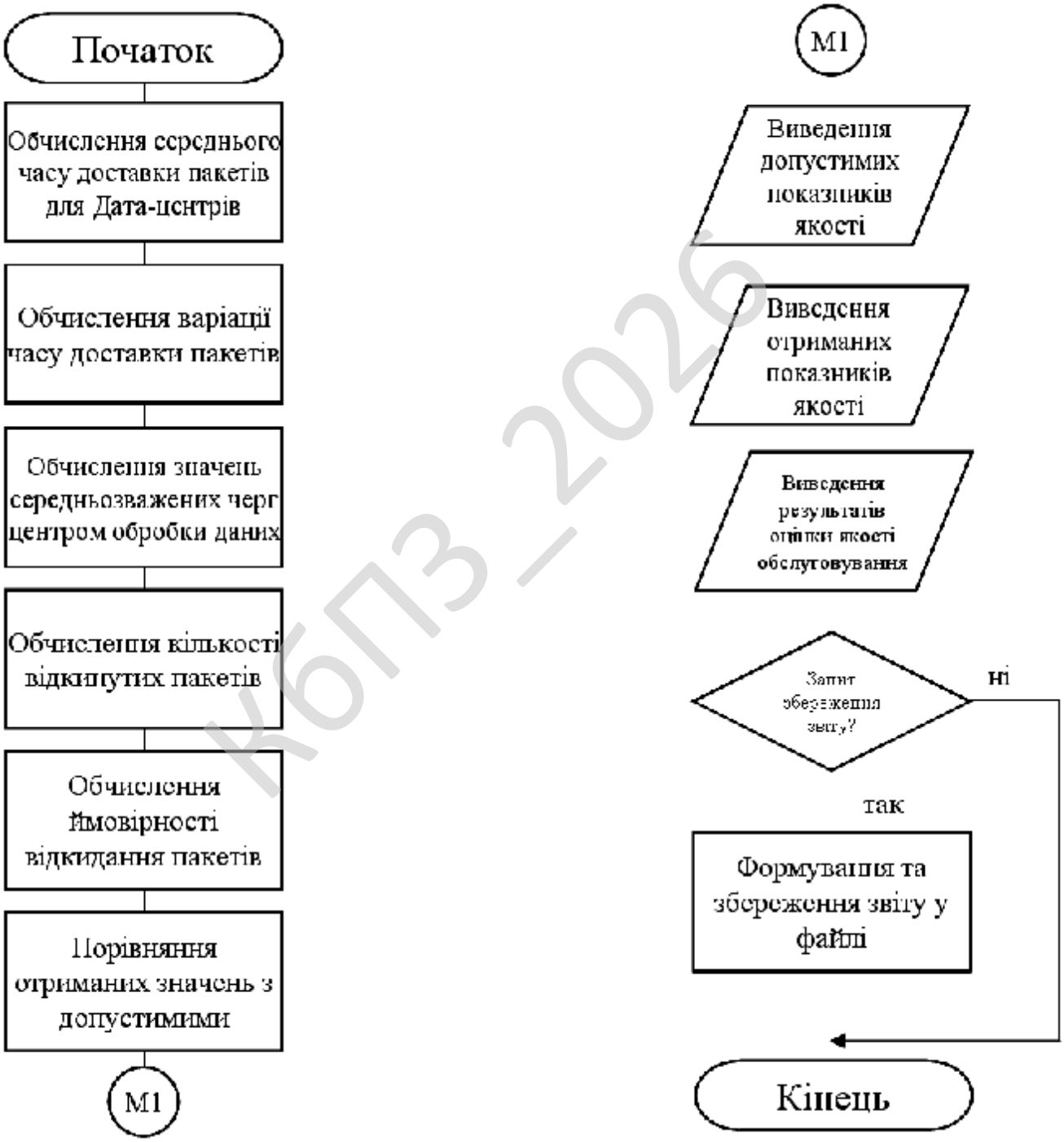

Рисунок 4.2 – Блок-схема роботи підпрограми

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.

Розглянемо обрану клієнт-серверну архітектуру. Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного

сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

– прикладний рівень, який реалізує основну логіку ПЗ і на якому здійснюється необхідна обробка інформації;

– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка ПЗ та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується

прикладна логіка ПЗ. Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення –звичайний браузер. Це позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає).

Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-

засоби можуть бути найрізноманітнішими. Так, для створення серверних програм, крім PHP, широко застосовуються Java, Perl, Python, Delphi.

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

Розглянемо обраний формат даних XML.

Розши́рювана мо́ва розмі́тки (Extensible Markup Language, скорочено XML) – запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. Є спрощеною підмножиною мови розмітки SGML.

XML-документ складається із текстових знаків, і придатний до читання людиною. Стандарт XML (Recommendation, перше видання від 10 лютого 1998, останнє, четверте видання 29 вересня 2006) визначає набір базових лексичних та синтаксичних правил для побудови мови описання інформації шляхом застосування простих тегів.

Цей формат достатньо гнучкий для того, аби бути придатним для застосування в різних галузях. Іншими словами, запропонований стандарт визначає метамову, на основі якої шляхом запровадження обмежень на структуру та зміст документів визначаються специфічні, предметно-орієнтовані мови розмітки даних. Ці обмеження описуються мовами схем (Schema), такими як XML Schema (XSD), DTD або RELAX NG. Прикладами мов, заснованих на XML, є: XSLT, XAML, XUL, RSS, MathML, GraphML, XHTML, SVG, а також XML Schema.

Основні поняття. Коректність.

Коректний документ (well-formed document) відповідає всім синтаксичним правилам XML. Документ, що не є коректним, не може називатись XML-

документом. Сумісний синтаксичний аналізатор (Conforming parser) не повинен обробляти такі документи. Зокрема, коректний XML-документ має:

1. Лише один елемент у корені.

2. Непорожні елементи розмічено початковим та кінцевим тегами (наприклад, <пункт> Пункт 1</пункт>). Порожні елементи можуть позначатися «закритим» тегом, наприклад <IAmEmpty />. Така пара еквівалентна <IAmEmpty></IAmEmpty>.

3. Один елемент не може мати декілька атрибутів з однаковою назвою. Значення атрибутів перебувають або в одинарних ('), або у подвійних (") лапках.

4. Теги можуть бути вкладені, але не можуть перекриватись. Кожен некореневий елемент мусить повністю перебувати в іншому елементі.

5. Документ має складатися тільки з правильно закодованих дозволених символів Юнікоду. Єдиними кодуваннями, які обов'язково має розуміти XML-процесор, є UTF-16 та UTF-8. Фактичне та задекларованe кодування (character encoding) документа мають збігатись. Кодування може бути задекларовано ззовні, як у заголовку «Content-Type» при передачі по протоколу HTTP, або в самому документі використанням явної розмітки на самому початку документа. У разі відсутності інформації про кодування, документ має бути в кодуванні UTF-8 (або його підмножині ASCII).

Валідність

Документ називається валідним (valid), якщо він є коректним, містить посилання на граматичні правила та повністю відповідає обмеженням, вказаним у цих правилах (DTD або XML Schema або іншому подібному документі).

Синтаксичний аналізатор

Синтаксичним аналізатором (часто парсер, від parser) називається програма або компонент, що читає XML-документ, проводить синтаксичний аналіз та відтворює його структуру. Якщо синтаксичний аналізатор перевіряє документ на валідність, то такий аналізатор називають валідатором (validating).

Назви елементів чутливі до регістру літер. Наприклад, наступна пара елементів правильна: <Step> … </Step> у той час як ця – ні: <Step> … </step>.

Правильний вибір назв для XML-елементів підкреслюватиме значення даних у створеній мові розмітки. Це сприятиме полегшенню роботи людей з такими документами, зберігаючи можливості для комп'ютерної обробки даних.

Вибір змістовних назв передає семантику елементів та атрибутів для людини, без посилання на зовнішню документацію. Однак це може призвести до надмірності розмітки, що ускладнює редагування й збільшує розмір файлів.

Правильний вибір назв для XML-елементів підкреслюватиме значення даних у створеній мові розмітки. Це сприятиме полегшенню роботи людей з такими документами, зберігаючи можливості для комп'ютерної обробки даних. Вибір змістовних назв передає семантику елементів та атрибутів для людини, без посилання на зовнішню документацію. Однак це може призвести до надмірності розмітки, що ускладнює редагування й збільшує розмір файлів. XML-документи мають як фізичну, так і логічну структуру.

Приклад XML-документа:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.3/" xml:lang="uk">
  <page>
    <title>Фукідід</title>
    <id>1529</id>
    <revision>
      <id>4382</id>
      <timestamp>2006-09-18T22:11:53Z</timestamp>
      <minor />
      <comment>Interwiki</comment>
      <text xml:space="preserve">{{Wikipedia}}
      </text>
    </revision>
  </page>
</mediawiki>
```

Фізична структура

Сутності (Entity). Головною сутністю є зміст документа. Інші можливі сутності вказуються за допомогою. Посилання на сутності (&назва; в самому

документі, та, наприклад %назва; у визначені його типу) можуть слугувати в ролі позначень спеціальних символів, посилань на спеціальні символи або окремих документів чи фрагментів тексту.

XML-декларація, в ній вказується версія XML, кодування та інша допоміжна інформація.

Декларація типу документа може застосовуватись для того, аби додавати нові типи сутностей та визначати логічну структуру документа.

Логічна структура

XML-документ має ієрархічну логічну структуру, і може представлятись у вигляді дерева. Вузлами цього дерева можуть бути: елементи, фізична структура яких складається із коректної пари відкриваючого та закриваючого тегів (<Назва-тега>) та (</Назва-тега>), або тега порожнього елемента (<Назва-тега />).

Атрибути, що мають вигляд пар ключ-значення (назва атрибута="значення атрибута") і знаходяться або у відкриваючому, або у порожньому тезі (подібно до метаданих).

Вказівки щодо обробки документа (Processing Instruction) (<?Обробник параметр ?>).

Коментарі (<!-- Текст коментаря -->).

Текст, або у вигляді простого тексту, або фрагментів CDATA (<![CDATA[довільний текст]]>).

XML-документ повинен мати лише один кореневий елемент. Решта елементів є піделементами цього кореневого елемента. Деякі веб-браузери здатні безпосередньо відображати XML-документи. Це може досягатись шляхом застосування таблиці стилів (Stylesheet). Вказані у таблиці стилів операції можуть призводити до перетворення XML-документа в інший, відмінний від XML формат.

Коректність XML-документів

Залишивши назви, дозволену ієрархію, та значення елементів і атрибутів відкритою та можливою бути визначеною в спеціалізованих схемах або визначеннях типу документа (DTD).

XML утворює синтаксичну основу для створення спеціалізованих, заснованих на XML мовах розмітки даних.

Загальний синтаксис таких документів стабільний і наперед визначений – документи мають відповідати базовим вимогам XML, гарантуючи те, що довільне програмне забезпечення з підтримкою XML буде здатне щонайменше зчитати і відтворити відносну структуру інформації, що міститься в них.

Схема лише доповнює синтаксичні правила множиною обмежень. Зазвичай схеми обмежують назви елементів та атрибутів, дозволені типи значень і допустиму ієрархію елементів, наприклад, дозволяючи лише елементу з назвою «народження» містити під-елемент з назвою «місяць» та з назвою «день», і кожен із них мусить містити лише літери. Обмеження, вказані в схемі, можуть також включати присвоєння певних типів даних для впливу на те, як обробляється інформація; наприклад, дані елемента «місяць» можна визначити як такі, що містять лише місяць, як це визначено відповідно до використаної мови схем.

Коректний XML-документ, що відповідає обмеженням схеми або DTD, називається валідним.

DTD (Document Type Definition). Найдавнішим форматом схем для XML є успадкований від SGML формат визначення типу документа (Document Type Definition, DTD). У той час, як через включення до стандарту XML 1.0 DTD став поширеним форматом схем, він має такі обмеження:

1. Відсутність нових можливостей XML, із найважливішою з посеред них простори назв.

2. Брак виразності. Деякі формальні аспекти XML-документів неможливо відобразити в DTD.

3. Використовується спеціалізований, заснований не на XML синтаксис для опису схем.

DTD все ще використовується в багатьох програмах, оскільки він вважається найпростішим форматом для аналізу та збереження.

XML Schema. На заміну DTD було розроблено нову мову схем – XML Schema (буквально – XML-схема), скорочено позначається як XSD (від XML Schema Definition). XSD набагато потужніші за DTD в описанні заснованих на XML мов. Вони використовують багатий набір типів даних, підтримують детальніші обмеження на структуру документів, та повинні оброблятись складнішими системами. XSD побудовано на основі XML, що робить можливим використання звичайних інструментів XML для їхньої обробки, хоча реалізації XSD вимагають набагато більше аніж просто можливість читати XML.

Серед недоліків XSD називають такі:

1.Специфікація дуже велика, що робить її складною для розуміння та реалізації.

2. Оснований на XML синтаксис додає надмірності мові, що ускладнює читання та запис XSD.

3. Валідація відносно схеми може бути дорогим додатком до синтаксичного аналізу XML-документів.

4. Можливості моделювання дуже обмежені, без можливості впливу значень атрибутів на вміст елементів.

5. Модель отримання типів даних є дуже обмеженою, зокрема в тому, що отримання шляхом розширення є рідко коли корисним.

6. Механізми ключа/посилання на ключ/унікальності не враховують тип даних.

7. Концепція PSVI (Post Schema Validation Infoset) не має стандартного представлення або прикладного програмного інтерфейса, що працює проти незалежності від реалізації, якщо не виконується повторна валідація.

RELAX NG є іншою поширеною мовою схем для XML. Вперше RELAX NG було визначено стандартом OASIS, а тепер – міжнародним стандартом ISO (як частина DSDL). Ця мова схем має два формати: оснований на XML, та компактний, не-XML. Компактний синтаксис призначений для покращення можливості читання та написання схем, однак, оскільки існує точно визначений спосіб перетворення компактного формату в оснований на XML, і навпаки, не втрачаються переваги від використання стандартних XML-інструментів. RELAX NG має простіші системи для визначення та валідації у порівнянні з XML Schema, що робить її привабливішою для використання та реалізації. Також існує можливість використання модулів роботи з типом даних; наприклад, автор схеми RELAX NG може вказати, що значення XML-документа мають відповідати визначенням типам даних у форматі XML Schema Datatypes.

ISO DSDL та інші мови схем

Стандарт ISO DSDL (Document Schema Description Languages, мови описання схем документів) об'єднує широке коло малих мов схем, кожна із яких призначена для розв'язання окремих проблем.

До DSDL належить RELAX NG із повним та компактним синтаксисом, мова припущень Schematron, та мови для визначення типів даних, обмежень на літери, перейменування та розкривання мнемонік, та основане на просторах назв перенаправлення фрагментів документів у різні валідатори.

Мови DSDL все ще не мають підтримки як у XML Schema, і є, певною мірою, реакцією видавців на брак можливостей XML Schema для видавничої справи.

Деякі мови схем не тільки описують структуру певного формату XML-документів, а ще і мають обмежені можливості впливу на обробку документів цього формату. Як DTD, так і XSD мають цю можливість; наприклад, вони можуть визначати значення для атрибутів «за замовченням». Натомість, як RELAX NG так і Schematron таких можливостей не мають.

Обробка XML-документів

До XML-обробки відносяться форматування, синтаксичний аналіз, редагування, перевірка коректності і перетворення в інші формати.

До традиційних технологій обробки XML-документів належать такі три технології:

1. Написання програм на мові програмування із використанням API SAX.

2. Написання програм на мові програмування із використанням API DOM.

3.Застосування механізму перетворення та фільтра.

До новіших технологій, що почали здобувати поширення останнім часом належать: Активний аналіз. Зв'язування даних.

Простий API для XML (SAX)

Простий програмний інтерфейс для XML (Simple API for XML, SAX) є основаним на подіях інтерфейсом лексичного аналізу. Відповідно до цієї моделі, документ аналізується послідовно, а вміст документа передається на обробники подій аналізатора користувача. SAX є порівняно швидким та легким для реалізації проте складним з точки зору задачі отримання інформації із різних частин XML-документів, оскільки розробник аналізатора мусить дбати про відстеження поточної частини документа. Запропонований SAX підхід краще підходить до ситуацій, коли певний тип інформації завжди обробляється однаково, попри те, в якій частині документа вона знаходиться.

Об'єктна модель документа (DOM)

Об'єктна модель документа (Document Object Model, DOM) є програмним інтерфейсом, який дозволяє здійснювати обхід цілого документа так, наче він є деревом, вузлами якого є об'єкти, що відтворюють зміст документа. Документ DOM може створюватись синтаксичним аналізатором або користувачами (з деякими обмеженнями).

Типи даних вузлів DOM-дерев є абстрактними; реалізації мають власні, специфічні для мов програмування типи даних. Реалізації DOM мають тенденцію до інтенсивного використання пам'яті, оскільки зазвичай перед початком роботи

документ має бути повністю завантажений, оброблений та перетворений на дерево об'єктів.

Перетворення документів

Extensible Stylesheet Language фільтр в родині XSL може перетворювати XML-документи на інші XML-документи, для перегляду на екрані або друку.

XSL-FO є декларативною, заснованою на XML мовою для макетування сторінок. XSL-FO процесор може перетворювати XSL-FO документ в інший, не заснований на XML формат, такий як PDF.

XSLT є декларативною, основаною на XML мовою описання перетворення документів.

XSLT-процесор може використовувати XSLT-стиль в ролі інструкції для перетворення дерева даних, представленого одним XML-документом на інше дерево, що може потім бути серіалізоване в XML, HTML, простий текст, або інший, підтримуваний процесором, формат.

XQuery є розробленою консорціумом W3C мовою для написання запитів, створення та перетворення XML-даних.

XPath є подібною до DOM моделлю дерева та мовою описання шляхів для вибору даних в XML-документах. XSL-FO, XSLT та XQuery використовують XPath. XPath також містить бібліотеку додаткових функцій.

Активний аналіз

З точки зору активного аналізу (Pull parsing) XML-документ розглядається як послідовність елементів, які зчитуються послідовно, використовуючи шаблон проектування ітератор.

Такий підхід дозволяє створення рекурсивних аналізаторів, у яких структура коду відображає структуру аналізованих XML-документів, проміжні результати аналізу можуть бути використані і розміщені у вигляді локальних змінних в підпрограмах, що виконують аналіз, передані як параметри до підпрограм нижчого рівня або повернені до підпрограм вищого рівня.

До прикладів активних аналізаторів належать StAX у мові програмування Java, SimpleXML у PHP та System.Xml.XmlReader у .NET.

Активний аналізатор створює ітератор, що послідовно обходить різні елементи, атрибути та дані в XML-документі.

Код, що використовує цей «ітератор», може перевіряти поточний елемент (аби дізнатись, наприклад, чи є цей елемент стартовим, кінцевим або текстовим), та дізнаватись про його атрибути (локальна назва, простір назв, значення XML-атрибутів, зміст тексту тощо) і може пересунути ітератор на наступний елемент. Таким чином, код аналізатора може зчитувати інформацію із документа під час обходу.

Підхід рекурсивного спуску сприяє зберіганню даних у вигляді типізованих локальних змінних в коді аналізатора, в той час як SAX, наприклад, зазвичай вимагає, аби аналізатор явно зберігав проміжні дані в стеку елементів, що є вищими елементами від того, який зараз аналізується.

Код активного аналізатора може бути більш прямолінійним та зрозумілим і простішим для підтримки, аніж код SAX аналізатора.

Зв'язування даних

Іншим підходом до обробки XML-документів є зв'язування даних (Data binding). Відповідно до цього підходу, XML-дані доступні у вигляді спеціальних, строго типізованих структур даних.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму DES.

DES є класичною мережею Фейштеля із двома гілками. Дані шифруються 64-бітними блоками, використовуючи 56-бітний ключ. Алгоритм перетворить за кілька раундів 64-бітний вхід в 64-бітний вихід. Довжина ключа дорівнює 56 бітам. Процес шифрування складається із чотирьох етапів. На першому з них

виконується початкова перестановка (IP) 64-бітного вихідного тексту (забілювання), під час якої біти переставляються у відповідності зі стандартною таблицею. Наступний етап складається з 16 раундів однієї й тої ж функції, що використовує операції зрушення й підстановки. На третьому етапі ліва й права половини виходу останньої (16-й) ітерації міняються місцями. Нарешті, на четвертому етапі виконується перестановка IP^{-1} результату, отриманого на третьому етапі. Перестановка IP^{-1} інверсна початковій перестановці.

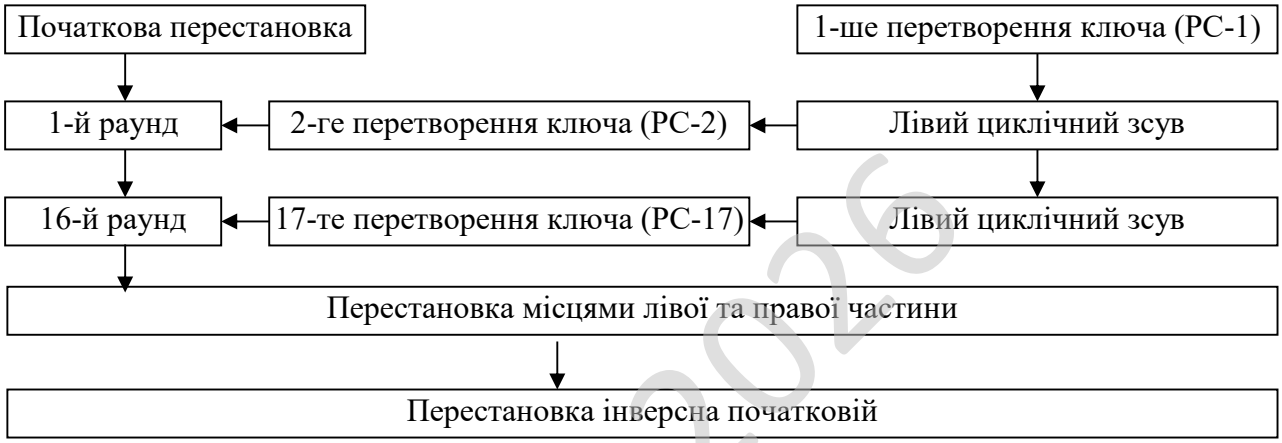

Рисунок 4.3 – Загальна схема DES

Праворуч на рисунку показаний спосіб, яким використовується 56-бітний ключ. Спочатку ключ подається на вхід функції перестановки. Потім для кожного з 16 раундів підключ K_i є комбінацією лівого циклічного зрушення й перестановки. Функція перестановки та сама для кожного раунду, але підключи K_i для кожного раунду виходять різні внаслідок повторюваного зрушення біт ключа.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у бакалаврської дипломної роботі програмне забезпечення системи Інтелектуального Software-Defined Power для дата-центрів. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функції представлені у графічному вигляді (іконки).

– Розділу виведення результату роботи системи.

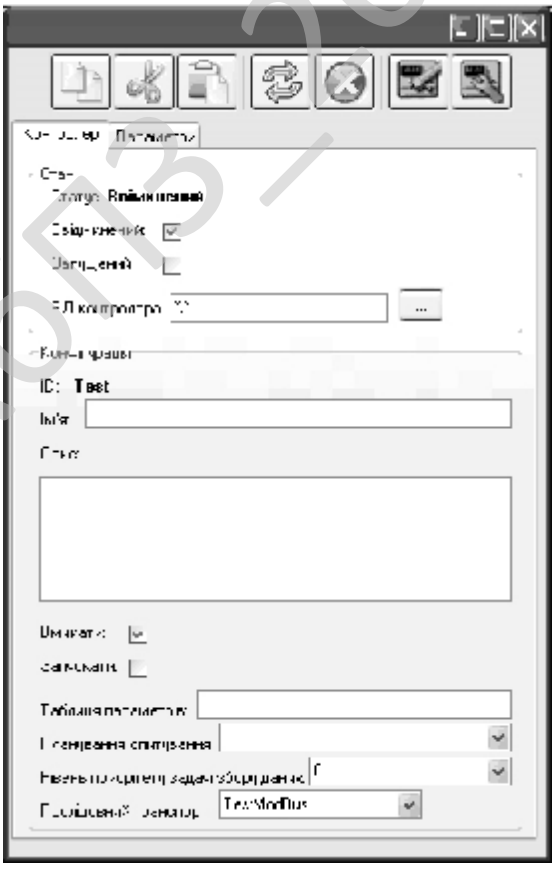

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і 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.

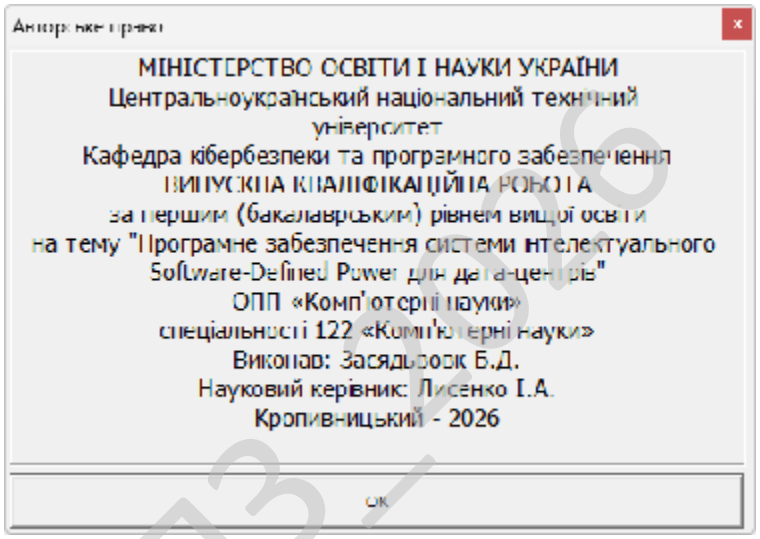

Рисунок 5.2 – Авторське право

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 IT рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене IT 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.

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 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи інтелектуального Software-Defined Power для дата-центрів.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем інтелектуального Software-Defined Power для дата-центрів.

– Досліджена система інтелектуального Software-Defined Power для дата-центрів.

– На основі отриманих результатів досліджень створена програмна реалізація системи інтелектуального Software-Defined Power для дата-центрів.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання інтелектуального Software-Defined Power для дата-центрів.

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 Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи інтелектуального Software-Defined Power для дата-центрів. 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 DES.

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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

2. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

3. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

4. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп'ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». *Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка»* (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.

5. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». *International Review on Modelling and Simulations* 18 (1), 2025. pp. 32-42.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

12. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

13. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

14. 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.

15. 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.

16. 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.

17. 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.

18. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

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

20. 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.

21. 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.

22. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and

Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

23. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

24. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

25. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645.

26. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». *International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019*; Odessa; Ukraine; 9-13 September 2019. P.22-28.

27. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

28. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

29. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

30. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

31. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

32. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

33. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

34. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

35. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

36. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

37. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у

комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

38. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

39. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

40. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

41. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

42. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

43. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

44. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

45. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

46. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

47. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

48. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

49. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

50. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". - Випуск 5 (142). - Х.: ХУПС - 2016. - С. 148-152.

51. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.