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
“____” _____ 2021 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-20М-1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Хлистун В.В.
«____» _____ 2021 р.

Керівник проекту
кандидат технічних наук
_____ Олександр ДОРЕНСЬКИЙ
«____» _____ 2021 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

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

Спеціальність *123 “Комп’ютерна інженерія”*

Освітньо-професійна (освітньо-наукова) програма _*“Комп’ютерна інженерія”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2021 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Хлистуну Владиславу Васильовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu*

2. Керівник роботи *Доренський Олександр Павлович, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 42-13 від 02.08.2021 року

3. Строк подання студентом роботи до захисту *10.12.2021 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu*

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

1. Призначення та область використання.	*7. Економічна ефективність розробленої*
2.Перегляд аналогічних існуючих систем.	*програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	
6. Наукова новизна	

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2021	14.11.2021
Охорона праці	Оришака О.В.	06.10.2021	16.11.2021

7. Дата видачі завдання « 6 » вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2021 р.	
3.	Розробка моделі компонента	20.10.2021 р.	
4.	Розробка структур даних	25.10.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2021 р.	
6.	Програмування алгоритмів	10.11.2021 р.	
7.	Розрахунок економічної ефективності	13.11.2021 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2021 р.	
9.	Оформлення ПЗ	17.11.2021 р.	
10.	Попередній захист роботи	10.12.2021 р.	

Дата видачі завдання
« 6 » вересня 2021 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2021 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Хлистун В.В. Дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Метою розробки є дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Об'єктом дослідження є процес програмно визначаємого ЦОД на базі технологій Fujitsu.

Предметом дослідження є методи програмно визначаємого ЦОД на базі технологій Fujitsu.

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

Результат роботи – програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu.

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 архітектури IBM PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено в середовищі RAD Studio Delphi 10.3.2.

Ключові слова: комп'ютерна інженерія, програмно визначаємий ЦОД, Fujitsu

ABSTRACT

Khlystun V.V. Research and software implementation of a software-defined data center system based on Fujitsu technologies. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this final qualification work on the second (master's) level of higher education the software which is intended for system of the software defined data center on the basis of Fujitsu technologies is developed.

The purpose of development is research and software implementation of the system of software-defined data center based on Fujitsu technologies.

The object of research is the process of software-defined data center based on Fujitsu technologies.

The subject of the research is the methods of software-defined data center based on Fujitsu technologies.

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

The result is a software implementation of a system-defined data center based on Fujitsu technologies.

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.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in the environment of RAD Studio Delphi 10.3.2.

Keywords: computer engineering, software-defined data center, Fujitsu

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ДПТМ	–	динамічний перерозподіл трафіку мережі
ЕВЕ	–	ефективність використання електроенергії
ЕЦП	–	електронний цифровий підпис
ДБЖ	–	джерело безперервного живлення
ІВК	–	інфраструктура відкритих ключів
ІТ	–	інформаційні технології
ЛОМ	–	локальна обчислювальна мережа
ПЗ	–	програмне забезпечення
СЗД	–	системи зберігання даних
СКЗІ	–	система комплексного захисту інформації
УК	–	управляючі компоненти
УЦ	–	удостовірюючий центр
ЦОД	–	центр обробки даних
DCIM	–	Datacenter Infrastructure Management – керування інфраструктурою ЦОД
DMaaS	–	Datacenter Management as a Service – керування дата-центром як послуга
IGMP	–	Internet Group Management Protocol
NLB	–	Network Load Balancing, балансування мережного навантаження
PKI	–	Public Key Infrastructure
VPN	–	віртуальна приватна мережа

ВСТУП

Актуальність теми. Цифрова трансформація робить весь зростаючий вплив на бізнес і суспільство. При цьому самі інформаційні технології змушені мінятися прискореними темпами, щоб не відстати від прогресу. Одне із ключових змін в IT виражається в тому, що, програмне забезпечення змінює світ. За останнє десятиліття цей процес зайшов досить далеко, але за легкою доступністю застосунків і сервісів ховаються невидимі для більшості змін в інфраструктурі.

Майбутнє ЦОД – за гібридними рішеннями. Хмари прийшли, щоб залишитися. Fujitsu не збирається конкурувати з мегапровайдерами хмарних послуг, такими як Amazon, Google, Microsoft, які надають доступні послуги в необмежених обсягах.

Сервіси з оплатою в міру використання корисні для деяких областей IT. В Fujitsu є подібна глобальна хмарна система з оплатою за використання за назвою Fujitsu Cloud Service K5, але вона орієнтована на азіатські ринки. Вона присутня на європейському й американському ринку, але скоріше допомагає нам надавати власні рішення, чим конкурує із продуктами вищезгаданих компаній.

Fujitsu фокусується на гібридних рішеннях. Fujitsu є постачальниками Azure Stack і VMware Cloud Foundation, в Fujitsu також є власна версія всіх необхідних механізмів керування, за допомогою яких ЦОД із традиційною формою володіння – як корпоративні, так і орендовані – можуть взаємодіяти із хмарними сервісами.

Однак нікому більше не цікава інфраструктура, оскільки ключовим фактором є мобільність застосунків. Замовники виходять із того, що необхідна інфраструктура вже є (хоча якщо її ні, у вас великі проблеми). Вони хочуть говорити не про інфраструктуру, а про застосунки й мобільність рішень.

Як показує аналіз ринку, навантаження вертаються із хмар назад у корпоративні ЦОД – і це характерно як для американського, так і для

європейського ринку. Це викликано побоюваннями щодо інформаційної безпеки, незадоволеністю часом відгуку й, як це не здасться дивним, високими витратами. Нерідке розгортання гіперконвергентної інфраструктури на площадці замовника виявляється вигідніше. Таким чином, є цілий ряд стимулів, щоб залишити деякі рішення в традиційних ЦОД.

У випадку, коли ключове значення має мобільність застосунків, типовою моделлю є розробка в хмарі, по завершенні якої застосунок вертається на площадку замовника й там же виконується протягом усього свого життєвого циклу. Це пов'язане з тим, що ціна розробки не настільки критична, як сам ЦОД, продуктивність і наступні витрати.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu.

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

– Огляд існуючих систем програмно визначаємого ЦОД на базі технологій Fujitsu.

– Дослідження системи програмно визначаємого ЦОД на базі технологій Fujitsu.

– Програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Об’єктом дослідження є процес програмно визначаємого ЦОД на базі технологій Fujitsu.

Предметом дослідження є методи програмно визначаємого ЦОД на базі технологій Fujitsu.

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

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

– Удосконалено метод програмно визначаємого ЦОД на базі технологій Fujitsu.

– Розроблено вітчизняний продукт програмно визначаємого ЦОД на базі технологій Fujitsu, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі програмно визначаємого ЦОД на базі технологій Fujitsu.

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

Робота апробована на LV Науково-технічна конференція здобувачів вищої освіти «Наука – виробництву», 2021, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №12.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмно-визначаємі мережі й віртуалізація мережних функцій – досить зрілі технології в плані реалізації функціональності L3 у центрах обробки даних. Однак жодна з них не пройшла повноцінну перевірку на відповідність загальним критеріям безпеки IT (Common Criteria). Останні являють собою механізм, за допомогою якого регулювальні органи можуть переконатися в безпеці систем. Реалізація їхніх вимог у програмному середовищі надзвичайно важке й коштовне.

Програмно-визначаємі компоненти важливі, але вони далеко не весь арсенал рішень. Так, програмно-визначаємі сховища далекі від зрілості. Я б не радив розміщати на них критичні навантаження. Разом з тим їх цілком можна довірити гіперконвергентній інфраструктурі. Стек Fujitsu VCF або стек Nutanix – це зовсім іншу справу, оскільки вони реалізують наскрізний контроль помилок. Гіперконвергентне рішення цілком життєздатне, у всякому разі для деяких навантажень, але цього, однак, не можна сказати про окремих програмно-визначаємих сховища, оскільки високорівневі мережні компоненти ще не готові. Взяти, наприклад, такі рішення на базі відкритого вихідного коду, як пропоновані Red Hat – у масштабних середовищах вони не працюють.

Хоча мені неодноразово намагалися пояснити, але я так і не зрозумів, що таке програмно-визначаємий сервер. Це якась фікція. Таким чином, у програмно-визначаємих рішень є своя ніша, але вони ще недостатньо зрілі. Гіперконвергентні рішення більше зрілі – той самий програмний стек реалізує функціональність серверів, мережі й зберігання. Однак і вони підходять не для всіх завдань – наприклад, не здатні ефективно здійснювати віддалене тиражування даних.

Таким чином, мова може йти про окремі елементи, але не про цілісне рішення.

1.2 Область застосування

Програмно-визначаємі центри обробки даних характеризуються не своїм наповненням, а способом керування ними. І це багато в чому вже реальність. Тут у гру вступає гібридний підхід. Якщо центр обробки даних здатний надавати застосунки, які можуть виконуватися як у хмарі, так і на площадц і замовника, такий ЦОД по визначенню є програмно-визначаємим. І за його функціонування відповідають програми керування. Без сумніву, програмно-визначаємий ЦОД – важливий елемент цифрової інфраструктури майбутнього.

В остаточному підсумку апаратне забезпечення – це механізм виконання коду. Воно, звичайно, важливо, але час, коли застосунки залежали від платформ, на яких вони виконувалися, у минулому. Модель вертикального масштабування вмирає, тоді як модель горизонтального масштабування процвітає. Деякі застосунки пишуться таким чином, що для них необхідна багатопроцесорна обробка, але для цього є HPC.

Невелике застереження. Самі серверні архітектури міняються. Чи чули ви про дезагрегації? Давайте почнемо спочатку. Що таке сервер? Це гра в скільки знадобиться процесорів, пам'яті, портів для виконання типового завдання. При цьому обмеження визначаються не завданням, що треба буде розв'язати, а технологічною можливістю реалізувати необхідний функціональний набір.

У дійсності ж користувачі просто хотіли б мати досить апаратних ресурсів для виконання поточного завдання. Дезагрегацію можна розглядати як апаратний еквівалент програмної віртуалізації. Який би гіпервізор не застосовувався, він надає механізм для спільного використання наявних ресурсів сервера для рішення різних завдань. Але ефективність програмної віртуалізації обмежена, оскільки гіпервізор працює поверх доступної апаратної архітектури.

До появи гіпервізорів рівень використання ресурсів сервера був дуже низьким і не перевищував 12%. Сервер використовувався під конкретне завдання – у якості Web-сервера, сервера бази даних і т.п. Якщо він справлявся зі своїм завданням, те й чудово – нікого не турбувало, якщо він простоював.

У Великобританії є Департамент по податках і зборам Її Королівської Величності (Her Majesty's Revenue and Customs, HMRC). Ця урядова організація ввела процедуру самостійної оцінки. Кожний громадянин повинен заповнити форму, де потрібно вказати свої доходи, активи й т.д. – на основі цих даних розраховується податок. За законом дані повинні бути надані 10 січня (я не пам'ятаю точно, але це не суть важливо). Як неважко догадатися, декларації заповнюються в останній момент – 97% платників податків роблять це в останній тиждень.

Для прийому декларацій була створена величезна система, вона розрахована на пікове навантаження й розміщається в декількох ЦОД. Уряд не може собі дозволити, щоб система виявилася непрацездатної в піковий період, оскільки від цього залежить його дохід. Іншу частину року вона не робить нічого, крім як обігріває ЦОД. Це найбільш показовий приклад неефективності використання устаткування, що тільки можна привести.

Зміни неминучі. Незабаром у серверах з'явиться енергонезалежна оперативна пам'ять – цього року Intel повинна випустити пам'ять Apache Pass (Optane DIMM). Пам'ять буде надзвичайно швидкою й розташовуватися близько до процесора. Поряд із кремнієвою оптикою й подальшим розвитком програмно-визначаємих підходів, її поява буде сприяти дезагрегації – звичних «коробок» більше не буде. Процесори будуть перебувати в одному місці, пам'ять – в іншому, ввод-вивід – у третьому. Всі ці компоненти будуть розподілені так, як зручно замовникові, і зв'язані високошвидкісними з'єднаннями аналогічно тому, що в часи мейнфреймів називалося когерентною моделлю. Під конкретне завдання буде виділятися необхідний обсяг ресурсів апаратним гіпервізором.

Таким чином, апаратне забезпечення також зміниться. Але питання не в ньому. Головне – де перебуває мій застосунок, як воно надається клієнтам, яке час відгуку, як виконуються вимоги захисту даних і т.д.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

EcoStruxure IT – хмарне програмне забезпечення для керування інфраструктурою ЦОДів

Система дистанційного контролю – Asset Advisor

Відкрийте для себе хмарну систему цілодобового дистанційного контролю. Засновані на актуальних дані рекомендації від нашої команди сервісного обслуговування допоможуть вам краще контролювати свою систему.

EcoStruxure IT Expert

Підтримуйте безперебійну роботу системи й управляйте сигналами тривоги за допомогою хмарного моніторингу. Користуйтеся можливостями безпечного контролю системи з будь-якої точки світу.

Комерційно нейтральний моніторинг на базі хмарних рішень

Профілактичні рекомендації на основі даних про продуктивність і сигнали тривоги – для безпечного контролю, де б ви не перебували.

Інформація на основі даних

Відповідність галузевим стандартам для поліпшення розподілу ресурсів, планування, резервування й продуктивності.

Звітність

Побудова графіків на основі даних декількох датчиків, що налаштовуються панелі керування, відстеження аудитів, керування життєвим циклом активів для запчастин і гарантій.

APC Data Center Infrastructure Management, (DCIM)

До складу рішення **APC Data Center Infrastructure Management(DCIM)** входять системи-модулі:

StruxureWare Data Center Operation

Керування ресурсами й документування роботи ЦОД за допомогою керування запасами.

Рисунок 2.1 – Інтерфейс користувача StruxureWare Data Center Operation

Оперативний контроль операцій у центрі обробки даних за допомогою керування інфраструктурними ресурсами, калькулятора ефективності використання електроенергії (EBE), відображення інформаційних сигналів від пристроїв у реальному режимі часу й багаторівневого подання інформації з можливістю деталізації по окремих зонах.

Модуль InfraStruXure Operations підтримує незалежне від виробника пристроїв керування ресурсами інфраструктури з моніторингом стану пристроїв у

реальному часі й відображенням даних на схемі фізичного компонування центра обробки даних, а також надає рекомендації з дозволу проблемних ситуацій. Багаторівневе подання з можливістю деталізації забезпечує структурований огляд окремих зон центра обробки даних – як глобальне, так і локальне подання з деталізацією до окремих пристроїв. Калькулятор ефективності використання електроенергії (ЕВЕ) надає інформацію із щоденного споживання електроенергії. Для оперативних відновлень «на ходу» пропонується пристрій InfraStruXure Mobile, що забезпечує бездротовий доступ до інформації InfraStruXure Operations.

Data Center Operation: Capacity

Моделювання, планування й оптимізація інфраструктури для підбора оптимальної потужності ЦОД.

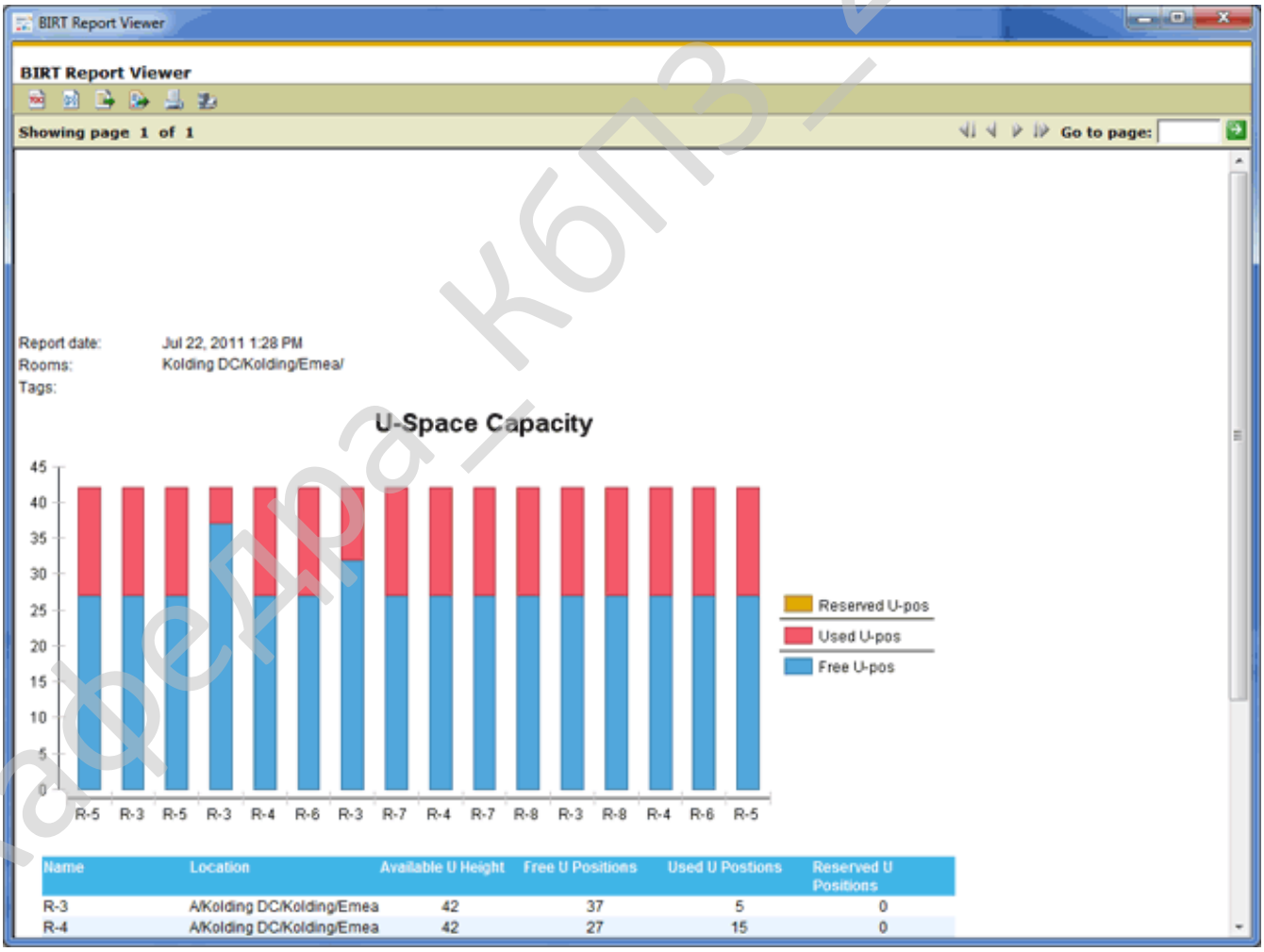

Рисунок 2.2 – Інтерфейс користувача Data Center Operation: Capacity

Планування й оптимізація використання існуючих ресурсів інженерної інфраструктури на основі моделі спільного використання центра обробки даних дозволяють ефективно вибирати схеми підключення й визначати номінали інфраструктурного устаткування.

Рішення InfraStruXure Capacity визначає оптимальне розміщення інфраструктурного й IT-устаткування з урахуванням вільного місця й потреби в ресурсах; ураховуються задаються також користувачем вимоги резервування, групування за ознаками використання й об'єднання в мережі. Завдяки оптимальному використанню інженерної інфраструктури мінімізуються загублені ресурси й непланові простої. Ефективне моделювання на основі реальних даних дозволяє точно прогнозувати ефект передбачуваних змін. Одержувані дані використовуються для прийняття рішень і планування розраховуючи на поточні й майбутні вимоги до ресурсів інфраструктури.

Data Center Operation: IT Optimize

Одержання інформації про енергоспоживання IT-систем і їхньому використанні, необхідної для підвищення ефективності й зниження витрат.

Скорочення енергоспоживання IT-системи за рахунок глибокої оптимізації використання серверів з погляду підвищення характеристик ЦОДу

StruxureWare Data Center Operation: компонент IT Optimize забезпечує підвищення відсотка використання інфраструктури й IT-активів за рахунок складання точних профілів енергоспоживання ЦОДів з деталізацією до рівня окремої стійки й окремого сервера.

Data Center Operation: Change

Повністю інтегрована система документообігу для інженерної інфраструктури IT. Функція обліку наявного устаткування дозволяє з легкістю відслідковувати й виконувати переміщення, установку й модернізацію устаткування в центрі обробки даних.

Модуль InfraStruXure Change дозволяє операторам одержати повний контроль над інфраструктурою центра обробки даних за рахунок впровадження

організованої системи для переміщення, додавання й модернізації робочих процесів, що значно знижує ризик позапланового простою. За допомогою автоматизованої системи документообігу оператори можуть формувати вбрання на виконання робіт, резервувати простір, відслідковувати стан і одержувати вибірки з журналу реєстрації подій для повного візуального контролю й ведення історії змін. Мобільний пристрій InfraStruXure Mobile (опція) дозволяє оперативно реєструвати зміни з будь-якого місця центра обробки даних, підтримує сканування штрих-кодів і забезпечує цілісність даних, а також поліпшення експлуатаційної ефективності.

Data Center Operation: Energy Efficiency

Автоматизований аналіз ефективності використання електроенергії (EBE/шкала DCiE) на рівні підсистем.

Повна картина про витрати на споживану електроенергію, а також про поточну і історичну енергоефективність для всього об'єкта, визначення втрат ефективності й поліпшення показника ефективності використання електроенергії (EBE/шкала DCiE) на рівні підсистем.

Модуль InfraStruXure Energy Efficiency надає поточні й історичні значення показника ефективності використання електроенергії (EBE), що дає обґрунтоване подання про те, яка потужність направляється на роботу встановленого IT-устаткування в порівнянні із загальним енергоспоживанням на об'єкті. Дається деталізоване подання про те, наскільки ефективно використовується електроенергія до рівня підсистем, а також надаються рекомендації з поліпшення енергоефективності. Дані про підсистеми можуть вимірятися безпосередньо або оцінюватися за непрямими показниками, що дозволяє замовникам, що не розташовують вимірниками потужності, користуватися всіма перевагами застосунки. У веб-формі моніторингу відображаються, зокрема, показники ефективності по поточних і історичних значеннях EBE, а також докладний аналіз вартості споживаної електроенергії на рівні підсистем. Модуль InfraStruXure Energy Efficiency доступний через модуль InfraStruXure Operations, що підтримує

інтеграцію із сервером InfraStruXure Central і системами керування корпоративною інфраструктурою сторонніх розроблювачів.

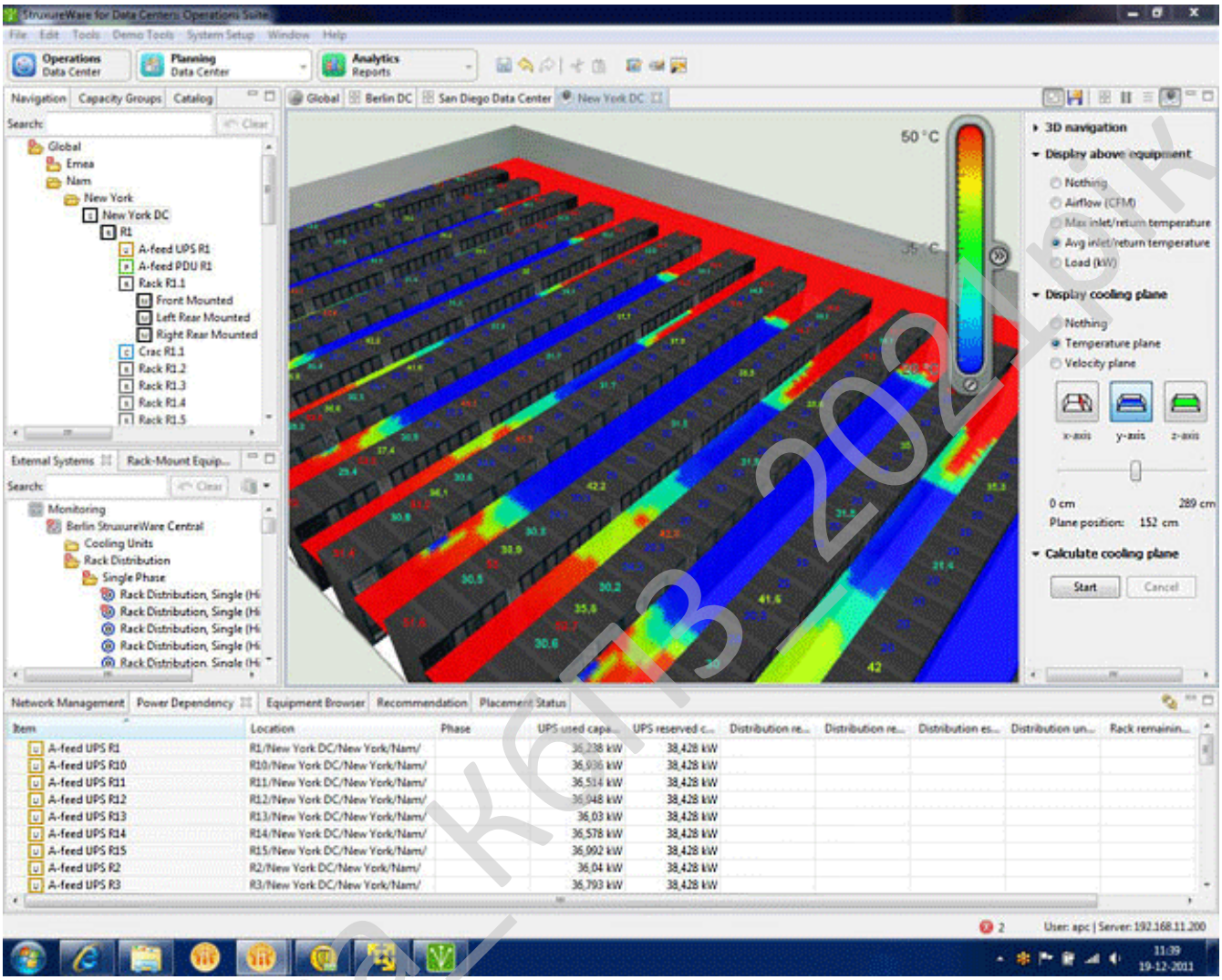

Рисунок 2.3 – Інтерфейс користувача Data Center Operation: Energy Efficiency

Data Center Operation: Energy Cost

Оперативна візуалізація енергоспоживання в стійках.

Аналіз вартості споживаної електроенергії в кіловат-годинах з деталізацією до рівня стійки для розрахунку рівня витрат на електроживлення заданого устаткування, об'єктивного виставляння рахунків споживачам і підвищення фінансової ефективності.

Модуль InfraStruXure Energy Cost формує звіт по використанню електроенергії, що показує споживану в центрі обробки даних електроенергію в кіловат-годинах і її вартість по поточному тарифі, з деталізацією на рівні стійки. Показники енергоспоживання засновані на обмірюваних даних, зібраних протягом заданого періоду часу. При відсутності обмірюваних даних очікуване енергоспоживання буде розраховуватися на основі енергоспоживання окремих IT-ресурсів або номінальних значень. У звіті по використанню електроенергії (Energy Usage Report) є можливість включення коефіцієнта накладних витрат, що дозволяє врахувати енергетичні втрати при розрахунку показника ефективності використання електроенергії (EBE). Цей звіт може настроюватися індивідуально з використанням угруповання по окремих категоріях, наприклад, по департаментах, орендарям, призначенню, щільності й т.д.

Data Center Operation: VIZOR

Високорівневі ключові параметри центра обробки даних доступні «на ходу» через планшетний комп'ютер або смартфон.

Перегляд ключових параметрів ЦОДу: характеристики електроживлення, охолодження, робітника простору й локальної мережі, а також високорівневі показники використання ресурсів ЦОДу, які направляються на ваш смартфон або планшетний комп'ютер

Застосунок StruxureWare Operations: VIZOR надає ключові параметри ЦОДу: характеристики електроживлення, охолодження, робітника простору й локальної мережі, а також прогнози резерву потужності з урахуванням поточних темпів росту, які направляються прямо на ваш смартфон або планшетний комп'ютер. Завдяки його простому інтерфейсу користувача радикально спрощується завдання спостереження за станом вашого центра обробки даних «на ходу» – за допомогою функції деталізації ситуації в будь-якій зоні або кімнаті, що забезпечує повне візуальне подання про стан всіх ресурсів центра обробки даних. Для спрощення ідентифікації ресурсів можна сканувати штрих-коди за допомогою убудованої камери. Застосунок StruxureWare Operations:

VIZOR здійснює обмін даними із серверами StruxureWare Operations і є одним зі значимих компонентів програмного комплексу Schneider Electric Data Center Infrastructure Management (DCIM), даючи мобільному користувачеві повне візуальне подання про використання ресурсів ЦОДу. Модуль StruxureWare Operations: У цей час застосунок VIZOR доступно для пристроїв Apple iPhone і iPad, а також смартфонів на платформі Android.

Data Center Operation: Mobile

Бездротове керування центром обробки даних.

Кишеньковий комп'ютер зі сканером штрих-кодів і підтримкою бездротової мережі для перегляду, створення й оперативної синхронізації змін «на ходу». Пристрій побудований на базі КПК Motorola Symbol MC70.

Пристрій InfraStruXure Mobile, побудоване на апаратній платформі Motorola Symbol MC70, дозволить вам працювати з базою пристроїв центра обробки даних, не використовуючи стаціонарний комп'ютер. Інтегрований сканер штрих-кодів полегшує роботу з виконання вбрань на роботи й визначенню устаткування. Використовуючи вашу бездротову мережу, пристрій InfraStruXure Mobile автоматично синхронізує дані про розташування серверів, забезпечуючи цілісність даних, запобігаючи помилки операторів і підвищуючи експлуатаційну ефективність.

Data Center Operation: Insight

Повнофункціональний інструмент індивідуального складання звітів для візуалізації даних.

Генератор звітів, легко адаптуємий до індивідуальних потреб, що відслідковує історичні дані по використанню ресурсів ЦОДу й який надає повне подання про ключові показники продуктивності ЦОДу.

Модуль StruxureWare Operations Модуль Insight – зроблений інструмент розробки звітів, що надає необмежену кількість варіантів індивідуальної адаптації й візуалізації звітів по ЦОДу відповідно до потреб вашого бізнесу. Цей модуль надає шаблони настроювань за замовчуванням, джерела даних, набори

даних і параметри фільтра, даючи можливість формувати індивідуальні рішення. Звіти, створені в середовищі StruxureWare Operations: Insight дозволяють поєднувати дані StruxureWare Operations з будь-якими зовнішніми джерелами даних, доступними через веб-служби або бази даних. Будь-який звіт можна опублікувати на сервері StruxureWare Operations або перетворити у файл одного з безлічі доступних форматів.

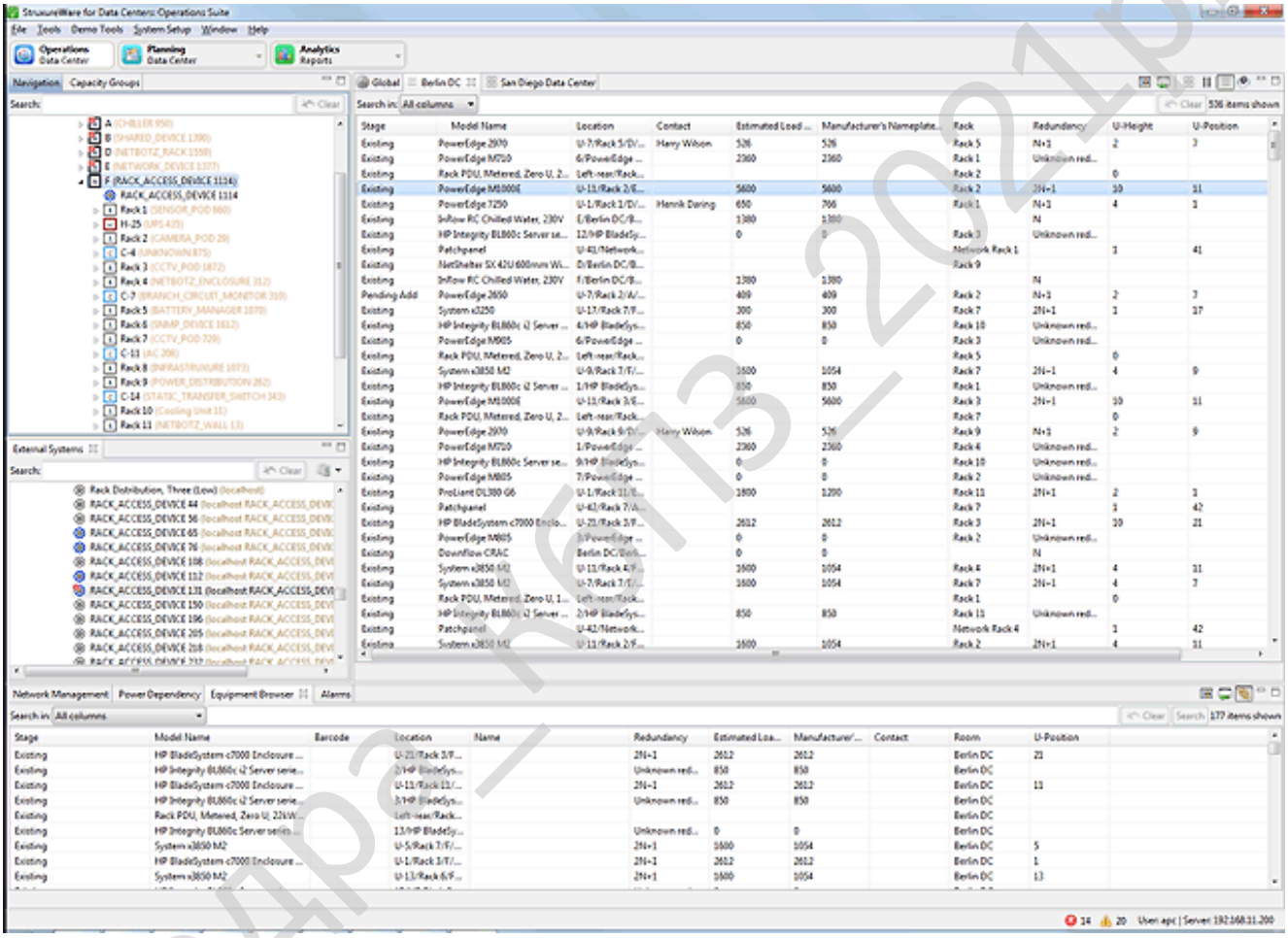

Рисунок 2.4 – Інтерфейс користувача Data Center Operation: Insight

DCIM – засіб продовження строку життя ЦОД

Ефективне використання вільного місця ЦОД стає критичним фактором у продовженні строків його експлуатації. Проблема пов'язана з безперервним ростом енергетичного навантаження: нове обладнання займає багато місця –

його закупівлі не завжди виправдані, якщо можна обійтися вертикальною модернізацією. Для рішення завдання потрібні точні розрахунки, засновані на безлічі параметрів.

Необхідна ефективна система, що дозволяє контролювати всі аспекти існування ЦОДу – від параметрів IT-устаткування (обчислювальні потужності, системи зберігання даних, мережні параметри) до інфраструктури, що забезпечує (електроустаткування, охолодження, вентиляція, протипожежні системи й системи безпеки) для виходу на оптимальні показники функціонування дати-центра. Системи такого типу, називані DCIM (Datacenter Infrastructure Management – керування інфраструктурою ЦОД), існують і розвиваються з 2006 року й пропонуються багатьма вендорами устаткування ЦОДів.

Опис

Основне завдання DCIM-систем – оптимізація потужностей дата-центра, його охолодження й займаного устаткуванням місця. Збір інформації реалізується інтеграцією до складу DCIM, як більше старих систем керування інфраструктурою будинку (Building Management System), так і шляхом установки власних датчиків, що контролюють критичні параметри (вологість і температура) навколишнього середовища дати-центра. DCIM-система допомагає поліпшити керування ресурсами, дає можливість створити систему керування інформаційними сервісами, оцінити рівень зрілості процесів керування інформаційними технологіями по методології ITIL (Information Technology Infrastructure Library).

Варіанти реалізації

Найбільш розвинені системи містять можливості тестування різних сценаріїв, наприклад при розширені ЦОДу.

– Чи вистачить потужності ДБЖ при установці додаткових серверних стійок?

– Чи впораються системи охолодження?

Отримана інформація, укупі з варіантами керування, повинна подаватися у двомірному виді й в інтерактивному 3D-вигляді, на одному великому екрані. Всі статистичні показники повинні відображатися на панелях моніторингу – вкладках, кожна з яких відображає різні показники й дозволяє дати оцінку ситуації в дата-центрі, як на рівні окремих обчислювальних вузлів (або інженерного устаткування), так і по ЦОД. Це дозволить операторові бачити що відбувається, оцінювати й оперативно реагувати.

Alsok пропонує робота Reborg-Z для моніторингу дата-центра

Інженери японської компанії Alsok запропонували проектувальникам, будівельниками й операторам ЦОД із усього світу новий інструмент для підвищення безпеки серверних ферм. Мова про робота Reborg-Z.

Машина здатна здійснювати патрулювання машзалів, моніторинг устаткування й інші процедури в центрі обробки даних. Охоронний робот оснащений датчиками для виявлення небезпечних газів і має функцію пожежогасіння.

Пристрій уже був придбаний операторами ряду японських ЦОД. Зокрема, робот уже використовується в недавно відкрившемося дата-центрі компанії Internet Initiative Japan потужністю 50 Мвт у місті Широи, префектура Тиба (Японія).

Schneider Electric обновляє платформу EcoStruxture IT Advisor для аналітичного моніторингу ЦОД

Компанія Schneider Electric випустила нову версію своєї програмної платформи EcoStruxure, що спрощує керування інфраструктурою центрів обробки даних, а також планування використання ресурсів і моделювання дата-центрів.

Рішення з підзаголовком EcoStruxure IT Advisor, було представлено на заході DCD New York 2019. Розроблювачі змогли додатково спростити розгортання платформи, додавши можливість установки її як у хмарі, так і на

локальному сервері. Клієнтам було запропоновано мінімізувати витрати на продукт завдяки моделі підписки.

Також був розширений функціонал продукту в сфері кібербезпеки й в інших областях. Зокрема, з'явилася можливість моделювання впливу різних інцидентів на вже встановлені в ЦОД IT-пристрої й допоміжну інфраструктуру.

APC by Schneider Electric пропонує нові системи спостереження Netbotz для дата-центра

Компанія APC by Schneider Electric представила нові продукти для ринку систем безпеки ЦОД, які можуть похвастати підтримкою Po і відеоспостереження із застосуванням 4-мегапіксельної камери. Мова про пристрої Netbotz 750 і NetBotz Camera Pod 165c.

Підтримка бездротових датчиків, який володіють продукти, усуває необхідність у прокладці стандартних мережних кабелів при розгортанні цих пристроїв. Системи були розроблені для захисту IT-середовищ від фізичних і екологічних погроз, таких як перегрів і підвищена вологість, витоки води, дим і пожежа, а також несанкціонований доступ.

Оператори ЦОД тепер можуть віддалено через єдину IP-адресу контролювати такі функції як відеоспостереження, зчитування даних з підключених датчиків і керування смарт-картами для контролю доступу. Аварійні сигнали й дані датчиків з декількох пристроїв з лінійки NetBotz можуть централізовано оброблятися в хмарної DCIM-платформі EcoStruxure IT.

Raritan демонструє сенсорну технологію для моніторингу центрів обробки даних

Компанія Raritan представила продукти наступного покоління для моніторингу центрів обробки даних. Рішення з лінійки SmartSensor покликані спростити інженерами одержання інформації середовищу усередині ЦОД у режимі реального часу. Пристрою допомагають швидко визначати «гарячі точки», оптимально прохолоджувати IT-устаткування й запобігати дорогі простої ЦОД.

Інтелектуальні датчики дозволяють контролювати температуру, вологість, напрямок повітряних потоків і тиск повітря, а також виявляти витоку води, вібрації й несанкціоновані відкриття дверцят серверної стійки.

Крім того, пристрої здатні попереджати операторів ЦОД про потенційні ризики. Датчики з лінійки SmartSensor можна легко підключити до існуючої інфраструктури живлення усередині стійки, що допомагає оптимізувати витрати й прискорити розгортання.

Постачальник систем аналітичного моніторингу для дата-центрів Romonet куплений CBRE

Компанія Romonet, що спеціалізується на рішеннях для моніторингу й оптимізації інфраструктури центрів обробки даних, була придбана фірмою CBRE, що займається наданням послуг у сфері комерційної нерухомості й інвестицій. Ціна угоди не розголошується.

Компанія Romonet розробляє хмарне програмне забезпечення для моделювання й аналітики, що дозволяє замовникам знизити енергоспоживання свого центра обробки даних і підвищити надійність ЦОД. Цей продукт може виявитися досить корисним для фірми CBRE, що управляє більш ніж 8 сотнями ЦОД по усьому світі.

Sunbird Software виводить на ринок нову версію DCIM-рішення dcTrack

Розроблювач інноваційних програмних інструментів для операторів центрів обробки даних Sunbird Software оголосив про доступність оновленої платформи для керування інфраструктурою серверної ферми (DCIM). Мова про рішення dcTrack, що обновилося до версії з порядковим номером 6.3.

Новий реліз може похвастати крім іншого наявністю вдосконаленої панелі бізнес-аналітики, що забезпечує цілісне інтегроване подання про інфраструктуру центра обробки даних.

Був поліпшений механізм інтеграції dcTrack 6.3 із хмарою й CMDB-платформами, щоб спростити об'єднання декількох систем для більше

ефективного керування ресурсами дата-центра й планування періодичних відновлень, щоб підтримувати базу даних dcTrack в актуальному стані.

У новій версії також з'явилася можливість спільного використання моніторингових панелей, що налаштовуються. Система дозволяє створити необмежену кількість персоналізованих інформаційних панелей, посиланнями на які користувачі можуть ділитися з іншими членами своєї команди й керівництвом.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero RAD Studio Delphi 10.3.2 Rio Architect – це найшвидший спосіб створювати й обновляти інтенсивно працюючі з даними, сильно взаємодіючі застосунки з візуально насиченим користувальницьким інтерфейсом для Windows 10, Mac, мобільних пристроїв, IoT і інших платформ за допомогою Object Pascal і C++. Широкий вибір функцій п ідтримки Windows 10, у тому числі нові компоненти VCL для Windows 10, стилі для VCL і FMX, а також служби UWP (універсальної платформи Windows), наприклад повідомлення, дозволяють легко й швидко перенести застосунки в Windows 10, зберігши користувачів. Нова платформа дозволяє підтримувати великі проекти на більшому числі платформ із подвоєним обсягом пам'яті в середовищі розробки й удвічі більшим розміром підтримуваних проектів. Крім того, підтримка декількох моніторів і десятки нових функцій середовища розробки, призначених для прискорення створення коду, зроблять роботу як ніколи ефективною. За допомогою RAD Studio 10 розроблювачі зможуть створювати застосунки в 5 разів швидше в порівнянні з іншими інструментами, а розробка застосунків для декількох настільних, мобільних, хмарних платформ і платформ баз даних, включаючи 32і 64 -розрядні версії Windows 10, Mac OS X, iOS і Android, стане ще швидше.

Зміни у версії 10.3 Rio:

– Створюйте міжплатформені застосунки. 80% інтернет користувачів мають смартфони й застосунки доступу, а також дані з мобільного пристрою й ноутбука / настільного комп'ютера, саме тому так важливо в цей час, щоб застосунки працювали в будь-якому пристрої.

– У всіх версіях Professional, Enterprise і Architect RAD Studio 10.3 надається підтримка процесу розробки застосунків для мобільних пристроїв. Розроблювачі RAD Studio кодують лише один раз, компілюють споконвічно для кожної платформи, що скорочує час і трудозатрати на вивчення декількох мов і дозволяє паралельно управляти циклами розробки.

– Підтримка Android API26, відповідність вимогам Google Play Store відносно нових застосунків із серпня 2018 року й відновлення застосунків з листопада 2018 року.

– Власні елементи керування Android і стилізовані елементи керування FMX в одній і тій же формі Android, включаючи тему матеріального дизайну для Android 5.0 або вище.

– Підтримка iOS 12 (32і 64-біт) для створення App Store і корпоративних застосунків.

– Підтримка смайликів Юнікод.

– Програмуйте по-своєму. Завдяки двом новим темам самостійне налаштування інтегрованого середовища розробки для відповідності вашому стилю кодування ще ніколи не була настільки простій.

– Темне й світле оформлення Незалежно від того чи волієте ви кодувати вночі або у світлий час доби, завдяки темному й світлому оформленню RAD Studio ви можете вибрати потрібний вам стиль. Було доведено, що темне оформлення допомагає знизити зорову напругу в умовах низького освітлення, дозволяючи вам працювати більш продуктивно вночі. Немає нічого простіше, ніж перейти від темного до світлого оформлення й навпаки за допомогою меню панелі інструментів.

– Виконаєте користувальницьке налаштування свого середовища розробки Поліпшена програма установки інтерфейсу користувача й менеджера ліцензій інтерфейсу користувача дозволяє визначити ті можливості, які необхідні й опустити непотрібні, незалежно від того чи розробляєте ви застосунки для декількох платформ або всього однієї.

– Чистий, оновлений інтерфейс користувача інтегрованого середовища розробки Знайдіть потрібні можливості. Швидко. Головне вікно інтегрованого середовища розробки відцентровано й відрізняється високим ступенем читаності. Ви з легкістю визначите, де перебуває область фокусування клавіатури з оновленими змінами фонових квітів фокуса. Вкладки редактора більше, що полегшує читання шрифтів, тому ви можете швидко внести зміни й зберегти кодування.

– Чудові застосунки Windows з VCL. Бібліотека візуальних компонентів (Visual Component Library, VCL) пропонує просту й візуальну розробку користувальницького інтерфейсу застосунки, у версії 10.3 представлені нові відновлення, які дозволять вашим застосункам виглядати сучасними й свіжими.

– Розширена підтримка HighDPI. Завдяки новому елементу керування VCL High DPI ImageList у версії 10.3 розроблювачі, що створюють нові застосунки VCL для Windows або оновлюючі існуючі застосунки для High DPI дисплеїв, можуть повністю підтримувати зроблені до рівня пікселів зображення зі змінною розв’язною здатністю на всіх елементах керування, а також будь-яке користувальницьке креслення, що вимагає масштабованих зображень для моніторів з різною розв’язною здатністю.

– Підтримка Per Monitor V2. Переконаєтеся, що ваш застосунок масштабується правильно для всіх типів масштабування в Windows, реагуючи на зміни масштабування DPI на різних екранах під час виконання.

– Розширена підтримка Windows 10 і WinRT API. Сюди відноситься ряд ключових API-інтерфейсів WinRT і останні API-інтерфейси Windows 10,

включаючи готові до використання компонента для убудованих у застосунки покупок і випробувань у магазині Windows 10 Store.

– Розгортання застосунків на основі служб за допомогою RAD Server. Продуктивність RAD Server була значно поліпшена завдяки десятикратному збільшенню потужності відносно простих операцій.

– Нові компоненти обробки JSON допоміжного засобу.

– Розширена підтримка RAD Server для клієнта Ext J S. Об'єднайте зовнішній інтерфейс javascript і веб-службу, підтримувану REST Server REST. (У версії Architect тепер включена ліцензія ExtJS Professional).

– Версії Enterprise включають ліцензію для одиничного розгортання RAD Server.

– Версії Architect включають ліцензію для розподіленого розгортання RAD Server.

– Що нового в C++? Підтримка C++17 Win32 збільшує продуктивність, поліпшує роботу компілятора й прискорює процес кодування. Були оновлені RTL і STL.

– Нова версія STL/Dinkumware 2018 для Win32 і Win64.

– Поліпшене автодоповнення коду Автодоповнення коду для даного компілятора тепер асинхронне, швидше й із кращими результатами, чим у попередньому автодоповненні коду C++. Ввод тексту не буде припинятися, поки виконується розрахунок.

– Тепер є підтримка налагодження для оптимізації компонувань.

– 2X швидкість математичної продуктивності для Win64.

– Нові додаткові лабораторії C++ в GetIt.

– Нові й поліпшені можливості роботи з базами даних. InterBase 2017 / IBToGo 2017 в RAD Studio. Версії Professional включають ліцензію розроблювача InterBase 2017, у той час як версії Enterprise і Architect містять у собі ліцензії InterBase ToGo. InterBase ToGo доповнена можливістю шифрування, функціями

зміни подань, призначених для простої синхронізації даних застосунку по підписці без обмежень за розміром файлу бази даних.

– Поліпшена й оновлена підтримка для популярних баз даних, включаючи mySQL v8.0, MariaDB 10.3, SQL Server 2017, PostgreSQL v10, Firebird v3.0, MongoDB, InterBase, SQLite 3.23.1, SQL Anywhere і багатьох інших.

– Удосконалення DataSnap.

– Поліпшення REST. Підтримка додаткових родинних REST методів, типів і властивостей.

– Повністю оновлений модуль живлення версії Architect. Одержіть більше від версії Architect, включаючи ці ліцензії сімейства Idera.

– Ліцензія Sencha ExtJS Professional: Створіть свій ідеальний мережний вхідний інтерфейс за допомогою javascript і ExtJS.

– Ліцензія на розгортання InterBase ToGo. Додайте сховище даних у свої застосунки за допомогою цієї гнучкої, зашифрованої бази даних, що вбудовується.

– Ліцензія для розподіленого розгортання RAD Server. Ідеально підходить для серверного застосунку архітектури мікросервісів.

– Ліцензія AquaData Studio. Вражаючий аналіз бази даних.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи програмно визначаємого ЦОД на базі технологій Fujitsu.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Останні три-чотири роки спостерігався відхід від модульної (blade) серверної архітектури – партнери Fujitsu партнери її вже практично не використовують. Причина цього – високі витрати на ввод-вивід. У випадку блейд-серверів висока обчислювальна потужність концентрувалася в невеликому обсязі, платою за це було дороге ввод-вивід. Під платою я маю на увазі не фунти або рублі, а високі накладні витрати на ввод-вивід, оскільки все компоненти було потрібно з'єднувати між собою.

На даний момент тенденцією є перехід на горизонтально масштабовані архітектури, стимулом до чого служить розвиток мережних технологій і високопродуктивних технологій. Вони будуть розвиватися доти, поки їм на зміну не прийде модель із загальною пам'яттю, про яку Fujitsu говорили вище, – ефективна розподілена модель когерентної пам'яті з фізично роздільними блоками.

До недавніх часів продуктивність підсистеми зберігання мала ключове значення при проектуванні рішення, однак незабаром це буде вже не так. Восени цього року Intel продемонструвала пам'ять Apache Pass (Optane DIMM), і вже в наступному році почнуться її масові поставки. Критичний для продуктивності ввод-вивід буде здійснюватися за допомогою шини між процесором і пам'яттю, так що це (продуктивність СЗД) не буде мати значення. Підсистеми зберігання будуть служити для забезпечення безпеки даних, зовнішньої реплікації, переміщення й міграції даних і т.п., тоді як основні ресурси зберігання, ключові для роботи застосунків, будуть перебувати усередині сервера.

Із цієї причини ніхто з великих гравців більше не інвестує в розробку більше швидких жорстких дисків, оскільки в них немає потреби – усе буде

робитися в NVRAM. Наявні диски цілком здатні справлятися з тим, що від них потрібно, – захист, реплікація, міграція. До слова, розмови про смерть магнітної стрічки йдуть стільки, скільки я себе пам'ятаю. Однак, на мій погляд, для перерахованих завдань стрічка релевантна настільки ж, як і диски. Тому для них погроза навіть більше, ніж для стрічки, хоча, звичайно, і ті й інші буде ще застосовуватися якийсь час, але навряд чи Fujitsu побачимо нове покоління 3, 5-дюймових дисків зі швидкістю обертання 22K RPM і більше високою щільністю запису.

Навіть такі виробники, як Toshiba і Seagate, інвестують у твердотільні накопичувачі. Однак і SSD – лише проміжне рішення між зовнішнім зберіганням і пам'яттю усередині сервера.

Загалом кажучи, в Fujitsu є власні мережні продукти, однак вони продаються в основному в Азії. У Європі й Північній Америці Cisco займає пануюче положення на ринку, так що Fujitsu їх там не просуває. Хоча в Fujitsu і є мережні компоненти, ринкова частка Fujitsu не настільки велика й лінійка мережного устаткування не настільки відома, щоб мало сенс витрачати сили на конкуренцію з визнаними вендорами. Однак традиційні мережні рішення поступово будуть витіснятися програмно-визначаємими, так що згодом цей сегмент стане усе менш і менш прибутковим.

В Fujitsu трохи інший підхід до ринку, ніж у великих американських компаній. Крім мережного устаткування, в Fujitsu є багато продуктів, які пропонуються тільки на японському ринку (або переважно японському ринку), такі як мобільні телефони, телекомунікаційні продукти й навіть мейнфрейми. Американської ж компанії пропонують за рубежем усе, що в них є в прайс-листі.

В Fujitsu є гіперконвергентні рішення, причому не одне, а цілих чотири:

- Azure Stack.
- VMware VCF, де використовується програмний стек VMware ESX.
- Nutanix на апаратній платформі Fujitsu.
- Рішення Fujitsu на базі Open Stack Ubuntu Linux.

Де яке варто використовувати, залежить винятково від вимог замовника.

Azure Stack переважніше, якщо замовник уже використовує Azure, але не навпаки. VCF вигідно для тих, у кого вже є корпоративні ліцензії VMware, але надмірно дорого для тих, у кого їх немає. Власне рішення Fujitsu підійде тим, хто робить ставку на рішення з відкритим вихідним кодом. Для всіх інших – Nutanix.

Замовники не хочуть більше говорити про те, як їм надавати інфраструктуру. Вони припускають її наявність як щось саме собою що розуміє. Вони виходять із того, що, маючи справу з такою компанією, як Fujitsu, HPE або будь-яким іншим великим вендором комплексних рішень, вони не повинні ні про що турбуватися й можуть зосередитися на питаннях бізнесу.

Засоби керування інфраструктурою центрів обробки даних (Data Center Infrastructure Management; DCIM) призначені для віддаленого моніторингу використання ресурсів ЦОД, стану навколишнього середовища, переміщення активів і інших параметрів, а також для регулювання окремих компонентів інфраструктури дата-центра: від вентиляторів усередині серверів до системи безпеки.

Програмне забезпечення з категорії DCIM також дозволяє генерувати звіти, спрощує планування потужностей і розподіл ресурсів, так само як і забезпечити максимально ефективне використання електроенергії, устаткування й площі.

Останнім часом усе більше широке поширення знаходить більше сучасна варіація на тему DCIM. Мова йде про рішення з категорії DMaaS (Datacenter Management as a Service або Керування дата-центром як послуга). Що таке DMaaS? Це хмарний сервіс, заснований на програмному забезпеченні DCIM.

Але це не просто версія програмного забезпечення DCIM, випущена з використанням бізнес-моделі SaaS. Концепція DMaaS виводить процес збору й обробки даних про інфраструктуру ЦОД на якісно новий рівень: дані про устаткування й пристрої збираються з безлічі центрів обробки даних, а потім – поєднуються й аналізуються.

Деякі розроблювачі програмного забезпечення також додають у свої DMaaS-рішення підтримку анонімного збору даних, що надходять із серверних ферм різних клієнтів . Такий підхід необхідний для виявлення трендів і виробітку корисних рекомендацій на основі результатів вивчення дійсно більших наборів даних.

Саме ця особливість DMaaS, на думку експертів, може потенційно трансформувати керування дата-центрами – особливо якщо можливості DMaaS-рішень у цій області будуть розширені за рахунок технології машинного навчання.

Ключовим завданням у даному контексті є застосування штучного інтелекту з метою прогнозування й запобігання інцидентів і збоїв інфраструктури дата-центрів, а також виявлення неефективності при використанні ресурсів і/або недостачі потужностей.

Два перших гравці на ринку DMaaS – Schneider Electric і Eaton. Обоє виробника одержали безліч даних із клієнтських ЦОД завдяки своєму багаторічному досвіду роботи в індустрії дата-центрів, включаючи проектування й створення модульних ЦОД і допоміжного устаткування для дата-центрів, систем розподілу електроенергії й охолодження.

Доступ до даних такого роду, що надходять від широкого кола клієнтів з різноманітними ЦОД, являє собою реальну цінність для DMaaS-рішень, які дозволяють операторам ЦОД порівнювати інфраструктуру своїх дата-центрів з погляду ефективності із глобальними контрольними показниками.

Наприклад, DMaaS-рішення EcoStruxure IT від Schneider Electric, містить контрольні показники, отримані за підсумками аналізу вихідної інформації з дата-центрів більш ніж 500 клієнтів і 2,2 млн датчиків.

Чим більше даних буде збиратися із центрів обробки даних різних типів для статистичного аналізу з використанням комбінації машинного навчання, виявлення аномалій і відтворення потоків подій, тим більше «розумними» будуть ставати DMaaS-рішення.

Наявність великих наборів даних про продуктивність конкретного устаткування в певних середовищах (при певній температурі, вологості, тиску повітря й так далі) може дозволити постачальникам DMaaS-рішень із більшою точністю прогнозувати, наприклад, ризики збоїти устаткування.

З обліком того, що простої дата-центрів шкодять підприємствам, що володіє ними, як у фінансовому, так і в репутаційному плані, легко зрозум іти передумови для зростаючої привабливості інструментів з категорії DMaaS, які можуть забезпечити мінімізацію даунтаймів і багато інших коштовних переваг.

На думку експертів, DCIM-рішення будуть програвати в конкурентній війні проти DMaaS, оскільки останні простіше розгорнути й використовувати. До того ж нові продукти забезпечують додаткові переваги. Але це лише думки. Реальні тренди будуть визначати оператори ЦОД, «голосуючи гаманцем».

Ринок програмного забезпечення для керування інфраструктурою центрів обробки даних (Data Center Infrastructure Management; DCIM) продовжує активно формуватися. Але сама концепція вже цілком оформилася з погляду визначення й очікувань клієнтів в області функціональності програмного забезпечення. Аналітики відзначають, що на даному етапі усе ще спостерігається деяка плутанина: усе більше розроблювачів ПЗ починають додавати в назви своїх продуктів акронім «DCIM», у той час як у дійсності ці рішення не здатні справлятися з усіма завданнями, які по полечу «сьогоденню» комплекту ПЗ DCIM.

Незалежне дослідницьке агентство Enterprise Management Associates (EMA) недавно опублікувала звіт про ринок DCIM, визначивши базовий понятійний апарат, якому варто використовувати під час обговорення DCIM-рішень, а також представивши список постачальників подібного ПЗ. EMA визначає комплект ПЗ DCIM як рішення, що «дає цілісне подання про IT-екосистемі й динамічно оцінює взаємозв'язку між конкретним пристроєм і всіма іншими». Це означає, що якщо в оператора ЦОД є система керування енергопостачанням, то він використовує не DCIM-рішення, а спеціалізоване

програмне забезпечення трохи іншого плану. Якщо вендор називає свій продукт для моніторингу якості навколишнього середовища «DCIM-рішенням», то він уводить клієнта в оману.

До числа самих прогресивних і успішних вендорів експерти EMA віднесли Emerson, iTRACS, Cormant, FieldView, Nlyte, Raritan і Modius. Аналітики назвали продукти ще трьох компаній «справжніми» DCIM-рішеннями, але не включили їх у доповідь, тому що виникли проблеми з одержанням інформації від цих вендорів при підготовці документа до публікації. Мова про наступні компанії: CA, IBM і Schneider Electric.

У доповіді проаналізована архітектура кожного рішення, а також легкість його інтеграції в систему, функціональність, швидкість розгортання й зручність адміністрування, цінова привабливість і впливовість вендора (популярність бренда). Аналітики також зрівняли різні продукти на основі цих критеріїв. Кращим було назване рішення Trellis, постачальником якого виступає компанія Emerson Network Power. Експерти Enterprise Management Associates назвали цей продукт найбільш комплексним, а також найпривабливішим у плані співвідношення « ціна-якість». Вони відзначили широкі можливості Trellis в області підтримки візуалізації й відображення інформації в реальному часі, а також удалий механізм інтеграції з усіма провідними технологічними платформами.

Тим менш, щоб одержати більше повну картину, у даній роботі проаналізували представників деяких вендорів зі списку EMA, щоб довідатися, скільки вони просять за с вої рішення, і в якому напрямку вони розвивають свої технології.

Ціна DCIM

Цілком природно, що першою справою клієнт звертає увагу на ціну. Вартість комплектів ПЗ, а також те, як постачальники визначали її, були одними із ключових критеріїв при порівняльному аналізі різних вендорів експертами Enterprise Management Associates. Коли ви розробляєте програмне рішення, що

підключається майже до всіх пристроїв у будинку, визначення вартості дуже швидко може стати вкрай складним і заплутаним процесом. Фахівці EMA рекомендують вендорам уникати цього шляхом розробки простих для розуміння клієнтів і простих з погляду проведених розрахунків моделей ціноутворення для DCIM-рішень.

Цінник на всі продукти, які вивчали аналітики Enterprise Management Associates, виставлявся на основі числа «кінцевих вузлів», з якими програмне забезпечення обмінюється інформацією, будь то стійкі з мережним устаткуванням, сервери або інші пристрої. Як правило, вендори розраховують експлуатаційні витрати як відсоток від загальної вартості ліцензії, при цьому деякі виробники пропонують клієнтам бонуси, такі як безкоштовне технічне обслуговування протягом року.

Тепер пройдемося безпосередньо по окремим вендорам. Так, компанії Raritan клієнт платить певну суму за кожну стойку, що обслуговується, с IT - устаткуванням у машзалі дата-центру. Вартість обслуговування стійки залежить від складності рішення, що постачальникові необхідно розгорнути в даті-центрі. Якщо рішення має багато компонентів, є більшим і складним, то воно, цілком очікувано, буде більше дорогим. Ще одним фактором є площа розгортання (кількість що обслуговуються машзалів). У цьому випадку клієнти виграють від ефекту масштабу. За словами представника компанії, розгортання DCIM-рішення Raritan на 100 серверних шаф (включаючи один рік обслуговування програмного забезпечення) буде коштувати біля $ 49 тис.

Компанія iTRACS (недавно була придбана CommScope) вибрала аналогічний підхід до ціноутворення, тобто гроші із клієнтів вона також бере за обслуговуються елементи, що, інфраструктури. Але у випадку iTRACS усе не обмежується серверними стійками – ціна може калькулюватися також виходячи із числа кондиціонерів, що обслуговуються, (CRAC), блоків розподілу електроживлення (PDU) і інших IT- і інфраструктурних активів, які розміщаються в дата-центрі. Розмір оплати у всіх випадках той самий, незалежно

від типу активу. Тут також є знижки при збільшенні сукупного розміру обслуговується системи, що.

Подібна модель ціноутворення функціонує дуже добре, тому клієнти швидко розуміють механізм її роботи. Складності в області ціноутворення, з якими зштовхнулися інші постачальники, обернулися з розвитку ринку DCIM-рішень. Раніше були рішення, ціна яких визначалася на основі потужності об'єкта або елементів інфраструктури ЦОД, а також площі машзалів, що привело до плутанини на ринку.

Набагато легше взяти масив активів, підключити до системи й визначити вартість відповідно до їх кількості. Немає даних щодо середнього розміру оплати, що iTRACS бере за обслуговуються активи, що, але є дані, що ця цифра «значно менше» $ 1000 з розрахунку на стійку.

Компанія Cormant перейшла на більш деталізований підхід до ціноутворення, якщо порівнювати з iTRACS і Raritan. Фахівці Cormant при визначенні вартості свого DCIM-рішення грунтуються на кількості реальних пристроїв, які вони також називають «активам». Джерело безперебійного живлення (ДБЖ) або систем кондиціювання повітря отут як і раніше вважається окремими активами, але серверна шафа тепер розділена на складові: один сервер або окремий мережний комутатор у випадку Cormant являють собою готельні активи.

Застосовувана компанією цінова модель краще або гірше, ніж підходи конкурентів: Зрештою, це просто трохи інша модель ціноутворення. Ключовою перевагою моделі Cormant є те, що вона дозволяє клієнтові розраховувати на більшу гнучкість, якщо його серверні стійки слабко й нерівномірно заповнені IT-устаткуванням.

Розміщення DCIM-рішення від Cormant у дата -центрі на 100 стійок при повній їхній комплектації обійдеться в $ 18 тис. Платіж разовий. У вартість не входить техпідтримка, за яку прийдеться платити окремо щороку. Техпідтримка є необов'язковою, але більшість клієнтів за неї платять.

Ще один вендор, компанія Nlyte, пропонує два варіанти оплати свого програмного забезпечення DCIM: разова оплата або підписка. Більшість компаній вибирають варіант разової оплати. Ціноутворення отут також походить із розрахунку на стійку (ціна перебуває в районі $ 1000 за стійку).

Функціонал DCIM-рішення

Відповідно до методики EMA, «сьогодення» DCIM-рішення повинне автоматизувати, як мінімум, три основні функції:

– збір даних;

– моделювання інфраструктури;

– створення аналітичної звітності.

Збір інформації, у даному контексті, означає створення централізованого сховища всіх даних, що надходять із пристроїв, і постійний моніторинг кожного з них, а також моніторинг таких речей, як споживання енергії, температура, вологість і повітряні потоки. Програмне забезпечення використовує ці дані, щоб створити цифрову модель інфраструктури, що обновляється в міру зміни інфраструктурних елементів. В ідеалі вона (модель) представлена в зручному графічному форматі. Вона є ключовим елементом DCIM-рішення, тому що відображає взаємозв'язку між пристроями усередині інфраструктури. Аналітична підсистема займається інтерпретацією даних, які збирає DCIM-рішення, дозволяючи знайти проблеми або визначити причини неефективності.

Сама по собі концепція DCIM є досить новою, тому автоматизація в кожній із цих областей продовжує розвиватися. Всі вендори зі списку EMA постійно додають нові функції у свої продукти, і ці доповнення, як правило, стосуються саме тих трьох областей автоматизації, які виділили експерти Enterprise Management Associates. Вендори розширюють можливості своїх рішень всілякими способами: розвивають безпосередньо функціонал, збільшують ступінь інтеграції своїх продуктів з іншими рішеннями або поглинають конкурентів і використовую їхні наробітки.

Поліпшення DCIM-рішень

З останніх нововведень можна відзначити наступне. Компанія Nlyte зайнялася розширенням можливостей свого програмного забезпечення в області 3D-моделювання й моніторингу електроживлення в режимі реального часу, моніторингу енергоефективності й збору даних з датчиків умов навколишнього середовища. Компанія також купила ліцензію на повноцінний движок для бізнес-аналітики, що тепер дозволяє ПЗ генерувати найрізноманітніші звіти для потреб операторів. Велика бібліотека дозволяє зробити звіти гранично зрозумілим для різнопланових фахівців, що діють у рамках організації – від операторів ЦОД до фінансових менеджерів.

Ще одна нова доробка дозволяє програмному забезпеченню Nlyte самостійно пропонувати операторові ЦОД найбільш підходяще місце для розміщення додаткових елементів інфраструктури (наприклад, нового сервера). Компанія розробила цю технологію без сторонньої допомоги й успішно неї запатентувала, а тепер пропонує іншим вендорам купити ліцензію – бажаючих поки немає.

У майбутніх релізах фахівці Nlyte планують зосередитися на підвищенні якості моделювання. На цей момент вендор випустив уже шосту версію свого програмного забезпечення. Компанія Nlyte працює на ринку DCIM біля семи років.

Рішення Cormant були доступні на ринку вже більше восьми років, хоча раніше продукти компанії з лінійки CableSolve і не позиціонувалися як DCIM. Вендор перемінив назву свого продукту з CableSolve на Cormant на початку 2013 року.

Недавно відбувся реліз сьомої версії цього комплекту програмного забезпечення. Компанія готовить нову версію практично щороку або півтора, а також допрацьовує функціональність за допомогою проміжних релізів, які попадають на ринок у кожному кварталі. З останніх доповнень можна відзначити перехід від стандартного застосунки на веб-сервіс, що, на думку аналітиків, стало

більшим кроком уперед. Крім того, зовсім недавно Cormant додала у свій продукт повноцінну підтримку планшетів і смартфонів, а також додаткові можливості в області візуалізації. Найближчим часом фахівці Cormant хочуть збільшити прогнозно-аналітичну функціональність свого продукту.

В iTRACS недавно з'явилася функція інтелектуального керування доступним простором. Крім того, компанія розширює інтеграцію свого DCIM-рішення із продуктами інших вендорів. Це дозволяє одержати нові джерела даних, за допомогою яких і здійснюється моделювання. До числа цих вендорів відносяться Intel (Data Center Manager), Power Assure і RF Code. З метою сприяння такої інтеграції iTRACS створила платформу DCIM Open Exchange Framework, що чимсь нагадує інтерфейс прикладного програмування (API). Ця платформа дозволяє продуктам сторонніх компаній підключатися до рішення iTRACS для двонаправленного обміну інформацією.

У випадку компанії Raritan, що звичайно щорічно додає по парі нових функцій з нагоди релізу чергової версії свого DCIM-рішення, технологічний розвиток зосереджений на двох речах: автоматизація й спрощення продукту для кінцевих користувачів. Недавно компанія поліпшила API для свого веб-сервісу – тепер у її програмне забезпечення можна інтегрувати більше рішень сторонніх виробників. Крім того, з'явилися спеціальні звіти й діаграми по керуванню енергоспоживанням, за допомогою яких клієнти можуть легко знайти гарячі точки в машзалі. Ще одним важливим нововведенням є диспетчер підключень, що обчислює рівень силовий навантаження в кожній точці ланцюга електроживлення.

Автоматизації на фізичному рівні

Технологія автоматизації всього дата-центра нам поки ще недоступна, але вже зараз у даній роботі можемо спостерігати усе більше активне використання робототехніки й інтелектуальних апаратних рішень у середовищі ЦОД. Роботизовані маніпулятори вже контролюють величезні стрічкові бібліотеки в дата-центрах Google (ці пристрої можуть завантажувати й вивантажувати

стрічкові накопичувачі в міру необхідності). При цьому дискусії із приводу концепції автоматизації ЦОД за допомогою робототехніки ведуть фахівці й інших великих інтернет-компаній, а також постачальників устаткування для дата-центрів. Крім того, активно розвиваються й конвергентні обчислювальні системи начебто Cisco UCS, за допомогою яких оператори ЦОД уже зараз можуть створювати потужні моделі адаптації інфраструктури відповідно до потреб поточних і нових користувачів.

Концепції « Lights-Out» у ЦОД

Робота дата-центра за принципом «Lights-Out» припускає керування IT- і допоміжною інфраструктурою в умовах відсутності фізичного доступу до цих ресурсів). Недавно цією концепцією стали цікавитися великі вендори. Приміром, компанія Panduit анонсувала новий набір послуг Industrial Automation Advisory Services, які покликані зблизити IT-фахівців і інженерів з метою підвищення ефективності підключення, керування й автоматизації промислових мереж і систем керування. В офіційному прес-релізі вендора сказано, що «для збереження конкурентоспроможності сучасним промисловим організаціям доводиться збільшувати обсяги виробництва й знижувати витрати при збереженні якості й безпеки... З урахуванням стрімкої конвергенції мереж створення адекватної фізичної інфраструктури, а також механізмів керування нею в режимі реального часу, моніторингу, збір даних і конфігурування пристроїв починають грати ще більш важливу роль».

Для повноти картини відзначимо, що згідно із прогнозом Gartner Research, у період з 2020 по 2022 роки 80 відсотків даунтаймів критично важливих ЦОД будуть викликані горезвісним «людським фактором» і прорахунками при створенні робочих процесів. При цьому більше 50 відсотків із цих відключень дата-центрів будуть викликані проблемами при зміні / конфігуруванні / інтегруванні елементів IT- і допоміжної інфраструктури. Нова ініціатива Panduit саме-таки покликана звести ризик даунтайма внаслідок перерахованих вище інцидентів до нуля. Фахівці вендора досконально вивчають поточний стан

фізичного й віртуального середовища в ЦОД клієнта й розробляють рекомендацій з оптимізації в області підвищення ефективності й автоматизації.

Автоматизація на логічному рівні

Можливості віртуалізації, хмарних обчислень і сучасного ЦОД безупинно переплітаються, що дає кінцевим користувачам нові можливості й інструменти. Автоматизація пов'язаних із цим переплетенням робочих процесів на логічному рівні має вирішальне значення. Чому? Тільки так можна динамічно контролювати приплив користувачів і підбудовувати під нього обчислювальні потужності, а також вчасно адаптувати інфраструктуру під нові типи хмарного контента й механізми взаємодії кінцевих користувачів і ЦОД.

Платформи начебто Citrix Provisioning Services або Unidesk значно спрощують розгортання віртуалізованих десктопів / застосунків і контроль над ними. Інші платформи начебто CloudPlatform, OpenStack і Eucalyptus допомагають домогтися більше глибокої логічної автоматизації, спрощуючи оркестровку хмарних платформ. З їхньою допомогою організації можуть контролювати окремі хости, кластери, різні зони й навіть основні ресурси віртуальних машин. Не будемо забувати й про системи автоматизації IT-процесів і керування конфігураціями, які пропонує Puppet Labs і ряд інших компаній. Ці рішення дозволяють адміністраторам створювати єдиний підхід до автоматизації. З таким набором інструментів IT-фахівець може управляти повністю гетерогенною інфраструктурою. Подібний інструментарій дозволяє одночасно й досить ефективно контролювати такі платформи як VMware, Amazon EC2, Juniper Networks, Google Compute Engine і навіть системи класу “bare metal”**.** Крім того, дані інструменти дозволяють організаціям застосовувати просунуті політики безпеки й дотримувати нормативних вимог.

Роль і перспективи роботів в автоматизації ЦОД

Пошуковий гігант Google із завидною швидкістю скуповує виробників робототехніки. Це факт. За останні шість місяців Google купила 8 компаній, що займаються проектуванням і виробництвом роботів. Мова про Boston Dynamics

(саме свіже придбання американців), Autofuss, Bot & Dolly, Schaft, Industrial Perception, Meka, Redwood Robotics і Holomni. Чому пошуковий гігант скуповує виробників робототехніки? Ніхто не знає напевно (за виключення, мабуть, керівництва компанії), але Google давно славиться підвищеною увагою до оптимізації й підвищення ефективності своїх дата-центрів. Використання роботів у ЦОД відкриває величезні можливості для автоматизації. Логічно припустити, що інженери інтернет-компанії вже затурбувалися адаптацією технологій куплених стартапів для дата-центрів. Не відстає від конкурента й Amazon: на складах інтернет-ритейлера вже зараз трудяться майже півтори тисячі роботів Kiva Systems (Amazon купила цю компанію наприкінці 2012 року за $ 775 млн.), які ніколи не зіштовхуються один з одним і нічого не роняють. Ціл ком можливо, що інженери компанії вже створюють щось подібне для її ЦОД.

Очевидно, що робототехніка ще не швидко зможе замінити людей усередині дата-центрів. Але завдяки появі нових і подальшому розвитку існуючих технологій (наприклад, силовимірювальних систем і систем машинного зору, RFID і т.д.) вони усе краще підходять для автоматизації повторюваного / рутинної людської праці, підвищуючи продуктивність IT-фахівців. При цьому нові технології дозволяють роботам підбудовуватися під мінливе навколишнє середовище, а не покладатися повною мірою на складену раніше модель оточення. За прикладами далеко ходити не потрібно – у даній роботі розповідали про подібні системи протягом усього року. Нижче ви можете виявити інформацію про пару свіжих проектів:

– Роботизований пристрій для ЦОД уже представили інженери IBM**.** Фахівці “блакитного гіганта” скористалися технологічною базою компанії iRobot для створення роботизованої платформи моніторингу навколишнього середовища. Їхній утвір може переміщатися усередині машзалів без втручання з боку. Після того, як машина попадає в машзал ЦОД, вона спочатку знаходить стіну, а потім починає переміщатися по периметрі приміщення, постійно вимірюючи рівень температури й вологості. В остаточному підсумку пристрій

створює теплову карту всього внутрішнього простору. Дані в режимі реального часу передаються в репозиторій спеціалізованого комплекту програмного забезпечення IBM. На їхній основі створюються комплексні звіти про тепловий режим ЦОД і рівні вологості усередині машзалів.

– Напрочуд схожий девайс за назвою DC Robot (або Data Center Robot) розробили інженери індійського підрозділу транснаціональної корпорації EMC. Завдяки бортовими камерами й роботизованою платформі для досліджень iRobot Create, що складається з надійного й недорогого мобільного шасі й ряду датчиків, DC Robot здатний вільно переміщатися по машзалам. Робот допомагає операторам ЦОД контролювати температуру, вологість і вібрації в машзалах дати-центра. Пристрій збирає дані з використанням трьох цифрових датчиків, а потім передає цю інформацію через Wi-Fi на будь -який сумісний термінал для подальшої обробки. Спеціалізоване програмне забезпечення EMC перетворить дані в теплову карту, що, у свою чергу, допомагає операторам ЦОД визначити області, де температурний режим повинен бути скоректований у ту або іншу сторону.

На думку експертів, у найближчі роки роботи будуть усе більш активно трудитися разом з людьми. Примітно, що зовсім недавно Американський національний інститут стандартизації (American National Standards Institute; ANSI) переглянув свій стандарт на системи безпеки промислових об'єктів, зм'якшивши вимоги до організації спільної роботи машин і людей. Принципово новий моментом є те, що укладачі стандарту вже не заперечують проти того, щоб роботи й люди трудилися в буквальному значенні бік-об-бік, тоді як раніше машини рекомендувалося поміщати під спеціальні огородження, при цьому люди не повинні були до них наближатися, коли роботи виконували поставлені завдання в автоматичному режимі. Устаткування для забезпечення безпеки піддалося істотному вдосконаленню з моменту останнього перегляду стандарту ANSI, і тепер роботи можуть збирати більше інформації про їхнє оточення, чим коли-або колись.

Подивимося правді в очі, Є деякі речі, які люди завжди будуть робити краще, ніж роботи. Але якщо в даній роботі зможемо розробити процеси, що дозволяють роботові взяти на себе 90% навантаження, не травмуючи при цьому людини-оператора ЦОД, коли той робить все інше, ці пристрою почнуть із величезною швидкістю з'являтися у всіх дата-центрах.

Автоматизація в майбутньому

Згідно з даними зі свіжої доповіді Cisco Cloud Index Report, «річний обсяг минаючі через дата-центри глобального IP-трафіку досягне колосальних 7.7 зеттабайта до кінця 2020 року, при цьому глобальний трафік ЦОД досягне 644 екзабайт на місяць (в 2015 році середньомісячний показник досягав лише 214 екзабайт). У доповіді також говориться, що «до 2021 року на частку хмарних платформ буде доводитися 69 відсотків сукупного минаючого через ЦОД трафіку... Розширення хмарних ЦОД буде стимулювати розвиток технологій і автоматизації, а також стандартизацію цих і суміжних технологій. Це приведе до збільшення продуктивності хмарних дата-центрів, а також до росту ємності СЗД і пропускної здатності мережної інфраструктури».

Майбутнє автоматизації дата-центрів і оркестровки припускає ще більш тісну інтеграцію між логічним і фізичним рівнем. Апаратура усередині ЦОД зможе ще більш продуктивно взаємодіяти з IT-Навантаженнями, які з її допомогою обробляються. Крім того, робототехніка й інші технології автоматизації допоможуть усунути або мінімізувати проблеми, пов'язані з відключеннями ЦОД, масштабуванням обчислювальної інфраструктури й керуванням ЦОД у цілому. Дата-Центри продовжать розвиватися й еволюціонувати, точно також будуть розвиватися й всі технології, які допомагають підтримувати їхню працездатність.

Робототехніка вже використовується на складах, у промисловому виробництві, у фармації й багатьох інших сегментах. Інтенсифікація впровадження технологій автоматизація й використання робототехніки для забезпечення нормального функціонування дата-центрів – усього лише справа

часу. Уже зараз у даній роботі спостерігаємо створення великих розподілених ЦОД, появі й розвитку яких сприяє ріст пропускної здатності мережної інфраструктури. Багато компаній останнім часом усе активніше розміщають елементи своїх розподілених серверних ферм у північних країнах начебто Ісландії, де є швидкий інтернет, дешева й екологична електрика з ГеоЕС і можливість створення високоефективних систем природного охолодження серверів (фрикулінг).

Можливість обробки великого обсягу трафіку в таких ЦОД змушує керівників організації з, приміром, південній частині Європи або Азії всерйоз розглядати можливість створення нового об'єкта або оренди вже існуючого ЦОД у регіоні зі сприятливим кліматом і розвитий телекомунікаційною інфраструктурою – нехай і розташованому на відстані багатьох тисяч кілометрів. Ці дата-центри буде рости, оскільки усе більше користувачів будуть використовувати сервери, що перебувають у їхніх стінах, і СЗД. При цьому процеси їхнього моніторингу й контролю необхідно буде змінити: традиційні підходи у випадку гіпермасштабних ЦОД банально втрачають свою ефективність.

3.2 Розробка структурної схеми

Безліч протоколів і застосунків беруть участь у забезпеченні безперебійної роботи центра обробки даних: SSL, HTTP, балансувальники навантаження, черги проміжного програмного забезпечення, бази даних і протоколи зберігання. Продуктивність, обмірювана на вхідних дверях, буде характеризувати загальне обслуговування, але для ізоляції проблем потрібна видимість всього ланцюга. Перебої в роботі або відключення критично важливих застосунків можуть вплинути на продуктивність центра обробки даних.

Моніторинг ЦОД і керування стало основою нашої економіки. У результаті робота й захист Центра обробки даних мають вирішальне значення.

Для надійного й безпечного моніторингу ЦОД прозорість має першорядне значення. Це саме те, що забезпечує програмне забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu своїм інтегрованим і уніфікованим підходом до моніторингу й керування IT-інфраструктурою, що розуміє наскрізні операційні процеси.

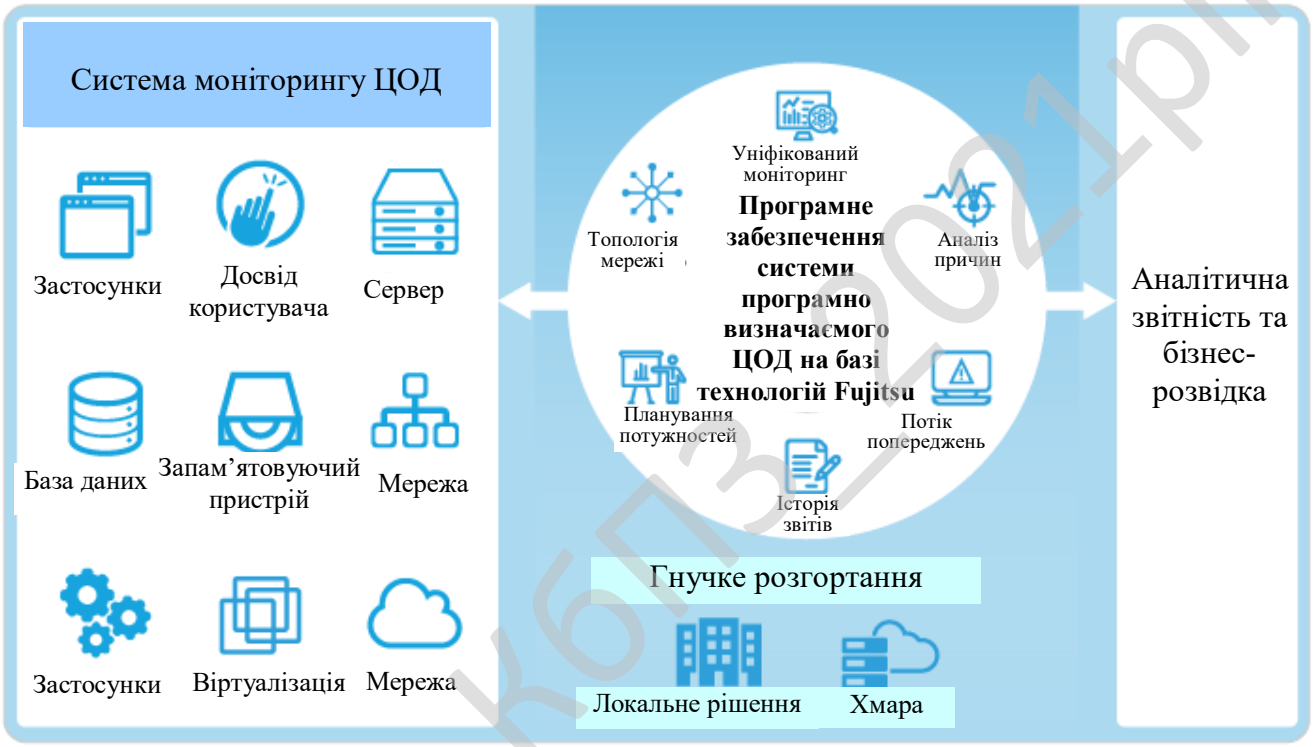

Рисунок 3.1 – Структурна схема системи

Моніторинг і оптимізація центрів обробки даних у фізичних, віртуальних і хмарних середовищах

Універсальне рішення, без схованих витрат і надбудов

Досвід єдиного моніторингу з непередбаченою простотою. За допомогою програмного забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu ви можете одночасно управляти IT-інфраструктурою, середовищем центра обробки даних і критично важливими бізнес-додатками й контролювати їх у рамках однієї консолі, що налаштовується, або панелі

моніторингу. Моніторинг продуктивності ЦОД включає синергизм продуктивності застосунків, продуктивності фізичних і віртуальних серверів і достатню пропускну здатність, що іноді може бути проблематичним. Проблема виникає щораз, коли існують розрізнені інструменти моніторингу. Програмне забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu вирішує всі проблеми керування ЦОД за допомогою єдиного рішення.

Будьте проактивно доступності & уникайте простоїв

Контролюйте свій центр обробки даних у режимі реального часу, залишіть всі свої турботи про вузькі місця, коливання або використання смуги пропущення. Усе, що вам потрібно зробити – це встановити граничні значення, і потужний механізм оповіщення програмне забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu буде попереджати вас щораз, коли щось працює не так, як очікувалося. Завдяки сторонній інтеграції програмне забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu з додатками для спільної роботи, такими як SMS, електронна пошта й т. Д., Можна уникнути незвичайних простоїв. Виявлення, виявлення, аналіз і усунення неполадок у мережі за допомогою попереджень, що попереджають, і усунення неполадок. Розуміти тенденції, моделі й поводження, щоб приймати більше обґрунтовані рішення.

Платформа росте в міру росту

Продуктивність і доступність ЦОД важливі для будь-якої організації. Через технологічний розвиток складність і розмір інфраструктури центра обробки даних продовжують рости, і у випадку невдачі підприємства страдають. Програмне забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu має відкриту архітектуру, і платформа готова до майбутнього. Платформа може бути масштабована відповідно до вимог, а також може відслідковувати поширення розподіленого середовища в різних місцях.

Моніторинг продуктивності в режимі реального часу

Забезпечте 24×7 моніторинг всіх ваших пристроїв і ефективно використовуйте ресурси в інфраструктурі ЦОД з моніторингом продуктивності й доступності в режимі реального часу. Стежите за тим, як ваша інфраструктура працює в режимі реального часу, і запобігайте дорогим збоям. Програмне забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu – «Мережа / доступ / постачальник», має можливості моніторингу на основі агентів і без агентів. Одержати повну видимість у цілому продуктивність мережі центрів обробки даних з більш ніж визначеними звітами 100.

Заощаджувати час доступності & ресурси

Щораз, коли користувачі повідомляють про повільний доступ до якого-небудь застосунки, основною причиною може бути несправність сервера, вичерпання смуги пропущення або неприступність самого застосунки. Щоб з'ясувати точну причину, IT-командам доводиться шукати на декількох моніторах і інформаційних панелях від вузьких місць пропускної здатності до сервера або застосунку. У цьому підході відсутня кореляція між даними, наданими декількома різнорідними інструментами моніторингу. На той час, коли ви зберете інформацію з даних і вистежите першопричину, застосунок уже буде закрито. Отже, аналіз кореневих причин програмне забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu вступає в гру. Знайдіть основну причину проблеми одним клацанням миші, як на багатофункціональній платформі, вона може корелювати дані метрики, потоку й журналу й видає коррелюванні метрики.

Ключові моменти

Вимір, моніторинг, керування й контроль ресурсів центра обробки даних і енергоспоживання компонентів інфраструктури об'єкта (наприклад, блоку розподілу живлення) і пристроїв IT-інфраструктури (наприклад, серверів, комутаторів і т. Д.)

Наскрізна видимість

Вимір, моніторинг, керування й контроль ресурсів ЦОД як компонентів інфраструктури об'єкта (наприклад, блоку розподілу живлення), так і пристроїв IT-інфраструктури (наприклад, серверів, комутаторів) і т. д.

Приймайте більше обґрунтованих рішень

Одержите повну прозорість всієї інфраструктури інфраструктури центра обробки даних, що дозволяє IT-адміністраторам приймати більше обґрунтовані рішення на основі важелів поліпшення роботи.

Масштаб як вам потрібно

Рішення готове до майбутнього й масштабовано. Центри обробки даних постійно ростуть, як і потреби моніторингу. Дані можуть масштабуватися й рости з вашим успіхом.

Корелювати дані

Нездатність розпізнати взаємодія або взаємозалежність різних елементів може привести до незапланованих відключень. Переконаєтеся, що ви запобігаєте це за допомогою механізму кореляції програмного забезпечення системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Автоматизувати процеси

Різко зменшити складність і людські помилки. Краще використовувати робочу силу за допомогою автоматизованого рішення для моніторингу й керування ЦОД.

Смарт усунення неполадок

Знайдіть і усунете основну причину проблеми в один клік, перш ніж вона вплине на ваших кінцевих користувачів. Активізуйте дії по усуненню неполадок.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2. Створена функціональна модель мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu дозволяє вирішити наступні завдання:

– точно визначити вхідні, вихідні й керуючі впливи на підсистему динамічного перерозподіл трафіку мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu, що надалі дозволяє провести аналітичне й імітаційне моделювання інформаційного потоку;

– сформувати наочне візуальне подання взаємодії основних процесів, що відбуваються у мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu;

– зробити чітке визначення ролі системи поліпшення функціонування в загальній системі системи програмно визначаємого ЦОД на базі технологій Fujitsu;

– сценарій динамічного перерозподілу трафіку мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu, розроблений у ході виконання магістерської роботи, є основою алгоритму імітаційного моделювання.

З рисунку видно, що розроблена система складається з наступних частин:

– Блок розрахунку вектору значень інтегрального критерію оптимізації.

– Блок ініціалізації граничних значень параметрів моделювання.

– Блок формування матриці динамічний перерозподіл трафіку мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu.

– Блок формування матриці транзитних вузлів.

– Блок формування сигналів для елементів мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu.

– Згенеровані звіти по поточному завантаженню елементів мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Інтернет-технології прийняли широке поширення, і використовуються як основа побудови корпоративних і критично важливих додатків, таких як WEB-

вузли, вузли потокового мультимедіа-віщання й сервери віртуальних приватних мереж. Служба динамічного перерозподілу трафіку мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu (ДПТМ) є оптимальним і ефективним рішенням, що забезпечує масштабованість і високу відказостійкість таких додатків як в Інтернеті, так і в інтр амережах. Служба ДПТМ дозволяє системним адміністраторам створювати кластери, що включають до 32 вузлів, між якими будуть розподілятися вступні від клієнтів запити. При цьому з погляду клієнтів кластер нічим не відрізняється від звичайного сервера; серверні додатки також не вимагають адаптації для роботі в кластері.

Рисунок 3.2 – Функціональна схема системи

Служба ДПТМ постачена всіма необхідним адміністраторам способами керування, у тому числі можливістю (після уведення пароля) віддалено управляти кластером з будь-якого комп'ютера в мережі системи програмно визначаємого ЦОД на базі технологій Fujitsu. Крім того, адміністратори мають можливість набудовувати кластер під спеціальні завдання, управляючи потоком даних на рівні портів. Вузли додаються й виключаються із кластера без припинення обслуговування. Крім того, програмне забезпечення на вузлах кластера можна обновляти без припинення обробки клієнтських запитів.

Служба ДПТМ використовує для розподілу навантаження між вузлами повністю розподілений алгоритм. На відміну від рішень на основі диспетчеризації така архітектура забезпечує високу продуктивність і низькі накладні витрати ресурсів на розподіл потоку запитів від клієнтів. Крім того, для цієї архітектури характерна висока відказостійкість (N-1) при числі вузлів N. Всі ці характеристики досягаються без необхідності використовувати спеціальні апаратні або програмні рішення.

Тести продуктивності демонструють, що використання програмної служби ДПТМ дає низькі накладні витрати на обробку потоку даних і чудові можливості масштабування продуктивності, обмежені тільки пропускною здатністю підмережі системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Служба ДПТМ демонструє пропускну здатність більше 200 Мбіт/с у реальних рішеннях по обслуговуванню електронної торгівлі з більш ніж 800 млн. запитів протягом дня.

Служба ДПТМ періодично розсилає ритмічні повідомлення, призначені для інформування кожного члена кластера про наявність інших вузлів. Збій будь-якого вузла виявляється протягом п'яти секунд, а відновлення обслуговування клієнтів виконується протягом десяти секунд.

Як при відключенні працюючого вузла, так і при додаванні нового вузла в кластер навантаження автоматично й прозоро перерозподіляється між членами кластера.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі системи програмно визначаємого ЦОД на базі технологій Fujitsu.

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.

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).

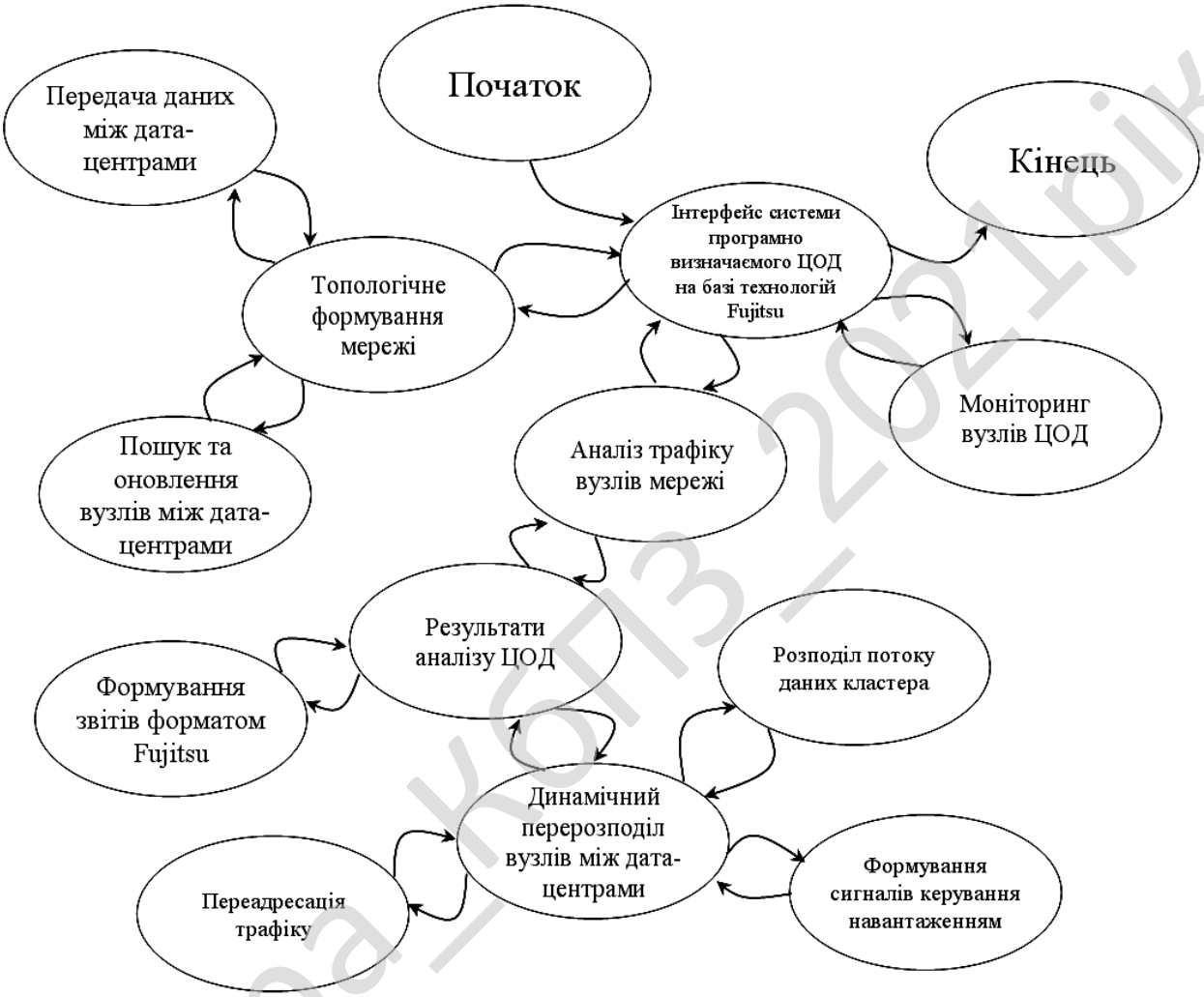

Рисунок 3.3 – Діаграма взаємодії процесів

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

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 створено блок-схеми роботи системи. Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо.

Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції.

Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

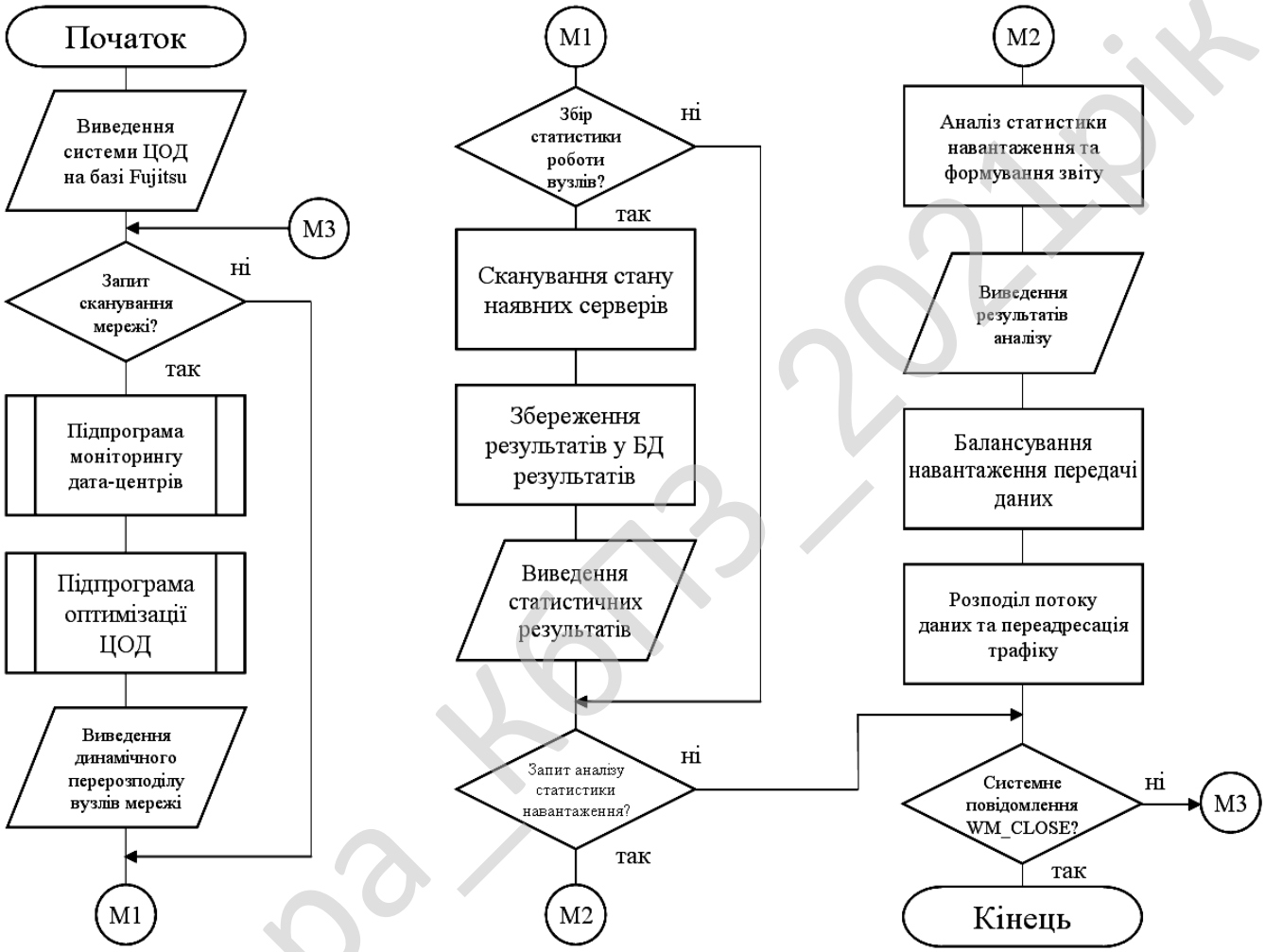

Рисунок 4.1 – Блок-схема основної програми

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

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

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач. Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

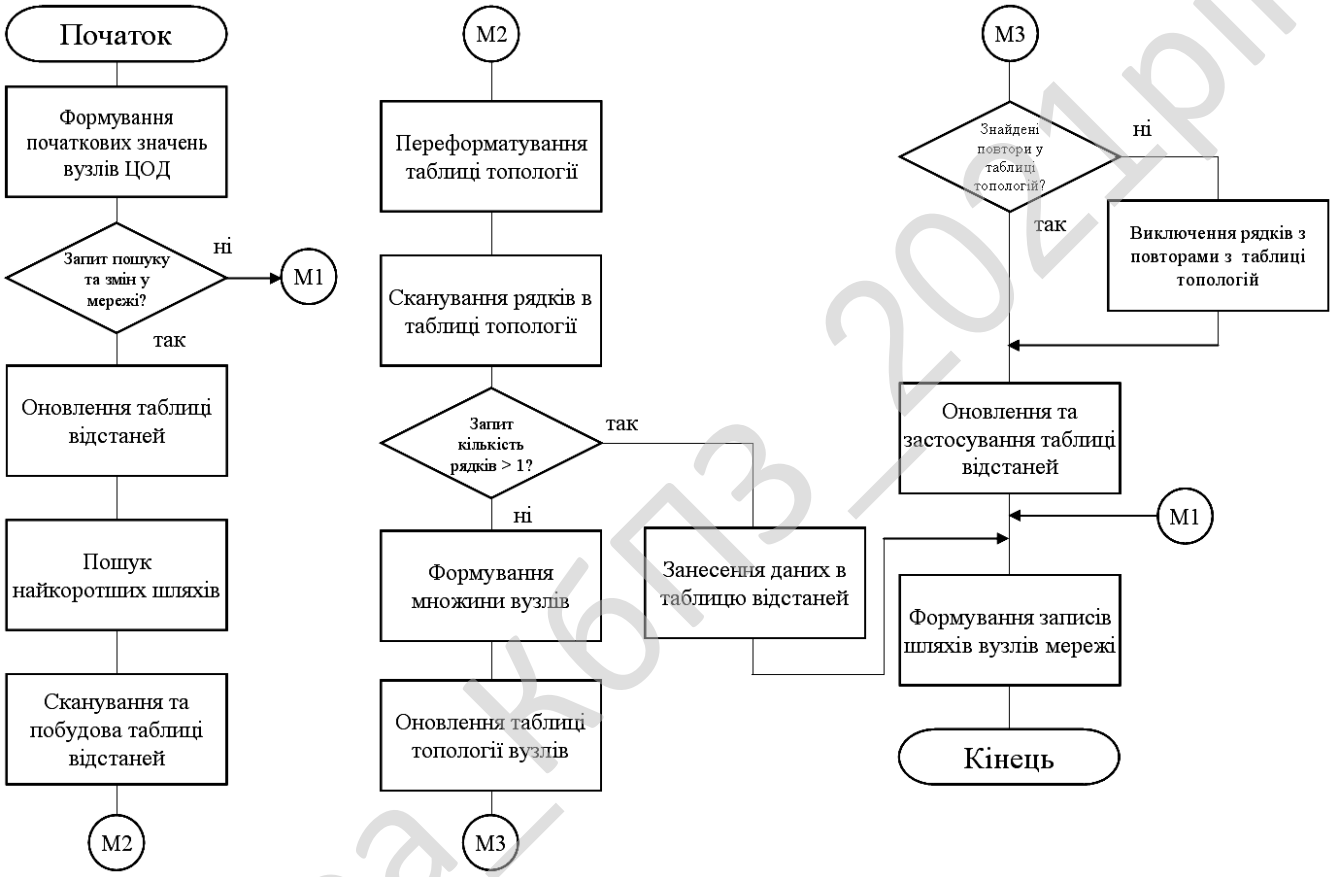

Рисунок 4.2 – Блок-схема роботи підпрограми

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції. Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу.

Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кож ен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується.

Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи. Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи). З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми.

Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

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

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

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись

один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

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

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

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

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

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

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

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

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

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

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

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

Розглянемо визначення API. Це прикладни́й програ́мний інтерфе́йс (Application Programming Interface, API) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

Спрощено – це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій, наприклад для малювання вікна чи іконок на екрані. API є абстрактним поняттям – програмне забезпечення, що пропонує деякий API, часто називають реалізацією (implementation) даного API. У багатьох випадках API є частиною набору розробки програмного забезпечення, водночас, набір розробки може включати як API, так і інші інструменти/ап аратне забезпечення, отже ці два терміни не є взаємозамінювані.

Високорівневі API часто програють у гнучкості. Виконання деяких функцій нижчого рівня стає набагато складнішим, або навіть неможливим.

В об'єктно-орієнтованих мовах, прикладний програмний інтерфейс зазвичай включає в себе опис набору визначень класу, з набором форм поведінки, пов'язаних з цими класами. Це абстрактне поняття пов'язане з реальними функціями, які надані або надаватимуться, класами, які реалізуються в методах класу.

Прикладний програмний інтерфейс в даному випадку можна розглядати як сукупність всіх методів, які публічно доступні в класах (зазвичай званий інтерфейс класу). Це означає, що прикладний програмний інтерфейс вказує методи, за допомогою яких взаємодіє з об'єктами, отриманими з визначень класів і обробляє їх.

У більш загальному плані можна визначити Прикладний Програмний Інтерфейс як сукупність усіх видів об'єктів, які можна вивести з визначення класу, і пов'язаних з ними можливих варіантів поведінки.

Наприклад: клас, що представляє Stack, може просто виставити публічно два методи Push() (для додавання нового елемента в стек) і *Pop()* (для вилучення останнього пункту, ідеально розташований на вершині стека).

У цьому випадку Прикладний Програмний Інтерфейс може бути інтерпретованим як два методи *pop()* і *push()*, або, більш широко, використовується варіант, коли можна використовувати елемент типу Stack, який реалізує поведінку стека, надаючи йому можливість для додавання / видалення елементів з вершини. Друга інтерпретація видається більш доречною в дусі об'єктно-орієнтованого підходу.

Якість документації, пов'язаної з Прикладним Програмним Інтерфейсом, є часто ключовим фактором, що визначає його успішність з точки зору простоти використання.

ППІ, як правило, пов'язаний із бібліотеками програмного забезпечення: ППІ описує і вказує очікувану поведінку в той час, як бібліотека є фактичною

реалізацією даного набору правил. Один ППІ може мати декілька реалізацій (або жодної, будучи абстрактним) у вигляді різних бібліотек, які мають так ий же інтерфейс.

Прикладний програмний інтерфейс також може бути пов'язаним з платформами програмування: платформа може бути заснована на кількох бібліотеках реалізує декілька інтерфейсів ППІ, але на відміну від звичайного використання ППІ, доступ до поведінки вбудований в платформу опосередкований шляхом розширення його змісту новими класами і вставлений в саму платформу. Крім того, загальний потік управління програми може бути під контролем абонента.

Прикладний програмний інтерфейс може бути також реалізацією протоколу.

Коли ППІ реалізує протокол, він може бути заснованим на проксі-методах віддалених викликів, що засновані на протоколі зв'язку. Роль ППІ може заключатися саме в тому, щоб приховати деталі транспортного протоколу. Наприклад: RMI є ППІ, який реалізує протокол або JRMP IIOP як RMI-IIOP.

Протоколи, як правило, розподіляються між різними технологіями і зазвичай дозволяють різним технологіям обмінюватися інформацією, діючи як абстракція між двома світами. ППІ, як правило, є специфічним для конкретної технології: звідси, інтерфейси даної мови не можуть бути використані на інших мовах, якщо виклики функції не будуть перетворені з конкретної адаптації бібліотеки.

Деякі мови, серед яких такі, що працюють на віртуальних машинах (наприклад: мови, сумісні з NET CLI середовища CLR і JVM сумісних мов у віртуальній машині Java) можуть ділитися програмними інтерфейсами.

У цьому випадку віртуальна машина дозволяє мові взаємодії завдяки спільному знаменнику віртуальної машини, що абстрагується від конкретної мови, використовувати проміжний байт-код і його мову.

При використанні прикладного програмного інтерфейсу в контексті веб-розробки, як правило, ППІ визначається набором повідомлень запиту HTTP, також визначається структура повідомлень-відповідей, зазвичай у розширенні мови розмітки XML або в форматі об'єктного запису JavaScript (JSON). У той час як прикладний програмний інтерфейс у Web історично був практично синонімом для веб-служби, останнім часом тенденція змінилась (так званий Web 2.0) на відхід від Simple Object Access Protocol (SOAP) на основі веб-сервісів і сервіс-орієнтованої архітектури (SOA) на більш прямі передачі репрезентативного стану (REST) стилів веб-ресурсів та ресурсів-орієнтованої архітектури (ROA).

Частина цієї тенденції пов'язана з рухом Семантичного веб-ресурсу до Опису Платформ (RDF), Концепції розвитку веб-технологій інженерних онтологій. Прикладні програмні інтерфейси у Web, що дозволяють комбінувати декількома прикладними програмними інтерфейсами в нові додатки називають гібридними.

Розглянемо формат що використовується – **JSON** (JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється джейсон) – це текстовий формат обміну даними між комп'ютерами.

JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). Розробив і популяризував формат Дуглас Крокфорд.

JSON знайшов своє головне призначення у написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти.

JSON з'явився через необхідність обміну даними з сервером у реальному часі без використання плагінів для браузерів, flash-додатків або Java-аплетів, які використовувались скрізь на початку 2000-х років. Дуглас Крокфорд був тим, хто активно просував новий на той час формат. Він з колегами хотів створити технологію, яка використовувала б можливості звичайного браузера давала б веб-розробникам можливість створювати веб-додатки із постійним двостороннім зв'язком із веб-сервером.

JSON вперше був використаний в проекті в Communities.com для Cartoon Network, він дозволяв обмінюватись повідомленнями і одночасно маніпулювати DHTML-елементами.

Веб-сайт JSON.org було запущено 2002 року. У грудні 2005-го року Yahoo! почав переводити деякі зі своїх веб-сервісів на роботу з JSON. Google взялася до роботи з технологією у своєму веб-протоколі GData у грудні 2006-го року.

Використання

За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур.

Якщо говорити про веб-застосунки, в такому ключі він доречний в задачах обміну даними як між браузером і сервером (AJAX), так і між самими серверами (програмні HTTP-інтерфейси).

Формат JSON так само добре підходить для зберігання складних динамічних структур в реляційних базах даних або файловому кеші.

Приклад використання JSON (JavaScript)

```
const ajaxData = '{"name": "wiki",
"fname": "pedia", "rates": [1, 4, 5, 6]}'
const ajaxObj = JSON.parse(ajaxData)
console.log(`${ajaxObj.name}_${ajaxObj.rates[2]}`) /* Виведе "wiki_5"*/
```

Синтаксис

JSON будується на двох структурах:

1. Набір пар назва/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом.

2. Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах.

У JSON використовуються такі їхні форми:

1. Об'єкт – це послідовність пар назва/значення. Об'єкт починається з символу { і закінчується символом } . Кожне значення слідує за : і пари назва/значення відділяються комами.

2. Масив – це послідовність значень. Масив починається символом [і закінчується символом]. Значення відділяються комами.

3. Значення може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну.

4. Рядок – це послідовність з нуля або більше символів юнікода , обмежена подвійними лапками, з використанням escape-послідовностей, що починаються зі зворотної косої риски \. Символи представляються простим рядком. Тип Рядок (String) дуже схожий на String в мовах C і Java. Число теж дуже схоже на C - або Java-число, за винятком того, що вісімкові та шістнадцяткові формати не використовуються. Пропуски можуть бути вставлені між будь-якими двома лексемами.

Наступний приклад показує JSON представлення об'єкта, що описує людину. У об'єкті є рядкові поля імені і прізвища, о б'єкт, що описує адресу, і масив, що містить список телефонів.

```
{
   "firstName": "Іван",
   "address": {
       "city": "Київ",
       "postalCode": 21000
   }}
```

Використання JSON в AJAX

Наступний фрагмент коду JavaScript показує, як клієнт може використати XMLHttpRequest для запиту об'єктів в форматі JSON з серверу. (Серверна частина коду пропущена, вона просто повертає на запит URL рядок у JSON форматі).

Треба відзначити, що тут використання XMLHttpRequest не є кросс - браузерним (за деталями звертайтеся на сторінку XMLHttpRequest), і код зазнаватиме незначних модифікацій в різних версіях оглядачів Internet Explorer, Opera, Safari або Mozilla. Використання запиту XMLHttpRequest обмежено правилом одного джерела (same origin policy): URL, що відповідає на запит, має посилатися на той же сайт, що обслуговує сторінку, що ініціювала запит.

Оглядачі можуть також використовувати тег <iframe> для асинхронного запиту JSON-даних в кросс-браузерному варіанті, або використати просте перенаправлення <form action="url_to_cgi_script" target="name_of_hidden_iframe">. Такий підхід був поширений до приходу популярного нині запиту XMLHttpRequest.

Динамічний тег <script> також можна використати для підвантаження JSON-даних. Ця техніка можлива, щоб обійти надто суворе правило одного джерела, але вона не є безпечною. Запит JSONRequest пропонується, як безпечніша альтернатива.

Питання безпеки

Хоча JSON призначений для передачі даних в серіалізованому вигляді, його синтаксис відповідає синтаксису JavaScript і це створює низку проблем безпеки. Часто для обробки даних, отриманих від зовнішнього джерела у форматі JSON, до них застосовується функція eval() без якої -небудь попередньої перевірки.

JavaScript eval()

Оскільки JSON представляється синтаксично правильним фрагментом коду JavaScript, природним способом розбору JSON-даних в JavaScript-програмі є використання вбудованої в JavaScript функції eval(), яка призначена для

обчислення JavaScript-виразів. При цьому підході відпадає необхідність у використанні додаткових парсерів.

Техніка використання eval() робить систему вразливою, якщо джерело JSON-даних, що використовуються, не відноситься до надійних. Такими даними може виступати шкідливий JavaScript-код для атак за допомогою ін'єкції коду. За допомогою цієї вразливості можливо здійснювати крадіжку даних, підробку автентифікації. Проте, вразливість можна усунути за рахунок використання додаткових засобів перевірки даних на коректність. Наприклад, до виконання eval() отримані від зовнішнього джерела дані можуть перевірятися за допомогою регулярних виразів. У RFC, що визначає JSON пропонується використовувати такий код для перевірки його відповідності формату JSON

```
const my_JSON_object = !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test(
text.replace(/"(\\.|[^"\\])*"/g, '''))) &&
eval('(' + testedData + ')');
```

Як безпечніша альтернатива eval() була запропонована нова функція parseJSON(), здатна обробляти тільки JSON-дані.

Вбудований JSON

Останні версії веб-браузерів мають вбудовану підтримку JSON і здатні його обробляти без виклику функції eval(), що пр изводить до описаної проблеми. Обробка JSON у такому разі зазвичай здійснюється швидше. Так у червні 2009 року вбудовану підтримку JSON мали такі браузери: Mozilla Firefox 3.5+; SeaMonkey 2; Thunderbird 3; Microsoft Internet Explorer 8; Opera 10.5; Браузери, засновані на WebKit (наприклад, Google Chrome, Apple Safari).

Принаймні дві популярні бібліотеки JavaScript використовують вбудований JSON у разі його доступності: jQuery; Dojo.

Підробка крос-доменного запиту

Непродумане використання JSON робить сайти вразливими до підробки міжсайтових запитів (CSRF або XSRF). Оскільки тег <script> допускає використання джерела, що не належить до того ж домену, що і використовуваний ресурс, це дозволяє виконувати код даних, представлених у форматі JSON, в

контексті довільної сторінки, що робить можливою компрометацію паролів або іншої конфіденційної інформації користувачів, що пройшли авторизацію на іншому сайті.

Це є проблемою тільки у разі вмісту в JSON -даних конфіденційної інформації, яка може бути компрометована третьою стороною і якщо сервер розраховує на політику одного джерела, блокуючи доступ до даних при виявленні зовнішнього запиту. Це не є проблемою, якщо сервер визначає допустимість запиту, надаючи дані тільки у разі його коректності. HTTP cookie не можна використовувати для визначення цього. Виключне використання HTTP cookie використовується підробкою міжсайтових запитів.

Розширення, JSONP

JSONP або «JSON з підкладкою» є розширенням JSON, коли назва функції зворотного виклику вказується як вхідний аргумент. Спочатку ідея була запропонована в блозі MacPython в 2005 році, і в наш час використовується багатьма Web 2.0 застосунками, такими, як Dojo Toolkit Applications, Google Toolkit Applications і zanox Web Services.

Подальші розширення цього протоколу були запропоновані з урахуванням введення додаткових аргументів, як, наприклад, у разі JSONPP за підтримки S3DB вебсервісів.

Оскільки JSONP використовує скрипт-теги, виклики по суті відкриті світові. З цієї причини JSONP може бути недоречними для зберігання конфіденційних даних.

Включення скриптових тегів від віддалених сайтів дозволяє їм передати будь-який контент на сайті. Якщо віддалений сайт має вразливості, які дозволяють виконати ін'єкції JavaScript, то початковий сайт також може бути ними зачеплений.

Розширення, BSON

BSON – це бінарна форма представлення простих структур даних і асоціативних масивів (які називають об'єктами або документами). Назва «BSON»

заснована на визначенні JSON і неофіційно означає «Binary JSON» (бінарний JSON).

JSON Reference

Стандарт JSON не описує посилання на інші об'єкти або частини, але існує чернетка стандарту IETF для посилань на об'єкти на основі JSON. Посилання дозволяють здійснювати трансклюзію – вставляти одні документи в інші.

JSON Reference – це JSON-об'єкт з ключем "$ref" (всі інші ключі ігноруються) і значенням стрічкового типу що містить URI, наприклад:

```
{ "$ref": "http://example.com/example.json#/foo/bar" }
```

Якщо URI містить ідентифікатор фрагмента (в прикладі вище "/foo/bar"), він інтерпретується як JSON Pointer.

Модуль dojox.json.ref в Dojo toolkit, забезпечує підтримку декількох форм JSON Reference.

Систе́ма керування ба́зами да́них (СКБД, Database Management System, DBMS) – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

Першим поколінням СКБД прийнято вважати ієрархічні й мережеві системи. Ці системи отримали широке поширення в 1970-х роках, а першою комерційною системою цього типу була система IMS компанії IBM.

У 1980-х роках ці системи були витіснені системами другого покоління – повсюдно використовуваними і донині реляційними СКБД. У цих системах використовувалися непроцедурні мови управління даними (SQL) і передбачався значний ступінь незалежності даних. Реляційні системи внесли значні удосконалення в управління даними: графічний користувацький інтерфейс (GUI), клієнт-серверні застосунки, розподілені бази даних, паралельний пошук даних та інтелектуальний аналіз даних.

Але вже до кінця 1980-х років існуюча тоді реляційна модель перестала задовольняти розробників в силу низки обмежень. Відповіддю на зростаючу

складність програм баз даних стали два нових напрямки розвитку СКБД: об'єктно-орієнтовані СКБД і об'єктно-реляційні СКБД.

У 1991 був утворений консорціум ODMG, основною метою якого стало вироблення промислового стандарту об'єктно-орієнтованих баз даних. Між 1993 та 2001 роками ODMG опублікувала п'ять ревізій своїх специфікацій. Остання версія стандарту має індекс 3.0, після чого група розпустилася. До кінця 1990-х існувало близько десяти компаній, що виробляли комерційні продукти, що позиціонуються на ринку як ООСКБД. Найбільш відомими системами даного класу стали Objectivity, Versant виробництва однойменних компаній, а також СКБД Jasmine, випущена компанією CA. Незважаючи на переваги, що дозволяють ефективніше вирішувати певний ряд завдань, об'єктно-орієнтовані системи так і не змогли завоювати значущу частку ринку СКБД, залишившись «нішевим» продуктом.

Постачальниками традиційних реляційних СКБД також була проведена значна робота з об'єднання об'єктно-орієнтованих і реляційних систем. Розробники постаралися розширити мову SQL, щоб включити в неї концепції об'єктно-орієнтованого підходу, зберігаючи переваги реляційної моделі (об'єктні розширення мови SQL були зафіксовані в стандарті SQL:1999).

Основний принцип – це еволюційний розвиток можливостей СКБД без поломки попередніх підходів та зі збереженням наступності з системами попереднього покоління.

Поняття СКБД третього покоління, якими, власне кажучи, і є об'єктно-реляційні СКБД, з'явилося після опублікування групою відомих фахівців в області баз даних «Маніфесту систем баз даних третього покоління». Основні принципи СКБД третього покоління, позначені в маніфесті:

Крім традиційних послуг з управління даними, СКБД третього покоління повинні забезпечити підтримку розвиненіших структур об'єктів і правил. Розвинутіша структура об'єктів характеризує засоби, необхідні для зберігання і

маніпулювання нетрадиційними елементами даних (тексти, просторові дані, мультимедіа).

СКБД третього покоління повинні включити в себе СКБД другого покоління. Системи другого покоління внесли вирішальний вклад у двох областях – непроцедурний доступ за допомогою мови запитів SQL і незалежність даних. Ці досягнення обов'язково повинні враховуватися в системах третього покоління.

СКБД третього покоління повинні бути відкриті для інших підсистем. Це включає оснащення різноманітними інструментами підтримки прийняття рішень, доступом з багатьох мов програмування, інтерфейсами до існуючих популярних систем і бізнес-застосунків, можливістю запуску програм з бази даних на іншій машині і розподілені СКБД. Весь набір інструментів і СКБД має ефективно функціонувати на різноманітних апаратних платформах з різними операційними системами.

Крім того, СКБД, що розраховує на широку сферу застосування, повинна бути оснащена мовою четвертого покоління (4GL).

У середині 1990 років було лише кілька дослідних прототипів СКБД, які поєднали найкращі риси реляційних і об'єктно-орієнтованих СКБД. Першим комерційним продуктом, якому були властиві об'єктно-реляційні риси, став Universal Server компанії Informix(згодом була поглинена IBM). В даний час більшість цих ідей вже втілено в реальних комерційних рішеннях, в тому числі і в продуктах основних постачальників СКБД (Oracle Database і IBM DB2).

Розвиток індустрії систем керування базами даних базується на значних фундаментальних наукових дослідженнях. Найчастіше, між самими дослідженнями та їхньою конкретною реалізацією в прикладних рішеннях минають роки, а іноді й десятиліття. Роботу в області управління даними проводять як університетські дослідницькі групи (MIT, Berkeley), так і центри розробок основних постачальників СКБД (Oracle, IBM, Microsoft). Інвестування в управління даними – це довгострокове, і разом з тим, вигідне вкладення коштів.

В даний час дослідники мають у своєму розпорядженні засоби, що дозволяють ефективно реалізувати найскладніші запити, що маніпулюють терабайтами й петабайтами різних даних.

Основними тенденціями, які дали привід для проведення різних масштабних досліджень в області баз даних стали:

Експонентний ріст даних. Обсяг даних, у тому числі синтетичних, що генеруються автоматизованими системами, значно зріс. Збільшилося і число прикладних областей, в яких вимагається обробка великих обсягів даних. До таких областей тепер відносяться не тільки традиційні корпоративні програми, пошук у веб, але також і наукові дослідження, обробка природних мов, аналіз соціальних мереж тощо.

Значне ускладнення структур використовуваних даних. Прості види даних у вигляді чисел і символьних рядків стали доповняться численною мультимедійною інформацією, просторовими, процедурними даними та великою кількістю інших складних форматів.

Широке поширення дешевих високопродуктивних апаратних засобів. Щорічно ми спостерігаємо зростання обчислювальних можливостей мікропроцесорів, збільшення ємності і зниження вартості доступних і зручних в експлуатації пристроїв дискової оперативної пам'яті.

Активний розвиток засобів комунікації та «всесвітньої павутини» World Wide Web. WWW стає єдиним інформаційним середовищем, що пронизує весь світ і об'єднує величезне число користувачів та електронних пристроїв.

Поява нових важливих областей застосування СКБД. У першу чергу, це пов'язано з інтелектуальним аналізом даних, сховищами даних, а останнім часом – з паралельними обчисленнями і хмарними технологіями.

Основні характеристики СКБД

1. Контроль за надлишковістю даних.
2. Несуперечливість даних.
3. Підтримка цілісності бази даних (коректність та несуперечливість).

4. Цілісність описується за допомогою обмежень.

5. Незалежність прикладних програм від даних.

6. Спільне використання даних.

7. Підвищений рівень безпеки.

Можливості СКБД

1. Дозволяється створювати БД (здійснюється за допомогою мови визначення даних DDL (Data Definition Language)).

2. Дозволяється додавання, оновлення, видалення та читання інформації з БД (за допомогою мови маніпулювання даними DML, яку часто називають мовою запитів)

3. Можна надавати контрольований доступ до БД за допомогою: Системи забезпечення захисту, яка запобігає несанкціонованому доступу до БД; Системи керування паралельною роботою прикладних програм, яка контролює процеси спільного доступу до БД; Система відновлення – дозволяє відновлювати БД до попереднього несуперечливого стану, що був порушений в результаті збою апаратного або програмного забезпечення.

Основні компоненти середовища СКБД

1. Апаратне забезпечення.

2. Програмне забезпечення.

3. Дані.

4. Процедури – інструкції та правила, які повинні враховуватись при проектуванні та використанні БД.

5. Користувачі: адміністратори даних (керування даними, проектування БД, розробка алгоритмів, процедур) та БД (фізичне проектування, відповідальність за безпеку та цілісність даних); розробники БД; прикладні програмісти; кінцеві користувачі.

Архітектура СКБД

Існує трирівнева система організації СКБД ANSI-SPARC, при якій існує незалежний рівень для ізоляції програми від особливостей представлення даних на нижчому рівні.

Рівні:

1. Зовнішній – представлення БД з точки зору користувача.

2. Концептуальний – узагальнене представлення БД, описує які дані зберігаються в БД і зв'язки між ними. Підтримує зовнішні представлення, підтримується внутрішнім рівнем.

3. Внутрішній – фізичне представлення БД в комп'ютері.

Логічна незалежність – повна захищеність зовнішніх моделей від змін, що вносяться в концептуальну модель.

Фізична незалежність – захищеність концептуальної моделі від змін, які вносяться у внутрішню модель.

MySQL – вільна система керування реляційними базами даних. Розробка та підтримка сайта MySQL здійснює корпорація Oracle, яка отримала права на торговельну марку разом з поглиненої Sun Microsystems, яка раніше придбала шведську компанію MySQL AB.

Продукт поширюється як під GNU General Public License, так і під власною комерційною ліцензією. Крім цього, розробники створюють функціональність за замовленням ліцензійних користувачів. Саме завдяки такому замовленню майже в найраніших версіях з'явився механізм реплікації.

MySQL є рішенням для малих і середніх додатків. Вход ить до складу серверів WAMP, AppServ, LAMP і в портативні збірки серверів Денвер, XAMPP, VertrigoServ. Зазвичай MySQL використовується як сервер, до якого звертаються локальні або видалені клієнти, проте в дистрибутив входить бібліотека внутрішнього сервера, що дозволяє включати MySQL в автономні програми.

Гнучкість СКБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що

підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують транзакції на рівні окремих записів.

Більш того, СКБД MySQL поставляється із спеціальним типом таблиць EXAMPLE, що демонструє принципи створення нових типів таблиць. Завдяки відкритій архітектурі і GPL-ліцензуванню, в СКБД MySQL постійно з'являються нові типи таблиць.

26 лютого 2008 року Sun Microsystems придбала MySQL AB за 1 млрд доларів, 27 січня 2010 року Oracle придбала Sun Microsystems за 7,4 млрд доларів і включила MySQL в свою лінійку СКБД.

Спільнотою розробників MySQL створені різні відгалуження коду, такі як Drizzle (англ.), OurDelta, Percona Server і MariaDB. Всі ці відгалуження вже існували на момент поглинання компанії Sun корпорацією Oracle.

MySQL має подвійне ліцензування. MySQL може розповсюджуватися відповідно до умов ліцензії GPL. Але за умовами GPL, якщо якась програма використовує бібліотеки MySQL, то вона теж повинна розповсюджуватися за ліцензією GPL. Проте це може розходитися з планами розробників, які не бажають відкривати сирцеві тексти своїх програм. Для таких випадків передбачена комерційна ліцензія компанії Oracle, яка також забезпечує якісну сервісну підтримку.

В разі використання та розповсюдження програмного забезпечення з іншими вільними ліцензіями, такими як BSD, Apache, MIT та інші, MySQL дозволяє використання бібліотек MySQL за ліцензією GPL.

MySQL виникла як спроба застосувати mSQL до власних розробок компанії: таблиць, для яких використовувалися ISAM – підпрограми низького рівня для індексного доступу до даних. У результаті був вироблений новий SQL-інтерфейс, але API-інтерфейс залишився в спадок від mSQL. Звідки походить назва «MySQL» – достеменно не відомо. Розробники дають два варіанти: або тому, що практично всі напрацювання компанії починалися з префікса My, або на

честь дівчинки на ім'я Му, дочки Майкла Монті Віденіуса, одного з розробників системи.

Логотип MySQL у вигляді дельфіна носить ім'я «Sakila». Він був обраний з великого списку запропонованих користувачами «імен дельфіна». Ім'я «Sakila» було відправлено Open Source-розробником Ambrose Twebaze.

За час розвитку під орудою Oracle дедалі більше відокремлює MySQL від спільноти і робить процес розробки все менш прозорим. Наприклад, повернута практика поставки власницьких розширених функцій в Enterprise-версії MySQL, спостерігається приховування інформації про вразливості, зі складу виключений тестовий набір, закритий доступ до більшої частини системи відстеження помилок та припинено публікація згрупованого логу змін, що дозволяє судити про прив'язку патчів до конкретних змін.

Можливості

MySQL – компактний багатопотоковий сервер баз даних. Характеризується високою швидкістю, стійкістю і простотою використання.

MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому.

Можливості сервера MySQL:

1. Простота у встановленні та використанні.

2. Підтримується необмежена кількість користувачів, що одночасно працюють із БД.

3. Кількість рядків у таблицях може досягати 50 млн.

4. Висока швидкість виконання команд.

5. Наявність простої і ефективної системи безпеки.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм SHACAL-1, який заснований на перетвореннях алгоритму SHA-1. SHACAL-1 шифрує 160-бітний блок даних з використанням 512-бітного ключа шифрування. Допускається використання більше коротких ключів шифрування (не коротше 128 біт), які п еред виконанням розширення ключа (процедура розширення ключа також успадкована від SHA-1 і буде описана нижче) повинні бути доповнені нульовими бітами для досягнення 512-бітного розміру.

В алгоритмі SHACAL-1 передбачено 80 раундів шифрування. Шифруєме повідомлення представляється у вигляді п'яти 32-бітних субблоків A, B, C, D і E, над якими в кожному раунді виконуються наступні дії:

$$A_{i+1} = K_i + (A_i <<< 5) + f_i(B_i, C_i, D_i) + E_i + M_i,$$

$$B_{i+1} = A_i,$$

$$C_{i+1} = B_i <<< 30,$$

$$D_{i+1} = C_i,$$

$$E_{i+1} = D_i,$$

де i – номер раунду (i = 0...79),

K_i – фрагмент розширеного ключа для i-го раунду,

f_i – функція для i-го раунду (див. нижче),

<<< – операція побітового циклічного зрушення вліво,

M_i – константи, що модифікують, певні в такий спосіб:

Раунди	Значення константи
0...19	5A827999
20...39	6ED9EBA1
40...59	8F1BBCDC
60...79	CA62C1D6

Використовувані в раундах функції f_i визначені так:

Раунди	Функція
0...19	f(x,y,z)=(x&y)|(x'&z)
20...39,60…79	f(x,y,z)=x ⊕ y ⊕ z
40...59	f(x,y,z)=(x ⊕ y)|(x ⊕ z)|(y ⊕ z)

У таблиці символами &, | і ⊕ позначені, відповідно, побітові логічні операції «і», «або» й «або, що виключає» (XOR); x' позначає побітовий комплемент до x.

Шифртекстом є конкатенація вмісту змінних A_{80}, B_{80}, C_{80}, D_{80} і E_{80}.

Процедура розширення ключа в алгоритмі SHACAL-1 також досить проста, вона виконується у два етапи:

Етап 1. 512-бітний вихідний ключ шифрування ділиться на 16 фрагментів по 32 біта $K_0 \ldots K_{15}$...

Етап 2. Інші фрагменти розширеного ключа $K_{16} \ldots K_{79}$ обчислюються з перших 16 фрагментів у такий спосіб:

$$K_i = (K_{i-3} \oplus K_{i-8} \oplus K_{i-14} \oplus K_{i-16}) <<< 1.$$

Раунди розшифрування виконуються у зворотній послідовності; кожний з них має на увазі виконання наступних операцій:

$$A_i = B_{i+1},$$

$$B_i = C_{i+1} <<< 2,$$

$$C_i = D_{i+1},$$

$$D_i = E_{i+1},$$

$$E_i = K'_i + (B_{i+1} <<< 5)' + f_i'(C_{i+1} <<< 2, D_{i+1}, E_{i+1}) + A_{i+1} + M'_i + 4.$$

Тут запис f'(x) позначає побітовий комплемент результату виконання операції f(x).

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у магістерської дипломної роботі система програмно визначаємого ЦОД на базі технологій Fujitsu. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи: Функції представлені у графічному вигляді; Розділу виведення результату роботи системи; Розділу обрання групи; Верхнього меню; Функціональних кнопок ПЗ.

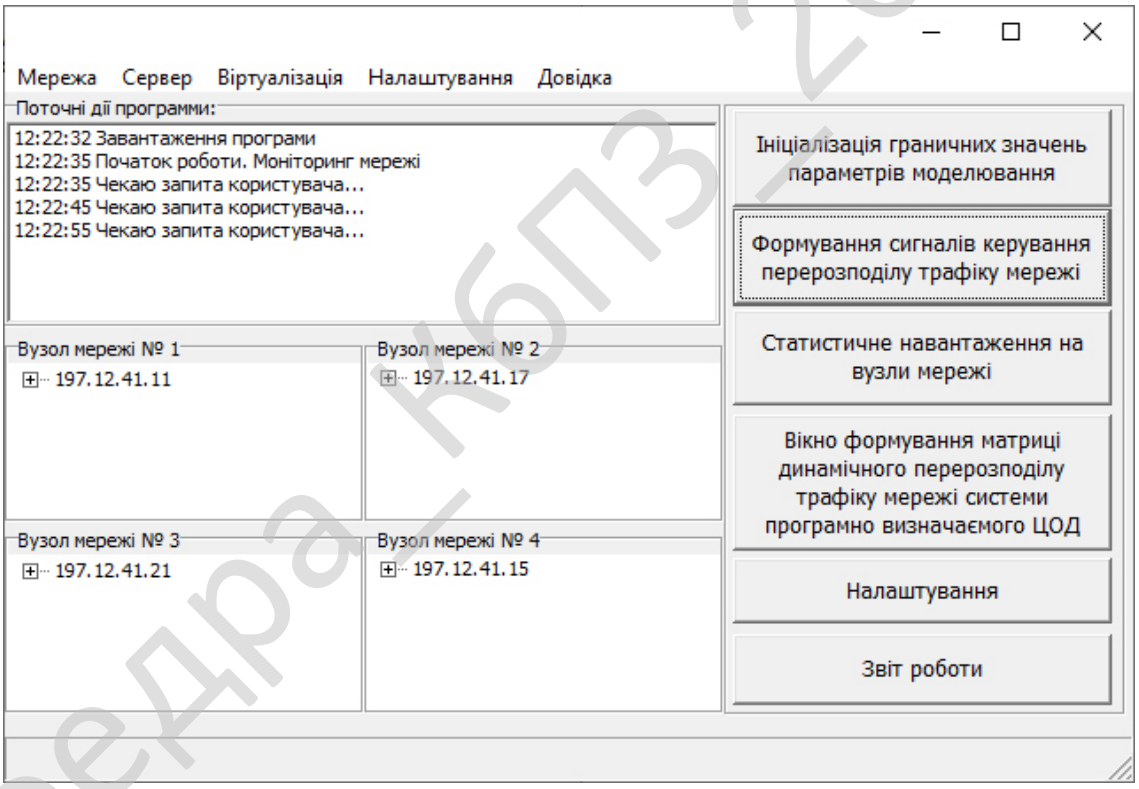

Рисунок 5.1 – Головне вікно ПЗ

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.

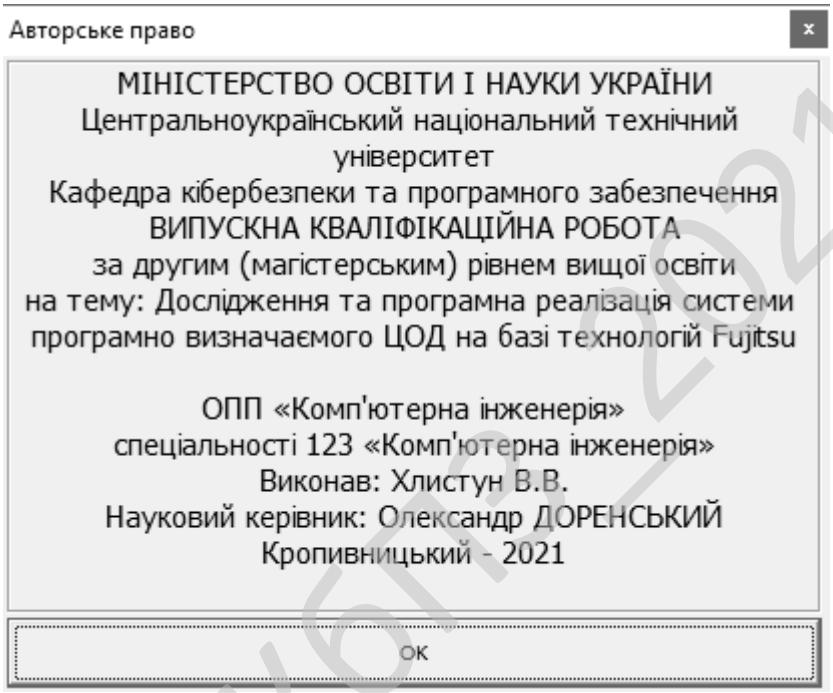

Рисунок 5.2 – Авторське право

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 ІТ рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене ІТ рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

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

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються

в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

Проводилось тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Проводилось тестування чорної скриньки.

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

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

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

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

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

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

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

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

Будь-який спосіб тестування «чорної скриньки» повинен:

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

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

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

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

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

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

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження

комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Метою розробки є дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Об'єктом дослідження є процес програмно визначаємого ЦОД на базі технологій Fujitsu.

Предметом дослідження є методи програмно визначаємого ЦОД на базі технологій Fujitsu.

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

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

– Удосконалено метод програмно визначаємого ЦОД на базі технологій Fujitsu.

– Розроблено вітчизняний продукт програмно визначаємого ЦОД на базі технологій Fujitsu, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці). В магістерській роботі було проведено дослідження та виконана програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu. Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	260 (2 ост. цифри № зал * 10)
3. Запланований термін розробки, днів	Fpq	48 (2 місяць)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	В
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	8
8. Кількість форм вихідної інформації.	–	6
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	1
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	3
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	4
17. Складність кінцевого програмного продукту (1-6)	–	5
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	3
20. Вимоги до швидкодії ПП (1-6)	–	3
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	4
23. Професійний рівень аналітиків (1-6)	–	3
24. Професійний рівень програмістів (1-6)	–	4
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	1
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	3
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	260000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	40
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \qquad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \qquad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+4{,}22+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}2^{1{,}027} = 5{,}5 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \qquad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 5{,}5 \cdot (1 \cdot 1{,}09 \cdot 1{,}30 \cdot 0{,}91 \cdot 1 \cdot 1 \cdot 1 \cdot 1{,}15 \cdot 1 \cdot 0{,}87 \cdot 1{,}10 \cdot$$

$$\cdot 1{,}22 \cdot 1{,}12 \cdot 1{,}10 \cdot 1 \cdot 1 \cdot 1{,}10) = 12{,}9 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \qquad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 12{,}9^{0{,}33+0{,}2(1{,}027-1{,}01)} \cdot 100 = 131 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	15	Д7
Робочий проект	131	Ф 7.1-7.4
Впровадження	15	Д13
Всього	180	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \qquad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{180 \cdot 1}{24 - 3} = 8{,}6 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	385	12	4620	77
Монітор	160	12	1920	32
Клавіатура	140	12	1680	28
Маніпулятор «мишка»	30	12	360	6
Принтер матричний	185	1	185	3
Принтер лазерний	355	2	710	12
Принтер струминний	300	1	300	5
Сканер	155	2	310	5
Концентратор-маршрутизатор	155	2	310	5
Кабельні господарства ЛОМ на 1 м. п.	2,5	70	175	3
Кабельне господарство електромережі	48	50	2400	40
Копіювальний апарат	285	1	285	5
Усього за рік:			$З_ч$	221

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{mic}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{221 \cdot 1}{1,2} = 184,1 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$Ч_{ел}$ = 184,1/(24·8) = 1 ставки.

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2019, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	2	0,5
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	0,5	
	Контроль взаєморозрахунків з постачальниками	0,5	
Всього		4	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	2	0,5
	Створення графічних і стилістичних елементів сайту	1	
	Оформлення банерів і промо-сторінок	0,5	
	Розміщення графіки і контенту на Інтернет сторінках	0,5	
Всього		4	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	17000	34000
Продакт-менеджер	0,5	15000	15000
Інженер-програміст	8,6	16000	275200
Інженер-електронщик	1	15000	30000
Інженер-системотехнік	0,25	15000	7500
Адміністратор мережі	0,5	15000	15000
Системний програміст	0,25	15000	7500
Дизайнер WEB	0,5	16000	16000
Інженер-верстальник	0,25	15000	7500
Бухгалтер-економіст	0,5	16000	16000
Всього за період розробки	$R_{сп}$ = 13,35	-	$Ф_{роб}$ = 423700

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{423700}{13,35 \cdot 48} = 661 \, грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R_{сп}^{1} S_{у} Ц_{пл}, \quad (7.9)$$

де: $R_{сп}^{1}$ – кількість робочих місць виконавців, шт. Приймаємо 12 робочих місць;

$S_{у}$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» (м. Кіровоград, вул. Глинки 16) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 800…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 25 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 12 \cdot 8 \cdot 20000 = 1920000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 192000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R_{сп}^{1} \cdot Ц_{м}, \quad (7.10)$$

де: $Ц_{м}$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 12 \cdot 3500 = 42000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Комп'ютерторг за 25.10.21 – джерело http://computorg.ua/ru/price.html

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11186
Системний блок	Lenovo ThinkCentre M73 USFF	6490
Процесор	Intel Core i5-4570T, LGA1150, (2 (4) ядра п 2.9 - 3.6 GHz); Cache Memory 4 MB	
Системна плата	Lenovo ThinkCentre M73, Intel H81 Express 2x USB 3.0, 3x USB 2.0, 2x Audio Ports, 1 LAN (RJ-45), 1x VGA, 1x DP, UDIMM, PC3 12800 1600 МГц DDR3, два 240-контактни роз'єму DIMM, Gigabit Ethernet, Realte RTL8111GN, бездротова мережа 11b/g/n PCIe Half Mini Card, Intel Centrino® Wireless-N 105	
Відеокарта	Integrated Intel HD Graphics 4600, частот графічної системи 200 MHz - 1.15 GHz	
Жорсткий диск	240 GB SSD	
Оперативна пам’ять	8 GB DDR3	
DVD-привод	Не комплектується	
Корпус	ThinkCentre M73 USFF	
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-E int. 3.5", 1*USB2.0+AUDIO+1394, multi: Al Type Cards, black	

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1 170/160, D-SUB, Wide)	3200
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37 Photo	2970
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійног живлення	UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS)	1496

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	12	11186	13423,2	147655,2
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050

Продовження таблиці 7.7

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Сканери	1	2970	297	3267
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	169473,7

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1920000	-	-
2. Передавальні пристрої	192000	-	-
Всього по групі	2112000	5	105600
Група 4			
3. Обчислювальна техніка	169474	-	-
Всього по групі	169474	50	84737
Нематеріальні активи			
4. Нематеріальні активи	260000	10	26000

Продовження таблиці 7.8

1	2	3	4
Група 5, 6			
5. Вимірювальні пристрої	5190	25	1297,5
6. Транспортні засоби	72500	20	14500
7. Господарський інвентар	42000	25	10500
Всього по групі	119690	-	26297,5
Разом	$К_р$ = 2661164		$А_р$ = 242634,5

Примітка: вартість автомобіля взята по даним з автосалону автотрейдінг, вкладки автобазар, джерело http: //www.auto-trading.com.ua/sale/lot20772.html, складає 2900 USD, що враховуючи прийнятий для розрахунку курс 25 складає 72500 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 661 \cdot 180/260 = 458 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 458 \cdot 10 \cdot 0{,}01 = 46 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$C_{оц} = 0{,}01 \cdot H_c (З_о + З_д), \quad (7.13)$$

де: H_c – відрахування на соціальні потреби, %.

$$C_{оц} = 0{,}01 \cdot 22(458+46) = 111 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $H_г$ = 15% від основної зарплати:

$$Г_{осп} = З_о \cdot H_г \cdot 0{,}01, \quad (7.14)$$

де: $H_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 458 \cdot 15 \cdot 0{,}01 = 69 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам’ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм приймаємо одну пачку паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$ = 120 грн., визначаємо вартість паперу за період розробки $N_м$ = 2 міс:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 120 \cdot 2 = 240 \text{ грн.}$$

Згідно виданих викладачем норм до вартості запам’ятовуючих пристроїв входить вартість CD дисків в кількості, що дорівнює кількості екземплярів програм та одного DVD диска для збереження резервної копії програми:

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 13 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 13 грн./шт.

$$З_{М2} = 260 \cdot 13 + 13 = 3393 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{МЗ} = \sum Ц_{з}.,\qquad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{МЗ} = 574+558+570 = 1702 \text{ грн.}$$

$$З_{М} = (240 + 3393 + 1702)/260 = 21 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01,\qquad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 458 \cdot 15 \cdot 0{,}01 = 69 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 260 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12},\qquad (7.20)$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 242634 \cdot 2/(260 \cdot 12) = 156 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м.\qquad (7.21)$$

$$С_п = 458+46+111+69+21+69+156 = 930 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($Р_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 40%.

$$П_р = 0{,}01 \cdot Р_п \cdot С_п,\qquad (7.22)$$

де: $Р_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 40 \cdot 930 = 379 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1	2	3
1. Основна зарплата виконавців	$З_о$	458
2. Додаткова зарплата виконавців	$З_д$	46
3. Відрахування на соціальні потреби	$С_{оц}$	111
4. Загальногосподарські витрати	$Г_{осп}$	69
5. Витрати на матеріали	$З_М$	21
6. Освоєння нових операційних систем, мов програмування	$О_п$	69
7. Амортизація основних фондів	$А_м$	156
8. Повна собівартість програмного забезпечення	$С_п$	930
9. Плановий прибуток	$П_р$	379
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	1309
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	261,8
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	1570,8

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та

пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	1571
Всього капітальних витрат	–	1571

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування системи	$З_р$	104676	58619
2. Витрати на електроенергію	$З_{ел}$	9900	9240
3. Витрати на амортизацію	$З_{ам}$	0	786
Всього витрат за рік	$І$	114576	68645

Витрати на обслуговування системи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_c), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування системи за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 250 годин на рік до 140 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 250 \cdot 26 \cdot 1{,}1 \cdot 1{,}22 \cdot 12 = 104676\ грн,$$

до:

$$З_{р\ нов} = 140 \cdot 26 \cdot 1{,}1 \cdot 1{,}22 \cdot 12 = 58619\ грн.$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 12 \cdot 0{,}15 \cdot 2500 \cdot 2{,}2 = 9900 \text{ грн.}$$

$$З_{ел\ нов} = 12 \cdot 0{,}14 \cdot 2500 \cdot 2{,}2 = 9240 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	1571	–	785,5
Всього відрахувань	-	–	1571	–	785,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{р_m} \cdot K_{р_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (1309 - 930) \cdot 260 - (0,05 \cdot 2112000 + 0,5 \cdot 169474 +$$
$$+0,25 \cdot 47190 + 0,2 \cdot 72500 + 0,1 \cdot 260000) \cdot 2/12 = 58101 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх стр ок служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{549164}{(1309 - 930) \cdot 260 \cdot 12 / 2} = 0,93 \text{ року}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н (K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно; $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (114576\text{-}68645)\text{-}0,5 \cdot 1571 = 45145,5 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

$$T_{сп} = \frac{1571}{114576 - 68645} < 0,1 \text{ року}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	260
2. Повна собівартість розробленої програми	Грн.	930
3. Ціна розробленої програми	Грн.	1309
4. Плановий прибуток від реалізації розробленої програми	Грн.	379
5. Рентабельність програмної продукції	%	40
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	2661164
7. Загальний прибуток від реалізації програмної продукції	Грн.	98540
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	58101
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,93
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	1571
11.Величина економічного ефекту у користувача програмної продукції	Грн.	45145,5
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,1

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Законом України “Про охорону праці” [1] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та «Правила охорони праці під час експлуатації електронно-обчислювальних машин» НПАОП 0.00-1.28-10,

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення впливу комп'ютера на організм програміста визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста,

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Програміст працює з електронно-обчислювальною машиною (ЕОМ) та іншим обладнанням, яке є джерелом небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення [2]. Так як програміст постійно перебуває в приміщенні, тому для комфортних умов праці в цьому приміщенні необхідно створити належний мікроклімат.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

– ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.

– ризик виникнення пожежі;

– негативний вплив на органи зору людини;

– ризики ураження електричним струмом;

– недостатня, або надмірна освітленість робочого місця;

– монотонність праці;

– електромагнітні (у т.ч. високочастотні) випромінювання (коливання);

– несприятливі мікрокліматичні умови;

– нервово-емоційна напруженість праці;

– інтелектуальні навантаження;

– невідповідність ергономічних показників робочого місця діючим вимогам;

– шуми;

– статичні навантаження на кістково-м’язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	6,4
Довжина	6,8
Висота	2,8

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	7,25
Обсяг, V	$м^3$	не менше 20.0	20,3

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працює 6 осіб. За даними, які наведено у табл. 8.1 та табл. 8.2, можна зробити висновок, що площа та об’єм приміщення у розрахунку на одно робоче місце програміста відповідають нормативним

вимогам (Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, яка виконується в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря у приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23,5	40-57	0,1
Тепла	23-25	50-70	0,1	24-25	52-68	0,11

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

В літню пору застосовується кондиціонер.

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

У приміщенні знаходяться наступні джерела шуму: принтер *HP Laser 135a*, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5 – 28 – 2006 р. можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути

освітлено достатньо рівномірно – ця основна гігієнічна вимога [2]. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

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

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

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень;

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

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов’язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

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

8.5 Розрахункова частина

Початкові данні для розрахунку захисного заземлення:

Тип заземлення: робоче заземлення нульової точки трансформатора. Заземленню підлягає виробниче обладнання організації. Напруга – 220/380 В. Розташування заземлюючих електродів – по контуру.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача методом коефіцієнта використання заземлювачів.

Початкові дані для розрахунку захисного заземлення: тип верхнього і нижнього шару грунта – полутвердий суглинок. Условна товщина верхнього шару грунта: H=1 м. Для захисного заземлення: застосовуються ветикальні електроди – пруток довжиною L = 3 м. Відстань між вертикальними заземлювачами (електродами) A=3 м. Діаметр ветикальньного електрода (прутка) D=12 мм, Тип горизонтального заземлювача: металева полоса. Розміри перетину з'єднуючої полоси: 12 х 4 мм. (b=12 мм.). Опір заземлювача, який нормується: $R_{ЗН}$ = 4 Ом. Глибина закладення контура заземлення t= 0,7 м.

Розрахунок захисного заземлення можна автоматизувати за допомогою програми, сирцевий код якої опублікован на стр.13-16 [3].

Розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,7+3/2=2,2 \text{ м.}$$

Еквівалентний питомий опір грунта [2]:

$$\rho_{екв}= \psi \rho_1 \rho_2 L /[\rho_1 \psi(L-H+t)+ \rho_2 \psi(H-t)]=103,9 \text{ Ом}$$

де

ψ = 1,4 табличне значення коефіцієнта сезонності у багатошаровому грунті;

ρ_1= 200 Ом*м. – питомий опір верхнього шару грунта;

ρ_2= 30 Ом*м. – питомий опір нижнього шару грунта.

Опір розтіканню електричного струму одного електрода вертикального заземлювача:

$$R_O = [\rho_1/(2\pi*L)]*[(ln)2L/D)+0,5ln[(4T+L)/(4T-L)]=36,23 \text{ Ом}.$$

Опір розтіканню електричного струму заземлювача, який нормується, якщо $\rho_{екв}$ >100 (у нас $\rho_{екв}$ =109,59 >100):

$$R = R_{ЗН} * \rho_{екв}/100=4*109,59/100=4,16 \text{ Ом}.$$

Опір розтіканню електричного струму горизонтального заземлювача (полоси):

$$R_П = 0,366(\rho_{екв}\ \psi / L_П \cdot \eta_П)\lg(2/ L_П * L_П/bt)=73,11 \text{ Ом}.$$

де $\eta_П$ = 0,32 (табличне значення коефіцієнта використання горизонтального заземлювача, залежить від співвідношення A/L); довжина горизонтального заземлювача (полоси) при розташуванні заземлювачів по контуру:

$$L_П = A*n =3*16=48 \text{ м.};$$

де n – ітераційна кількість вертикальних заземлювачів [3].

Опір заземлювача розтіканню електричного струму :

$$R_B = R_П R / (R_П - R) =4,23 \text{ Ом}.$$

Кількість вертикальних заземлювачів:

$$n = R_O / R_B * \eta_C =16 \text{ шт}.$$

де η_C = 0,51 (табличне значення коефіцієнта використання вертикального заземлювача, залежить від співвідношення A/L).

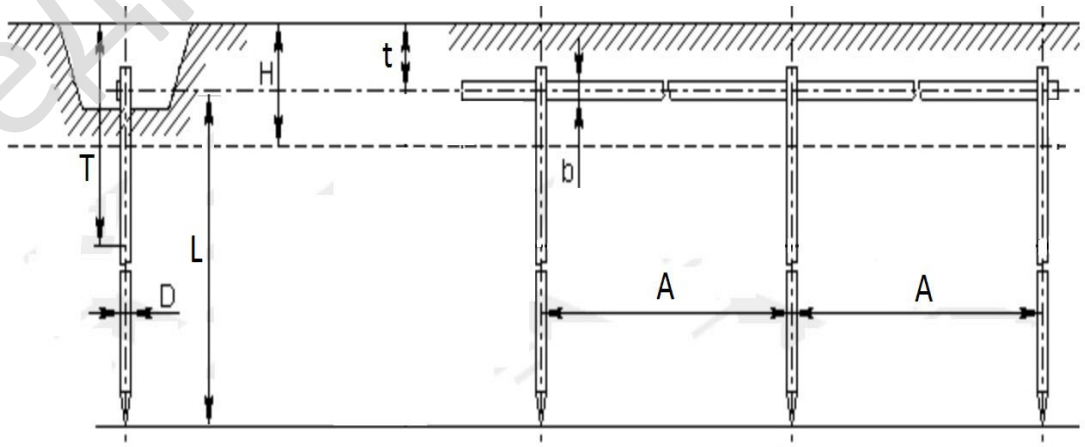

Рисунок 8.1 – Штучний заземлювач

8.6 Висновки до розділу

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

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

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи програмно визначаємого ЦОД на базі технологій Fujitsu.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів програмно визначаємого ЦОД на базі технологій Fujitsu.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем програмно визначаємого ЦОД на базі технологій Fujitsu.

– Досліджена система програмно визначаємого ЦОД на базі технологій Fujitsu.

– На основі отриманих результатів досліджень створена програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання програмно визначаємого ЦОД на базі технологій Fujitsu.

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

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

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

Програма реалізована на мові високого рівня RAD Studio Delphi 10.3.2. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows XP/Vista/7/8/10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм SHACAL-1.

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.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 45145,5 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,1 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хлистун В.В. Дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu // Збірник праць молодих науковців ЦНТУ. – Вип. 12. – Кропивницький: ЦНТУ, 2022.

2. Семенов Ю. А. Телекоммуникационные технологии. Интернет-университет информационных технологий [Электронный ресурс]. – Режим доступа:http://book.itep.ru/

3. Бакланов И. Г. NGN: Принципы построения и реализации / И. Г. Бакланов // – М: Эко-трендз, 2008.

4. Кузнецов А.А. Метод структурной идентификации информационных потоков в телекоммуникационных сетях на основе BDS-тестирования / А.А.Кузнецов, С.Г.Семенов, С.Н.Симоненко, Е.В.Мелешко // Науково-технічний журнал "Наука і техніка Повітряних Сил Збройних Сил України". Випуск 2 (4). – Харків: ХУПС. – 2010. – С. 131 – 137.

5. Кульгин М.Б. Технология и маршрутизация IP/IPX трафика / Максим Кульгин. – М.: Компьютер-пресс, 1998. – 324 с.

6. Кучерявый Е.А. Управление трафиком и качество обслуживания в сети Интернет / Евгений Андреевич Кучерявый. – СПб.: Наука и техника, 2004. – 336 с.

7. Кучук Г.А. Управление ресурсами инфотелекоммуникаций / Г.А. Кучук, Р.П. Гахов, А.А. Пашнев. – М.: Физматлит, 2006. – 220 с.

8. Кучук Г.А Структура управляющей подсети сети передачи данных / Г.А. Кучук, А.А. Болюбаш // Системи обробки інформації. – Х.: ХУПС, 2006. – Вип. 3 (52). – С. 70-74.

9. Лагутин В.С., Степанов С.Н. Телетрафик мультисервесных сетей связи / В.С. Лагутин, С.Н. Степанов. – М.: Радио и связь, 2000. – 320 с.

10. Майника Э. Алгоритмы оптимизации на сетях и графах: пер. с англ. / Э. Майника; под ред. Е.К. Масловского. – М.: Мир, 1981. – 321 с.

11. Мелешко Е.В. Математическая модель подсистемы управления и обслуживания в многопротокольном узле связи / Е.В.Мелешко // Збірник наукових праць Харківського університету повітряних сил.– Х.: ХУПС. – 2010. – Вип. 4 (26). – С. 124-128.

12. Мелешко Е.В. Методы идентификации трафика и динамического управления очередями в многопротокольных узлах связи и оценка их эффективности / Е.В.Мелешко // Системи обробки інформації. – Х.: ХУПС. – 2010. – Вип. 8 (89). – С. 68-74.

13. Мелешко Е.В. Усовершенствование математической модели подсистемы управления трафиком в узле телекоммуникационной сети / Е.В.Мелешко // Збірник тез та доповідей третьої науково -технічної конференції студентів та аспірантів «Захист інформації з обмеженим доступом та автоматизація її обробки (RISAP-2011)». Тези доповідей. Київ: НАУ, 2011. – 6 с.

14. Назаров А.Н. АТМ: Технология высокоскоростных сетей / А.Н. Назаров, М.В. Симонов. – М.: Эко-Трендз, 1997. – 232 с.

15. Назаров А.Н. Модели и методы расчета структурно-сетевых параметров АТМ сетей / Алексей Николаевич Назаров. – М.: Горячая линия – Телеком, 2002. – 256 с.

16. Новиков Ю.В. Локальные сети: архитектура, алгоритмы, проектирование / Ю.В. Новиков, С.В. Кондратенко – М.: ЭКОМ, 2000. – 312 с.

17. Одом Ш. Коммутаторы CISCO / Ш. Одом, Х. Ноттингем – М.:"Кудиц-Образ", 2003. – 528 с.

18. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов / В.Г. Олифер, Н.А. Олифер. – 2-е изд. – СПб.: Питер, 2007. – 958 с.

19. Пантелеев А.В. Т.А. Методы оптимизации в примерах и задачах / А.В. Пантелеев, Т.А. Летова. – М.: Высшая школа, 2002. – 544 с.

20. Руководство по технологиям объединенных сетей. 4-е изд. / пер.с англ. и ред. А.Н. Крикуна – М.: Изд. дом «Вильямс», 2005. – 1040 с.

21. Свами М.Н., Тхуласираман К. Графы, сети и алгоритмы: пер. с англ. / М.Н. Свами, К. Тхуласираман; под ред. В.А. Горбатова. – М.: Мир, 1984. – 454 с.

22. Семенов А.Б. Структурированные кабельные системы / А.Б. Семенов, С.К. Стрижаков, И.Р. Сунчелей – 3-е изд. – М.: "Компьютер-Пресс", 2001. – 608с.

23. Семенов А. Д. Идентификация объектов управления: Учебн. Пособие / А.Д.Семенов, Д.В.Артамонов, А.В.Брюхачев – Пенза: Изд-во Пенз. гос. ун-та, 2003.- 211 с.

24. Семенов С.Г. Анализ методов прогнозирования в телекоммуникационных сетях автоматизированных систем управления / С.Г.Семенов // Збірник наукових праць «Системи управління, навігації та зв'язку», – К.:ЦНДІ навігації і управління, – 2008.-Вип. 2(6) .- С.134-137

25. Семенов С.Г. Математическая модель процесса доставки информационных пакетов в компьютерной сети системы критического применения / С.Г.Семенов, И.В.Ильина // Науково -технічний журнал «Радіоелектронні і комп'ютерні системи» Х.:ХАІ, – 2008.-Вип. 1(28) – С.162-165

26. Семенов С.Г. Оптимизация трафика на основе сбалансированной загрузки информационно-телекоммуникационной сети // Системи обробки інформації. – Х.: ХВУ, 2004. – № 8(36). – С.206-210

27. Семенов С.Г. Сравнительные исследования методов идентификации трафика в телекоммуникационной сети для повышения оперативности передачи данных / С.Г.Семенов, Е.В.Мелешко // Науково-технічний журнал

28. "Прикладная радиоэлектроника" Том 9 №3. – Х: ХНУРЕ. – 2010. – С.444-448.

29. Семенов С.Г. Сравнительные исследования и анализ алгоритмов управления очередями в многопротокольных узлах связи телекоммуникационной сети / С.Г.Семенов, Е.В.Мелешко, В.В.Босько // «Новітні технології – для захисту повітряного простору» Матеріали шостої наукової конференції 14-15 квітня. – Х:

ХУПС, 2010.– С.132.

30. Semenov S. The method of processing and identification of telecommunication traffic based on BDS-tests / S. Semeno v, A.Smirnov., E.Meleshko // The book of materials International Conference «Statistical Methods of Signal and Data Processing (SMSDP-2010)» – Kiev, Ukraine, National Aviation University "NAU-Druk" Publishing House, October 13-14, 2010. – С.166-168. – engl.

31. Семенов Ю.А. Сети Інтернет. Архитектура и протоколы / Ю.А. Семенов. – М.: Блик плюс, 1998. – 424 с.

32. Смирнов А.А. Разработка методики оценки среднего времени обслуживания информационных пакетов в телекоммуникационной сети / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи управління, навігації та зв'язку. – Київ: ДП «Центральний науково-дослідний інститут навігації і управління», 2009. – Вип. 2(10). – С.162-165.

33. Смирнов А.А. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

34. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011. – 193-195 с.

35. Современные телекоммуникации. Технологии и экономика / [В.Л. Банкет, О.В. Бондаренко, П.П. Воробьенко и др.]; под ред. С.А. Довгого. – М.: Эко-Трендз, 2003. – 320 с.

36. Столлингс В. Современные компьютерные сети / Вильям Столлингс.– СПб.: Питер, 2003. – 778 с.

37. Сэломон Д. Сжатие данных, изображений и звука / Д. Сэломон.– М.: Техносфера, 2004. – 368 с.

38. Таненбаум Э. Компьютерные сети / Эндрю Таненбаум; пер. с англ. А. Леонтьев. – СПб.: Питер, 2002. – 848 с.

39. Телекоммуникационные системы и сети: учебное пособие. В 3 томах / [В.В. Величко, Е.А. Субботин, В.П. Шувалов, А.Ф. Ярославцев]; под ред. В.П. Шувалова. – М.: Горячая линия-Телеком, 2005, т. 3 – 592 с.

40. Уолрэнд Дж. Телекоммуникационные и компьютерные сети / Дж. Уолрэнд. – М.: Постмаркет, 2001. – 480 с.

41. Хайкин С. Нейронные сети: полный курс / С. Хайкин. – М.:Вильямс, 2006. – 1103 с.

42. Хаусли Т. Системы передачи и телеобработки данных: пер. с англ. / Т. Хаусли; под ред. Ю.М. Мартынова. – М.: Радио и связь, 1994. – 452 с.

43. Чернявский Г.М. Новые технологии в спутниковых системах / Г.М. Чернявский // Информационные технологии и вычислительные системы. – М.: Институт микропроцессорных вычислительных систем РАН, 2005. – Вып.1 – С. 3 – 11.

44. Шелевицький І.В. Методи та засоби сплайн-технології обробки сигналів складної форм / І.В. Шелевицький– Кривий Ріг: Європейський університет, 2002. – 304 с.

45. Шелухин О.И. Фрактальные процессы в телекоммуникациях: моногр. / О.И. Шелухин, А.М. Тенякшев, А.В. Осин – М.: Радиотехника, 2003. – 480 с.

46. Олифер В. Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов: 3-е изд./ В. Г Олифер., Н. А. Олифер // – СПб.: Питер, 2006.

47. Олифер В. Г. Искусство оптимизации трафика / В. Г. Олифер, Н. А Олифер // Журнал сетевых решений LAN. – № 12.– 2001.

48. Зайченко Ю. П. Анализ и оптимизация характеристик сетей MPLS по заданным показателям качества / Ю. П. Зайченко, Ахмед А. М. Шарадка // Вісник

національного технічного університету України КПІ сер. Інформатика управління та обчислювальна техніка. Вип. 43, с. 113–123

49. Будылдина Н. В. Разработка программного обеспечения для оптимизации мультисервісных сетей / Н. В. Будылдина., П. А. Коновалов // Открытое образование, июнь 2006.

50. Зайцев Д. А. Моделирование телекоммуникационных сетей в системе NS. / Д. А. Зайцев, Т. Н. Шинкарчук // Наукові праці ОНАЗ ім. О. С. Попова. – 2006.– № 2.

51. Кучерявый Е.А. Управление трафиком и качество обслуживания в сети Интернет / Е.А. Кучерявый // – М.: Наука и Техника. – 2007. – 336 с.

52. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

53. Про охорону праці: Закон України від 14.10.1992 р. № 2694-XII. Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12#Text

54. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

55. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. – Кіровоград: КІСМ, 1997. – 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи програмно визначаємого ЦОД на базі технологій Fujitsu.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 42-13 від 02.08.2021 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи програмно визначаємого ЦОД на базі технологій Fujitsu.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи програмно визначаємого ЦОД на базі технологій Fujitsu;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище RAD Studio Delphi 10.3.2.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2021 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 124 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2021 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 20.12.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

другим (магістерським) рівнем вищої освіти

_____ Доренський О.П.

Дослідження та програмна реалізація

системи програмно визначаємого ЦОД на базі технологій Fujitsu

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2021 року

Файл Networks.pas – робота з мережею програмно визначаємого ЦОД на базі технологій Fujitsu

```
unit Networks;

interface

uses Windows, ShellAPI, ShlObj, ActiveX, ComObj, Dim, Classes, SysUtils,
WinSock;

type
  ComputerFound = class (Exception);
  ECannotFindNetwork = class (Exception);

  TStringObject = class (TObject)
  private
    FValue: TString;
    FTag: Integer;
    FData: Pointer;
    FRefObj: TObject;
    procedure SetValue(const Value: TString);
    procedure SetData(const Value: Pointer);
    procedure SetRefObj(const Value: TObject);
    procedure SetTag(const Value: Integer);
  public
    property Value: TString read FValue write SetValue;
    property RefObj: TObject read FRefObj write SetRefObj;
    property Tag: Integer read FTag write SetTag;
    property Data: Pointer read FData write SetData;
  end;

  TStringObjectArray = class (TDynamicArray)
  private
    function GetObject(Index: Integer): TStringObject;
    function GetData(Index: Integer): Pointer;
    function GetRefObj(Index: Integer): TObject;
    function GetTag(Index: Integer): Integer;
    function GetValue(Index: Integer): TString;
    procedure SetData(Index: Integer; const Value: Pointer);
    procedure SetRefObj(Index: Integer; const Value: TObject);
    procedure SetTag(Index: Integer; const Value: Integer);
    procedure SetValue(Index: Integer; const Value: TString);

    function FreeObject(Index: Integer; var Obj: TStringObject): Integer;

    procedure FreeItem(Index: Integer);
    procedure CreateItem(Index: Integer);

  protected
    procedure SetCount(const NewCount: Cardinal); override;
  public
    function Add: Integer; override;
    procedure Insert(Index: Integer); override;
    procedure Delete(Index: Integer); override;
    function AddItem(const Item): Integer; override;
    procedure InsertItem(Index: Integer; const Item); override;
    procedure DeleteItem(Index: Integer; out Item); override;
    property Value[Index: Integer]: TString read GetValue write SetValue;
default;
    property RefObj[Index: Integer]: TObject read GetRefObj write SetRefObj;
    property Tag[Index: Integer]: Integer read GetTag write SetTag;
    property Data[Index: Integer]: Pointer read GetData write SetData;
    constructor Create;
    destructor Destroy; override;
  end;
```

```
  TStringObjectList = class (TStrings)
  private
    FArray: TStringObjectArray;
    function GetData(Index: Integer): Pointer;
    function GetTag(Index: Integer): Integer;
    procedure SetData(Index: Integer; const Value: Pointer);
    procedure SetTag(Index: Integer; const Value: Integer);
  protected
    function Get(Index: Integer): string; override;
    function GetCount: Integer; override;
    function GetObject(Index: Integer): TObject; override;
    procedure Put(Index: Integer; const S: string); override;
    procedure PutObject(Index: Integer; AObject: TObject); override;

  public
    property Data[Index: Integer]: Pointer read GetData write SetData;
    property Tag[Index: Integer]: Integer read GetTag write SetTag;

    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Exchange(Index1, Index2: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;


    constructor Create;
    destructor Destroy; override;
  end;

  {TNetworkWorkgroup - клас, який створює список всіх комп' ютерів в робочій
групі. Цей клас є нащадком класу TStrings і повністю сумісний з іншим нащадками
цього класу.  Об' єкти цього класу записуються у властивості об' єктів класу
TNetworkNeiborhood. Клас TNetworkNeiborhood містить об' єкти і властивості
робочих груп. }

  TNetworkWorkgroup = class (TStringObjectList);


  {TNetworkNeiborhood - клас, який створює список всіх робочих груп в мережі}
  TNetworkNeighborhood = class (TStringObjectList)
  private
    function CreatePIDL(Size: Integer): PItemIDList;
    procedure DisposePIDL(ID: PItemIDList);
    function NextPIDL(IDList: PItemIDList): PItemIDList;
    function GetPIDLSize(IDList: PItemIDList): Integer;
    function CopyPIDL(IDList: PItemIDList): PItemIDList;
    procedure StripLastID(IDList: PItemIDList);
    function GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
    class function GetDisplayName(ShellFolder: IShellFolder; PIDL: PItemIDList):
TString;
    function OriginFolder:  IShellFolder;
    function OriginFolderNT: IShellFolder;
    class function EnumObjects(ShellFolder: IShellFolder): IEnumIDList;
    class procedure ParseFolder(Folder: IShellFolder; Items: TStringObjectList;
 StorePIDLs: Boolean = False);
    class procedure ParseFolderEx(Folder: IShellFolder; Items: TStrings);

    function FreeRefObj(Index: Integer; var Obj: TStringObject): Integer;
    function GetWorkgroup(Name: TString): TNetworkWorkgroup;

  public
    { Процедура Оновлення шукає всі доступні робочі групи в мережі }

    procedure Refresh;

    { містить списки всіх комп' ютерів в  мережі}

    property Workgroup[Name: TString]: TNetworkWorkgroup read GetWorkgroup;
```

```
   { Функція FindComputer шукає комп’ ютер зі вказаним ім’ ям і повертає ім’ я
робочої групи де його знайдено, або порожню строку
     якщо комп’ ютера з таким іменем у мережі немає }

    function FindComputer(Name: TString): TString;

    { Процедура ListComputers копіює список всіх комп’ ютерів в мережі
      у об’ єкті TStrings}
    procedure ListComputers(Strings: TStrings);

    { Процедура ListNetwork копіює відсортований в алфавітному порядку список
всіх робочих груп і комп’ ютерів в мережі.
     Робочі групи мають ‘ TObject(1)’  у відповідному елементі властивості об’
єктів, а комп’ ютери - ‘ nil’ }

    procedure ListNetwork(Strings: TStrings);

    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;
    constructor Create;
  end;

{ Функція GetIPAddress отримує IP адресу комп’ ютера або сервера.
  Параметр NetworkName конкретизує ім’ я комп’ ютера або  сервера.
  Ця функція повертає адреси IP у форматі XXX.XXX.XXX.XXX у разі успішного
виконання, ‘ Error’  - коли неможливо ініціалізувати, ‘ Unknown’  – коли
параметр NetworkName посилається на неіснуючий комп’ ютер або на комп’ ютер  без
встановленого протоколу TCP/IP }

  function GetIPAddress(NetworkName: TString): TString;


{ GetIPAddresses отримує IP адреси всіх доступних комп’ ютерів мережі }

procedure GetIPAddresses(Network: TNetworkNeighborhood; List: TStrings);



{ Функція EnumSharedResources перераховує загальні ресурси мережі. Параметр
ComputerName конкретизує
  ім’ я комп’ ютера. }

function EnumSharedResources(ComputerName: TString; List: TStrings): Boolean;


implementation

uses NetConst;

{ TStringObject }

procedure TStringObject.SetData(const Value: Pointer);
begin
  FData := Value;
end;

procedure TStringObject.SetRefObj(const Value: TObject);
begin
  FRefObj := Value;
end;

procedure TStringObject.SetTag(const Value: Integer);
begin
  FTag := Value;
end;

procedure TStringObject.SetValue(const Value: TString);
```

```
begin
  FValue := Value;
end;

{ TStringObjectArray }

function TStringObjectArray.Add: Integer;
begin
 Result:=inherited Add;
 CreateItem(Result);
end;

function TStringObjectArray.AddItem(const Item): Integer;
begin
 Result:=Add;
end;

constructor TStringObjectArray.Create;
begin
 inherited Create(0, SizeOf(TStringObject));
end;

procedure TStringObjectArray.CreateItem(Index: Integer);
var
 P: ^TStringObject;
begin
 P:=GetItemPtr(Index);
 P^:=TStringObject.Create;
end;

procedure TStringObjectArray.Delete(Index: Integer);
begin
 FreeItem(Index);
 inherited;
end;

procedure TStringObjectArray.DeleteItem(Index: Integer; out Item);
begin
 Delete(Index);
end;

destructor TStringObjectArray.Destroy;
begin
 ForEach(Integer(Self), @TStringObjectArray.FreeObject);
 inherited;
end;

procedure TStringObjectArray.FreeItem(Index: Integer);
var
 P: ^TStringObject;
begin
 P:=GetItemPtr(Index);
 FreeAndNil(P^);
end;

function TStringObjectArray.FreeObject(Index: Integer;
  var Obj: TStringObject): Integer;
begin
 FreeAndNil(Obj);
 Result:=0;
end;

function TStringObjectArray.GetData(Index: Integer): Pointer;
begin
 Result:=GetObject(Index).Data;
end;

function TStringObjectArray.GetObject(Index: Integer): TStringObject;
begin
```

```
 GetItem(Index, Result);
end;

function TStringObjectArray.GetRefObj(Index: Integer): TObject;
begin
 Result:=GetObject(Index).RefObj;
end;

function TStringObjectArray.GetTag(Index: Integer): Integer;
begin
 Result:=GetObject(Index).Tag;
end;

function TStringObjectArray.GetValue(Index: Integer): TString;
begin
 Result:=GetObject(Index).Value;
end;

procedure TStringObjectArray.Insert(Index: Integer);
begin
 inherited;
 CreateItem(Index);
end;

procedure TStringObjectArray.InsertItem(Index: Integer; const Item);
begin
 Insert(Index);
end;

procedure TStringObjectArray.SetCount(const NewCount: Cardinal);
var
 i, OldCount: Integer;
begin
 OldCount:=Count;
 if NewCount > Count then begin
  inherited SetCount(NewCount);
  for i:=OldCount to NewCount - 1 do CreateItem(i);
 end else if NewCount < Count then begin
  for i:=NewCount to OldCount - 1 do FreeItem(i);
  inherited SetCount(NewCount);
 end;
end;

procedure TStringObjectArray.SetData(Index: Integer; const Value: Pointer);
begin
 GetObject(Index).Data:=Value;
end;

procedure TStringObjectArray.SetRefObj(Index: Integer;
  const Value: TObject);
begin
 GetObject(Index).RefObj:=Value;
end;

procedure TStringObjectArray.SetTag(Index: Integer; const Value: Integer);
begin
 GetObject(Index).Tag:=Value;
end;

procedure TStringObjectArray.SetValue(Index: Integer; const Value: TString);
begin
 GetObject(Index).Value:=Value;
end;

{ TStringObjectList }

function TStringObjectList.Add(const S: string): Integer;
begin
 Result:=FArray.Add;
```

```
 FArray.Value[Result]:=S;
end;

procedure TStringObjectList.Clear;
begin
 FArray.Count:=0;
end;

constructor TStringObjectList.Create;
begin
 inherited Create;
 FArray:=TStringObjectArray.Create;
end;

procedure TStringObjectList.Delete(Index: Integer);
begin
 FArray.Delete(Index);
end;

destructor TStringObjectList.Destroy;
begin
 FArray.Free;
 inherited;
end;

procedure TStringObjectList.Exchange(Index1, Index2: Integer);
begin
 FArray.Swap(Index1, Index2);
end;

function TStringObjectList.Get(Index: Integer): string;
begin
 Result:=FArray.Value[Index];
end;

function TStringObjectList.GetCount: Integer;
begin
 Result:=FArray.Count;
end;

function TStringObjectList.GetData(Index: Integer): Pointer;
begin
 Result:=FArray.Data[Index];
end;

function TStringObjectList.GetObject(Index: Integer): TObject;
begin
 Result:=FArray.RefObj[Index];
end;

function TStringObjectList.GetTag(Index: Integer): Integer;
begin
 Result:=FArray.Tag[Index];
end;

procedure TStringObjectList.Insert(Index: Integer; const S: string);
begin
 FArray.Insert(Index);
 FArray.Value[Index]:=S;
end;

procedure TStringObjectList.Put(Index: Integer; const S: string);
begin
 FArray.Value[Index]:=S;
end;

procedure TStringObjectList.PutObject(Index: Integer; AObject: TObject);
begin
 FArray.RefObj[Index]:=AObject;
```

```
end;

procedure TStringObjectList.SetData(Index: Integer; const Value: Pointer);
begin
 FArray.Data[Index]:=Value;
end;

procedure TStringObjectList.SetTag(Index: Integer; const Value: Integer);
begin
 FArray.Tag[Index]:=Value;
end;

{ TNetworkNeighborhood }

function TNetworkNeighborhood.Add(const S: string): Integer;
begin
 Result:=inherited Add(S);
 Objects[Result]:=TNetworkWorkgroup.Create;
end;

procedure TNetworkNeighborhood.Clear;
begin
 FArray.ForEach(Integer(Self), @TNetworkNeighborhood.FreeRefObj);
 inherited;
end;

function TNetworkNeighborhood.CopyPIDL(IDList: PItemIDList): PItemIDList;
var
 Size: Integer;
begin
 Size := GetPIDLSize(IDList);
 Result := CreatePIDL(Size);
 if Assigned(Result) then CopyMemory(Result, IDList, Size);
end;

constructor TNetworkNeighborhood.Create;
begin
 inherited Create;
 Refresh;
end;

function TNetworkNeighborhood.CreatePIDL(Size: Integer): PItemIDList;
var
 Malloc: IMalloc;
 HR: HResult;
begin
 Result := nil;
 HR := SHGetMalloc(Malloc);
 if Failed(HR) then  Exit;
 try
  Result := Malloc.Alloc(Size);
  if Assigned(Result) then FillChar(Result^, Size, 0);
 finally
 end;
end;

procedure TNetworkNeighborhood.Delete(Index: Integer);
begin
end;

procedure TNetworkNeighborhood.DisposePIDL(ID: PItemIDList);
var
 Malloc: IMalloc;
begin
 if ID = nil then Exit;
 OLECheck(SHGetMalloc(Malloc));
 Malloc.Free(ID);
end;
```

```
class function TNetworkNeighborhood.EnumObjects(
  ShellFolder: IShellFolder): IEnumIDList;
const
 Flags = SHCONTF_FOLDERS or SHCONTF_NONFOLDERS or SHCONTF_INCLUDEHIDDEN;
begin
 ShellFolder.EnumObjects(0, Flags, Result);
end;

function TNetworkNeighborhood.FindComputer(Name: TString): TString;
var
 i, j: Integer;
 List: TNetworkWorkgroup;
 S: TString;
begin
 Result:=' ' ;
 try
  for i:=0 to Count - 1 do begin
   List:=Objects[i] as TNetworkWorkgroup;
   for j:=0 to List.Count - 1 do begin
    S:=List[j];
    CleanUp(S);
    if EqualText(Name, S) then begin
     Result:=Strings[i];
     raise ComputerFound.Create(' ' );
    end;
   end;
  end;
 except
  if not (ExceptObject is ComputerFound) then raise;
 end;
end;

function TNetworkNeighborhood.FreeRefObj(Index: Integer;
  var Obj: TStringObject): Integer;
begin
 FreeAndNil(Obj.FRefObj);
 Result:=0;
end;

class function TNetworkNeighborhood.GetDisplayName(ShellFolder: IShellFolder;
  PIDL: PItemIDList): TString;
var
 StrRet: TStrRet;
 P: PChar;
begin
 Result := ' ' ;
 ShellFolder.GetDisplayNameOf(PIDL, SHGDN_NORMAL, StrRet);
 case StrRet.uType of
  STRRET_CSTR: SetString(Result, StrRet.cStr, lStrLen(StrRet.cStr));
  STRRET_OFFSET: begin
   P := @PIDL.mkid.abID[StrRet.uOffset - SizeOf(PIDL.mkid.cb)];
   SetString(Result, P, PIDL.mkid.cb - StrRet.uOffset);
  end;
  STRRET_WSTR: Result := StrRet.pOleStr;
 end;
 CleanUp(Result, True);
end;

function TNetworkNeighborhood.GetPIDLSize(IDList: PItemIDList): Integer;
begin
 Result := 0;
 if Assigned(IDList) then begin
  Result := SizeOf(IDList^.mkid.cb);
  while IDList^.mkid.cb <> 0 do begin
   Result := Result + IDList^.mkid.cb;
   IDList := NextPIDL(IDList);
  end;
 end;
end;
```

```
function TNetworkNeighborhood.GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
var
 Temp: PItemIDList;
begin
 Temp := CopyPIDL(PIDL);
 if Assigned(Temp) then StripLastID(Temp);
 if Temp.mkid.cb <> 0 then Result:=Temp else Result:=nil;
end;

function TNetworkNeighborhood.GetWorkgroup(Name: TString): TNetworkWorkgroup;
var
 Index: Integer;
begin
 Index:=IndexOf(Name);
 if Index<>-1 then Result:=Objects[Index] as TNetworkWorkgroup else Result:=nil;
end;

procedure TNetworkNeighborhood.Insert(Index: Integer; const S: string);
begin
end;

procedure TNetworkNeighborhood.ListComputers(Strings: TStrings);
var
 i, j: integer;
 L: TNetworkWorkgroup;
 S: TString;
begin
 Strings.BeginUpdate;
 try
  Strings.Clear;
  for i:=0 to Count - 1 do begin
   L:=Objects[i] as TNetworkWorkgroup;
   for j:=0 to L.Count - 1 do begin
    S:=L[j];
    CleanUp(S);
    Strings.Add(S);
   end;
  end;
 finally
  Strings.EndUpdate;
 end;
end;

procedure TNetworkNeighborhood.ListNetwork(Strings: TStrings);
var
 List: TStringList;
 i: Integer;
begin
 List:=TStringList.Create;
 try
  List.AddStrings(Self);
  for i:=0 to List.Count - 1 do List.Objects[i]:=TObject(1);
  for i:=0 to Count - 1 do begin
   List.AddStrings(Objects[i] as TStrings);
  end;
  for i:=Count to List.Count - 1 do List.Objects[i]:=nil;
  List.Sort;
  Strings.Assign(List);
 finally
  List.Free;
 end;
end;

function TNetworkNeighborhood.NextPIDL(IDList: PItemIDList): PItemIDList;
begin
 Result := IDList;
 Inc(PChar(Result), IDList^.mkid.cb);
end;
```

```
function TNetworkNeighborhood.OriginFolder: IShellFolder;
var
 Desktop: IShellFolder;
 S: TString;
 P: PWideChar;
 Len, Flags: LongWord;
 Machine, Workgroup, Network: PItemIDList;
begin
 S:=' \\' +GetComputerName;
 Len:=Length(S);
 P:=StringToOleStr(S);
 Flags:=0;
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Workgroup:=GetPrevPIDL(Machine);
 try
   Network:=GetPrevPIDL(Workgroup);
  try
   Desktop.BindToObject(Network, nil, IShellFolder, Pointer(Result));
  finally
   DisposePIDL(Network);
  end;
 finally
  DisposePIDL(Workgroup);
 end;
end;

function TNetworkNeighborhood.OriginFolderNT: IShellFolder;
var
 Desktop: IShellFolder;
 S: TString; W: WideString; P: PWideChar;
 Len, Flags: LongWord;
 Machine, Workgroup, Network: PItemIDList;
 NetShell: IShellFolder;
 Enum: IEnumIDList;
 ID: PItemIDList;
begin
 S:=' \\' +GetComputerName;
 Len:=Length(S);
 W:=S; P:=PWideChar(W);
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Workgroup:=GetPrevPIDL(Machine);
 Network:=GetPrevPIDL(Workgroup);
 Desktop.BindToObject(Network, nil, IShellFolder, NetShell);
 Enum:=EnumObjects(NetShell);
 Enum.Next(1, ID, Flags);
 NetShell.BindToObject(ID, nil, IShellFolder, Pointer(Result));
 DisposePIDL(Network);
 DisposePIDL(Workgroup);
end;

class procedure TNetworkNeighborhood.ParseFolder(Folder: IShellFolder;
  Items: TStringObjectList; StorePIDLs: Boolean);
var
 ID: PItemiDList;
 EnumList: IEnumIDList;
 NumIDs: LongWord;
 S: TString;
 Index: Integer;
begin
 Items.BeginUpdate;
 try
  Items.Clear;
  EnumList:=EnumObjects(Folder);
  if Assigned(EnumList) then while EnumList.Next(1, ID, NumIDs) = S_OK do begin
   S:=GetDisplayName(Folder, ID);
   Index:=Items.Add(S);
```

```
    if StorePIDLs then Items.Data[Index]:=ID;
   end;
  finally
   Items.EndUpdate;
  end;
end;

class procedure TNetworkNeighborhood.ParseFolderEx(Folder: IShellFolder;
  Items: TStrings);
var
 ID: PItemiDList;
 EnumList: IEnumIDList;
 NumIDs: LongWord;
 S: TString;
begin
 Items.BeginUpdate;
 try
  Items.Clear;
  EnumList:=EnumObjects(Folder);
  if Assigned(EnumList) then while EnumList.Next(1, ID, NumIDs) = S_OK do begin
   S:=GetDisplayName(Folder, ID);
   Items.Add(S);
  end;
 finally
  Items.EndUpdate;
 end;
end;

procedure TNetworkNeighborhood.Refresh;
var
 Network: IShellFolder;
 Workgroup: IShellFolder;
 i: Integer;
begin
 try
  if WinNT and (not Win2K) then Network:=OriginFolderNT else
   Network:=OriginFolder;
  ParseFolder(Network, Self, True);
  for i:=0 to Count - 1 do begin
   Network.BindToObject(PItemIDList(Data[i]), nil, IShellFolder, Workgroup);
   ParseFolder(Workgroup, Objects[i] as TStringObjectList, False);
   Workgroup:=nil;
  end;
 except
  raise ECannotFindNetwork.Create(SCannotFindNetwork);
 end;
end;

procedure TNetworkNeighborhood.StripLastID(IDList: PItemIDList);
var
 MarkerID: PItemIDList;
begin
 MarkerID := IDList;
 if Assigned(IDList) then begin
  while IDList.mkid.cb <> 0 do begin
   MarkerID := IDList;
   IDList := NextPIDL(IDList);
  end;
  MarkerID.mkid.cb := 0;
 end;
end;


procedure GetIPAddresses(Network: TNetworkNeighborhood; List: TStrings);
var
 Error: DWORD;
 HostEntry: PHostEnt;
 Data: WSAData;
 Address: In_Addr;
```

```
 i: Integer;
 TmpList: TStringList;
 S: TString;
begin
{ List.BeginUpdate;
 try}
  List.Clear;
  Error:=WSAStartup(MakeWord(1, 1), Data);
  if Error = 0 then begin
   TmpList:=TStringList.Create;
   try
    Network.ListComputers(TmpList);
    for i:=0 to TmpList.Count - 1 do begin
     HostEntry:=gethostbyname(PChar(TmpList[i]));
     Error:=GetLastError;
     if Error <> 0 then S:=‘ Unknown’  else begin
      Address:=PInAddr(HostEntry^.h_addr_list^)^;
      S:=inet_ntoa(Address);
     end;
     List.Add(Format(‘ %s [%s]’ , [TmpList[i], S]));
    end;
   finally
    TmpList.Free;
   end;
  end else begin
   List.Add(‘ Error’ );
  end;
{ finally
  List.EndUpdate;
 end;}
end;


function GetShellFolder(ComputerName: TString): IShellFolder;
var
 S: TString;
 W: WideString;
 P: PWideChar;
 Desktop: IShellFolder;
 Len, Flags: LongWord;
 Machine: PItemIDList;
begin
 S:=ComputerName;
 if Pos(‘ \\’ , S) <> 1 then S:=‘ \\’ +S;
 Len:=Length(S);
 W:=S;
 P:=@W[1];
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Desktop.BindToObject(Machine, nil, IShellFolder, Pointer(Result));
end;

function EnumSharedResources(ComputerName: TString; List: TStrings): Boolean;
var
 ShellFolder: IShellFolder;
begin
 ShellFolder:=GetShellFolder(ComputerName);
 Result:=Assigned(ShellFolder);
 if Result then TNetworkNeighborhood.ParseFolderEx(ShellFolder, List);
end;



function GetIPAddress(NetworkName: TString): TString;
var
 Error: DWORD;
 HostEntry: PHostEnt;
 Data: WSAData;
 Address: In_Addr;
```

```
begin
 Error:=WSAStartup(MakeWord(1, 1), Data);
 if Error = 0 then begin
  HostEntry:=gethostbyname(PChar(NetworkName));
  Error:=GetLastError();
  if Error = 0 then begin
   Address:=PInAddr(HostEntry^.h_addr_list^)^;
   Result:=inet_ntoa(Address);
  end else begin
   Result:=' Unknown' ;
  end;
 end else begin
  Result:=' Error' ;
 end;
 WSACleanup();
end;

end.
```

Основна програма

Файл Main.pas основної програми

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Dim, Networks, StdCtrls, ComCtrls, ImgList, ShellAPI, ShlObj, IpHlpApi,
IPtraff,
  ExtCtrls, About, Buttons;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    ClickMe: TButton;
    TreeView1: TTreeView;
    ImageList1: TImageList;
    ListView1: TListView;
    Memo2: TMemo;
    Button1: TButton;
    Label1: TLabel;
    Edit1: TEdit;
    Memo3: TMemo;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    MemoTR: TMemo;
    Timer1: TTimer;
    Label7: TLabel;
    Button2: TButton;
    Button3: TButton;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    MemoX: TMemo;
    Label8: TLabel;
    procedure ClickMeClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);

  private
    { Private declarations }
     procedure DOIpStuff;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
```

```
function GetShellFolder(CompName: TString): IShellFolder;
var
 S: TString;
 W: WideString;
 P: PWideChar;
 Desktop: IShellFolder;
 Len, Flags: LongWord;
 Machine: PItemIDList;
begin
 S:=CompName;
 if Pos(‘ \\’ , S) <> 1 then S:=‘ \\’ +S;
 Len:=Length(S);
 W:=S;
 P:=@W[1];
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Desktop.BindToObject(Machine, nil, IShellFolder, Pointer(Result));
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 Memo3.Lines.Clear;
 Screen.Cursor:=crHourGlass;
 try
  if not EnumSharedResources(Edit1.Text, Memo3.Lines)
   then raise Exception.Create(‘ Невірне ім"я комп"ютера’ );
 finally
  Screen.Cursor:=crDefault;
 end;
end;



procedure TForm1.ClickMeClick(Sender: TObject);
var
 N: TNetworkNeighborhood;
 List: TStringList;
 i, j: Integer;
 ListViewItem: TListItem;
 WorkgroupNode, ComputerNode: TTreeNode;
begin
 Screen.Cursor:=crHourGlass;
 try
  // Ініціалізація об’ єктів та сканування мережі
  N:=TNetworkNeighborhood.Create;
  try
   // Отримання списку всіх комп’ ютерів в мережі
   Memo1.Lines.Clear;
   N.ListComputers(Memo1.Lines);

   // Отримання списку всіх робочих груп і комп’ ютерів в мережі, відсортованих
в алфавітному порядку
   List:=TStringList.Create;
   ListView1.Items.Clear;
   try
    N.ListNetwork(List);
    for i:=0 to List.Count - 1 do begin
     ListViewItem:=ListView1.Items.Add;
     ListViewItem.Caption:=List[i];
     ListViewItem.ImageIndex:=Integer(List.Objects[i]);
    end;
   finally
    List.Free;
   end;
```

```
   // Побудова дерева робочих груп і комп’ ютерів в мережі
   TreeView1.Items.Clear;
   for i:=0 to N.Count - 1 do begin
    WorkgroupNode:=TreeView1.Items.Add(nil, N[i]);
    WorkgroupNode.ImageIndex:=1;
    WorkgroupNode.SelectedIndex:=1;
    for j:=0 to (N.Objects[i] as TStrings).Count - 1 do begin
     ComputerNode:=TreeView1.Items.AddChild(WorkgroupNode, (N.Objects[i] as
TStrings).Strings[j]);
     ComputerNode.ImageIndex:=0;
    end;
   end;

  // Отримання IP адрес комп’ ютерів
  GetIPAddresses(N, Memo2.Lines);

  finally
   N.Free;
  end;
  TreeView1.FullExpand;

 finally
  Screen.Cursor:=crDefault;
 end;
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
 Edit1.Text:=GetComputerName;

  if LoadIpHlp then
  begin
      DOIpStuff;
      Timer1.Enabled := true;
  end;

end;


procedure TForm1.Timer1Timer(Sender: TObject);
begin

   Timer1.Enabled := false;
   DoIPStuff;
   Timer1.Enabled := true;

end;


procedure TForm1.DOIpStuff;
begin

 Get_TCPTable( MemoX.Lines );
 Get_UDPTable( MemoTR.Lines );

end;


procedure TForm1.Button2Click(Sender: TObject);
begin

Form2.Show;

end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin

Form1.Close;

end;


procedure TForm1.SpeedButton2Click(Sender: TObject);
begin

Form1.Close;

end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin

Form2.Show;

end;


end.
```

Файл IPtraff.pas – відслідковування та динамічний перерозподіл трафіку мережі програмно визначаємого ЦОД на базі технологій Fujitsu

```
unit IPtraff;

interface

uses
  Windows, Messages, SysUtils, Classes, Dialogs, IpHlpApi;

const
  NULL_IP       = '   0.  0.  0.  0' ;


type
  TWellKnownPort = record
    Prt: DWORD;
    Srv: string[20];
  end;


const

  WellKnownPorts: array[1..32] of TWellKnownPort
  = (
//    ( Prt: 0; Srv:   ' RESRVED'  ),     {Зарезервовано}
    ( Prt: 7; Srv:   ' ECHO   '  ),     {Пінгування   }
    ( Prt: 9; Srv:   ' DISCARD'  ),
    ( Prt: 13; Srv:  ' DAYTIME'  ),
    ( Prt: 17; Srv:  ' QOTD   '  ),     {Показчик на день}
    ( Prt: 19; Srv:  ' CHARGEN'  ),     {Генератор символів}
    ( Prt: 20; Srv:  ' FTPDATA'  ),     { Протокол File Transfer Protocol -
дані}
    ( Prt: 21; Srv:  ' FTPCTRL'  ),     { Протокол File Transfer Protocol -
управління}
    ( Prt: 22; Srv:  ' SSH    '  ),
    ( Prt: 23; Srv:  ' TELNET '  ),
    ( Prt: 25; Srv:  ' SMTP   '  ),     { Протокол Simple Mail Transfer
Protocol}
    ( Prt: 37; Srv:  ' TIME   '  ),     { Протокол Time Protocol }
    ( Prt: 43; Srv:  ' WHOIS  '  ),     { WHO IS сервіс  }
    ( Prt: 53; Srv:  ' DNS    '  ),     { Domain Name Service }
    ( Prt: 67; Srv:  ' BOOTPS '  ),     { BOOTP сервер }
    ( Prt: 68; Srv:  ' BOOTPC '  ),     { BOOTP клієнт }
    ( Prt: 69; Srv:  ' TFTP   '  ),     { Стандартний  FTP  }
    ( Prt: 70; Srv:  ' GOPHER '  ),     { Протокол Gopher       }
    ( Prt: 79; Srv:  ' FINGER '  ),     { Протокол Finger       }
    ( Prt: 80; Srv:  ' HTTP   '  ),     { Протокол HTTP         }
    ( Prt: 88; Srv:  ' KERBROS'  ),     { Протокол Kerberos     }
    ( Prt: 109; Srv: ' POP2   '  ),     { Протокол Post Office Protocol Version
2 }
    ( Prt: 110; Srv: ' POP3   '  ),     { Протокол Post Office Protocol Version
3 }
    ( Prt: 111; Srv: ' SUN_RPC'  ),     { SUN віддалений виклик функцій }
    ( Prt: 119; Srv: ' NNTP   '  ),     { Протокол Network News Transfer
Protocol }
    ( Prt: 123; Srv: ' NTP    '  ),     { Протокол Network Time protocol
}
    ( Prt: 135; Srv: ' DCOMRPC'  ),     { Локальний сервіce             }
    ( Prt: 137; Srv: ' NBNAME '  ),     { NETBIOS сервіс імен          }
    ( Prt: 138; Srv: ' NBDGRAM'  ),     { NETBIOS сервіс датаграм     }
    ( Prt: 139; Srv: ' NBSESS '  ),     { NETBIOS сервіс сесій        }
    ( Prt: 143; Srv: ' IMAP   '  ),     { Протокол Internet Message Access
Protocol }
    ( Prt: 161; Srv: ' SNMP   '  ),     { Протокол Simple Netw. Management
Protocol }
    ( Prt: 169; Srv: ' SEND   '  )
    );
```

```
  const
  ICMP_ERROR_BASE = 11000;
  IcmpErr : array[1..22] of string =
  (
   ' IP_BUFFER_TOO_SMALL',' IP_DEST_NET_UNREACHABLE' , '
IP_DEST_HOST_UNREACHABLE' ,
   ' IP_PROTOCOL_UNREACHABLE' , ' IP_DEST_PORT_UNREACHABLE' , ' IP_NO_RESOURCES'
,
   ' IP_BAD_OPTION',' IP_HARDWARE_ERROR' , ' IP_PACKET_TOO_BIG' , '
IP_REQUEST_TIMED_OUT' ,
   ' IP_BAD_REQUEST' ,' IP_BAD_ROUTE' , ' IP_TTL_EXPIRED_TRANSIT' ,
   ' IP_TTL_EXPIRED_REASSEM' ,' IP_PARAMETER_PROBLEM' , ' IP_SOURCE_QUENCH' ,
   ' IP_OPTION_TOO_BIG' , ' IP_BAD_DESTINATION' ,' IP_ADDRESS_DELETED' ,
   ' IP_SPEC_MTU_CHANGE' , ' IP_MTU_CHANGE' , ' IP_UNLOAD'
  );


  ARPEntryType  : array[1..4] of string = ( ' Інший' , ' Несправний' ,
    ' Динамічний' , ' Статичний'
    );
  TCPConnState  :
    array[1..12] of string =
 ( ' CLOSED' , ' READ' , ' SYN_SENT' ,
    ' SYN_RCVD' , ' ESTABLISHED' , ' FIN_WAIT1' ,
    ' FIN_WAIT2' , ' WAIT' , ' CLOSING' ,
    ' LAST_ACK' , ' WAIT' , ' DELETE_TCB'    );

  TCPToAlgo     : array[1..4] of string =
    ( ' Const.Timeout' , ' MIL-STD-1778' ,
    ' Van Jacobson' , ' Other'  );

  IPForwTypes   : array[1..4] of string =
    ( ' other' , ' invalid' , ' local' , ' remote'  );

  IPForwProtos  : array[1..18] of string =
    ( ' OTHER' , ' LOCAL' , ' NETMGMT' , ' ICMP' , ' EGP' ,
    ' GGP' , ' HELO' , ' RIP' , ' IS_IS' , ' ES_IS' ,
    ' CISCO' , ' BBN' , ' OSPF' , ' BGP' , ' BOOTP' ,
    ' AUTO_STAT' , ' STATIC' , ' NOT_DOD'  );

type

// для IpHlpNetworkParams
  TNetworkParams = record
    HostName: string ;
    DomainName: string ;
    CurrentDnsServer: string ;
    DnsServerTot: integer ;
    DnsServerNames: array [0..9] of string ;
    NodeType: UINT;
    ScopeID: string ;
    EnableRouting: UINT;
    EnableProxy: UINT;
    EnableDNS: UINT;
  end;

  TIfRows = array of TMibIfRow ; // динамічний масив колонок

// для IpHlpAdaptersInfo
  TAdaptorInfo = record
    AdapterName: string ;
    Description: string ;
    MacAddress: string ;
    Index: DWORD;
    aType: UINT;
    DHCPEnabled: UINT;
    CurrIPAddress: string ;
    CurrIPMask: string ;
```

```
    IPAddressTot: integer ;
    IPAddressList: array of string ;
    IPMaskList: array of string ;
    GatewayTot: integer ;
    GatewayList: array of string ;
    DHCPTot: integer ;
    DHCPServer: array of string ;
    HaveWINS: BOOL;
    PrimWINSTot: integer ;
    PrimWINSServer: array of string ;
    SecWINSTot: integer ;
    SecWINSServer: array of string ;
    LeaseObtained: LongInt ;
    LeaseExpires: LongInt;
  end ;

  TAdaptorRows = array of TAdaptorInfo ;



function IpHlpAdaptersInfo(var AdpTot: integer;var AdpRows: TAdaptorRows):
integer ;
procedure Get_AdaptersInfo( List: TStrings );
function IpHlpNetworkParams (var NetworkParams: TNetworkParams): integer ;
procedure Get_NetworkParams( List: TStrings );
procedure Get_ARPTable( List: TStrings );
procedure Get_TCPTable( List: TStrings );
procedure Get_TCPStatistics( List: TStrings );
function IpHlpTCPStatistics (var TCPStats: TMibTCPStats): integer ;
procedure Get_UDPTable( List: TStrings );
procedure Get_UDPStatistics( List: TStrings );
function IpHlpUdpStatistics (UdpStats: TMibUDPStats): integer ;
procedure Get_IPAddrTable( List: TStrings );
procedure Get_IPForwardTable( List: TStrings );
procedure Get_IPStatistics( List: TStrings );
function IpHlpIPStatistics (var IPStats: TMibIPStats): integer ;
function Get_RTTAndHopCount( IPAddr: DWORD; MaxHops: Longint;
  var RTT: longint; var HopCount: longint ): integer;
procedure Get_ICMPStats( ICMPIn, ICMPOut: TStrings );
function IpHlpIfTable(var IfTot: integer; var IfRows: TIfRows): integer ;
procedure Get_IfTable( List: TStrings );
function IpHlpIfEntry(Index: integer; var IfRow: TMibIfRow): integer ;
procedure Get_RecentDestIPs( List: TStrings );

// утіліти перетворення
function MacAddr2Str( MacAddr: TMacAddress; size: integer ): string;
function IpAddr2Str( IPAddr: DWORD ): string;
function Str2IpAddr( IPStr: string ): DWORD;
function Port2Str( nwoPort: DWORD ): string;
function Port2Wrd( nwoPort: DWORD ): DWORD;
function Port2Svc( Port: DWORD ): string;
function ICMPErr2Str( ICMPErrCode: DWORD) : string;

implementation

var
  RecentIPs     : TStringList;


{ перетворення токена у рядок, потім рядок знищується }
function NextToken( var s: string; Separator: char ): string;
var
  Sep_Pos       : byte;
begin
  Result := ' ' ;
  if length( s ) > 0 then begin
    Sep_Pos := pos( Separator, s );
    if Sep_Pos > 0 then begin
```

```
      Result := copy( s, 1, Pred( Sep_Pos ) );
      Delete( s, 1, Sep_Pos );
    end
    else begin
      Result := s;
      s := ' ' ;
    end;
  end;
end;


{ пертворення MAC-адреси до ww-xx-yy-zz рядка }
function MacAddr2Str( MacAddr: TMacAddress; size: integer ): string;
var
  i             : integer;
begin
  if Size = 0 then
  begin
    Result := ' 00-00-00-00-00-00' ;
    EXIT;
  end
  else Result := ' ' ;
  //
  for i := 1 to Size do
    Result := Result + IntToHex( MacAddr[i], 2 ) + ' -' ;
  Delete( Result, Length( Result ), 1 );
end;


{ перетворення  IP-адреси  в мережний байт типу DWORD }
function IpAddr2Str( IPAddr: DWORD ): string;
var
  i             : integer;
begin
  Result := ' ' ;
  for i := 1 to 4 do
  begin
    Result := Result + Format( ' %3d.' , [IPAddr and $FF] );
    IPAddr := IPAddr shr 8;
  end;
  Delete( Result, Length( Result ), 1 );
end;


{ перетворення  крапкову десяткову IP-адресу  в мережний байт типу  DWORD}
function Str2IpAddr( IPStr: string ): DWORD;
var
  i             : integer;
  Num           : DWORD;
begin
  Result := 0;
  for i := 1 to 4 do
  try
    Num := ( StrToInt( NextToken( IPStr, ' .'  ) ) ) shl 24;
    Result := ( Result shr 8 ) or Num;
  except
    Result := 0;
  end;

end;


{ перетворення  номер порту в мережний байт типу  DWORD }
function Port2Wrd( nwoPort: DWORD ): DWORD;
begin
  Result := Swap( WORD( nwoPort ) );
end;
```

```
{ перетворення  номера порту в мережний байт типу  string }
function Port2Str( nwoPort: DWORD ): string;
begin
  Result := IntToStr( Port2Wrd( nwoPort ) );
end;


{ перетворення  відомого номера порту у сервіс ID }
function Port2Svc( Port: DWORD ): string;
var
  i             : integer;
begin
  Result := Format( ' %4d' , [Port] ); // у випадку якщо порт не знайдено
  for i := Low( WellKnownPorts ) to High( WellKnownPorts ) do
    if Port = WellKnownPorts[i].Prt then
    begin
      Result := WellKnownPorts[i].Srv;
      BREAK;
    end;
end;


{ фіксуємо параметри мережі }

procedure Get_NetworkParams( List: TStrings );
var
    NetworkParams: TNetworkParams ;
    I, ErrorCode: integer ;
begin
    if not Assigned( List ) then EXIT;
    List.Clear;
    ErrorCode := IpHlpNetworkParams (NetworkParams) ;
    if ErrorCode <> 0 then
    begin
        List.Add (SysErrorMessage (ErrorCode));
        exit;
    end ;
    with NetworkParams do
    begin
        List.Add( ' HOSTNAME          : '  + HostName );
        List.Add( ' DOMAIN            : '  + DomainName );
        List.Add( ' NETBIOS NODE TYPE : '  + NETBIOSTypes[NodeType] );
        List.Add( ' DHCP SCOPE        : '  + ScopeID );
        List.Add( ' ROUTING ENABLED   : '  + IntToStr( EnableRouting ) );
        List.Add( ' PROXY   ENABLED   : '  + IntToStr( EnableProxy ) );
        List.Add( ' DNS     ENABLED   : '  + IntToStr( EnableDNS ) );
        if DnsServerTot <> 0 then
        begin
            for I := 0 to Pred (DnsServerTot) do
            List.Add( ' DNS SERVER ADDR   : '  + DnsServerNames [I] ) ;
        end ;
    end ;
end ;

//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
function IpHlpNetworkParams (var NetworkParams: TNetworkParams): integer ;
var
  FixedInfo     : PTFixedInfo;         //
  InfoSize      : Longint;
  PDnsServer    : PTIP_ADDR_STRING ;   //
begin
    InfoSize := 0 ;   //
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetNetworkParams( Nil, @InfoSize );  //
    if result <> ERROR_BUFFER_OVERFLOW then exit ; //
    GetMem (FixedInfo, InfoSize) ;                    //
    try
    result := GetNetworkParams( FixedInfo, @InfoSize );   //
```

```
    if result <> ERROR_SUCCESS then exit ;
    NetworkParams.DnsServerTot := 0 ;
    with FixedInfo^ do
    begin
        NetworkParams.HostName := trim (HostName) ;
        NetworkParams.DomainName := trim (DomainName) ;
        NetworkParams.ScopeId := trim (ScopeID) ;
        NetworkParams.NodeType := NodeType ;
        NetworkParams.EnableRouting := EnableRouting ;
        NetworkParams.EnableProxy := EnableProxy ;
        NetworkParams.EnableDNS := EnableDNS ;
        NetworkParams.DnsServerNames [0] := DNSServerList.IPAddress ;  //
        if NetworkParams.DnsServerNames [0] <> ' '  then
                                        NetworkParams.DnsServerTot := 1 ;
        PDnsServer := DnsServerList.Next;
        while PDnsServer <> Nil do
        begin
            NetworkParams.DnsServerNames [NetworkParams.DnsServerTot] :=
                                        PDnsServer^.IPAddress ;  //
            inc (NetworkParams.DnsServerTot) ;
            if NetworkParams.DnsServerTot >=
                        Length (NetworkParams.DnsServerNames) then exit ;
            PDnsServer := PDnsServer.Next ;
        end;
    end ;
    finally
       FreeMem (FixedInfo) ;                    //
    end ;
end;

//---------------------------------------------------------------------------

function ICMPErr2Str( ICMPErrCode: DWORD) : string;
begin
   Result := ' UnknownError : ' + IntToStr( ICMPErrCode );
   dec( ICMPErrCode, ICMP_ERROR_BASE );
   if ICMPErrCode in [Low(ICMpErr)..High(ICMPErr)] then
     Result := ICMPErr[ ICMPErrCode];
end;


//---------------------------------------------------------------------------

// включаємо байти вводу/виводу для кожного адаптеру

function IpHlpIfTable(var IfTot: integer; var IfRows: TIfRows): integer ;
var
  I,
  TableSize  : integer;
  pBuf, pNext : PChar;
begin
  result := ERROR_NOT_SUPPORTED ;
  if NOT LoadIpHlp then exit ;
  SetLength (IfRows, 0) ;
  IfTot := 0 ; //
  TableSize := 0;
   //  перший виклик: беремо необхідний розмір пам' яті
  result := GetIfTable (Nil, @TableSize, false) ;  //
  if result <> ERROR_INSUFFICIENT_BUFFER then exit ;
  GetMem( pBuf, TableSize );
  try
      FillChar (pBuf^, TableSize, #0);  // очищуємо буфер, з W98 не потрібно

   // беремо показчик на таблицю
      result := GetIfTable (PTMibIfTable (pBuf), @TableSize, false) ;
      if result <> NO_ERROR then exit ;
      IfTot := PTMibIfTable (pBuf)^.dwNumEntries ;
      if IfTot = 0 then exit ;
      SetLength (IfRows, IfTot) ;
```

```
      pNext := pBuf + SizeOf(IfTot) ;
      for i := 0 to Pred (IfTot) do
      begin
         IfRows [i] := PTMibIfRow (pNext )^ ;
         inc (pNext, SizeOf (TMibIfRow)) ;
      end;
  finally
      FreeMem (pBuf) ;
  end ;
end;

procedure Get_IfTable( List: TStrings );
var
  IfRows        : TIfRows ;
  Error, I      : integer;
  NumEntries    : integer;
  sDescr, sIfName: string ;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  SetLength (IfRows, 0) ;
  Error := IpHlpIfTable (NumEntries, IfRows) ;
  if (Error <> 0) then
      List.Add( SysErrorMessage( GetLastError ) )
  else if NumEntries = 0 then
      List.Add( ' Даних немає.'  )
  else
  begin
      for I := 0 to Pred (NumEntries) do
      begin
          with IfRows [I] do
          begin
             if wszName [1] = #0 then
                 sIfName := ' '
             else
                 sIfName := WideCharToString (@wszName) ;  // перетворюємо
Unicode у string
             sIfName := trim (sIfName) ;
             sDescr := bDescr ;
             sDescr := trim (sDescr);
             List.Add (Format (
               ' %0.8x - %3d - %16s - %8d - %12d - %2d - %2d - %10d - %10d  -
%-s - %-s' ,
               [dwIndex, dwType, MacAddr2Str( TMacAddress( bPhysAddr ),
               dwPhysAddrLen ), dwMTU, dwSpeed, dwAdminStatus,
               dwOPerStatus, Int64 (dwInOctets), Int64 (dwOutOctets),  //
counters are 32-bit
               sIfName, sDescr] )  //  , додаємо введення/вивід
               );
          end;
      end ;
  end ;
  SetLength (IfRows, 0) ;  // звільняємо пам' ять
end ;

function IpHlpIfEntry(Index: integer; var IfRow: TMibIfRow): integer ;
begin
  result := ERROR_NOT_SUPPORTED ;
  if NOT LoadIpHlp then exit ;
  FillChar (IfRow, SizeOf (TMibIfRow), #0);  // очищуємо буфер, з W98 не
потрібно
  IfRow.dwIndex := Index ;
  result := GetIfEntry (@IfRow) ;
end ;

//---------------------------------------------------------------------------
{ інформація про мережні адаптери }
```

```
function IpHlpAdaptersInfo(var AdpTot: integer; var AdpRows: TAdaptorRows):
integer ;
var
  BufLen        : DWORD;
  AdapterInfo   : PTIP_ADAPTER_INFO;
  PIpAddr       : PTIP_ADDR_STRING;
  PBuf          : PCHAR ;
  I             : integer ;
begin
  SetLength (AdpRows, 4) ;
  AdpTot := 0 ;
  BufLen := 0 ;
  result := GetAdaptersInfo( Nil, @BufLen );
  if (result <> ERROR_INSUFFICIENT_BUFFER) and (result = NO_ERROR) then exit ;
  GetMem( pBuf, BufLen );
  try
      FillChar (pBuf^, BufLen, #0);  // очищуємо буфер
      result := GetAdaptersInfo( PTIP_ADAPTER_INFO (PBuf), @BufLen );
      if result = NO_ERROR then
      begin
         AdapterInfo := PTIP_ADAPTER_INFO (PBuf) ;
         while ( AdapterInfo <> nil ) do
         begin
            AdpRows [AdpTot].IPAddressTot := 0 ;
            SetLength (AdpRows [AdpTot].IPAddressList, 2) ;
            SetLength (AdpRows [AdpTot].IPMaskList, 2) ;
            AdpRows [AdpTot].GatewayTot := 0 ;
            SetLength (AdpRows [AdpTot].GatewayList, 2) ;
            AdpRows [AdpTot].DHCPTot := 0 ;
            SetLength (AdpRows [AdpTot].DHCPServer, 2) ;
            AdpRows [AdpTot].PrimWINSTot := 0 ;
            SetLength (AdpRows [AdpTot].PrimWINSServer, 2) ;
            AdpRows [AdpTot].SecWINSTot := 0 ;
            SetLength (AdpRows [AdpTot].SecWINSServer, 2) ;
            AdpRows [AdpTot].CurrIPAddress := NULL_IP;
            AdpRows [AdpTot].CurrIPMask := NULL_IP;
            AdpRows [AdpTot].AdapterName := Trim( string(
AdapterInfo^.AdapterName ) );
            AdpRows [AdpTot].Description := Trim( string(
AdapterInfo^.Description ) );
            AdpRows [AdpTot].MacAddress := MacAddr2Str( TMacAddress(
                                AdapterInfo^.Address ),
AdapterInfo^.AddressLength ) ;
            AdpRows [AdpTot].Index := AdapterInfo^.Index ;
            AdpRows [AdpTot].aType := AdapterInfo^.aType ;
            AdpRows [AdpTot].DHCPEnabled := AdapterInfo^.DHCPEnabled ;
            if AdapterInfo^.CurrentIPAddress <> Nil then
            begin
                AdpRows [AdpTot].CurrIPAddress :=
AdapterInfo^.CurrentIPAddress.IpAddress ;
                AdpRows [AdpTot].CurrIPMask :=
AdapterInfo^.CurrentIPAddress.IpMask ;
            end ;

        // беремо список IP адрес та масок для IPAddressList
            I := 0 ;
            PIpAddr := @AdapterInfo^.IPAddressList ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].IPAddressList [I] := PIpAddr.IpAddress ;
                AdpRows [AdpTot].IPMaskList [I] := PIpAddr.IpMask ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].IPAddressList) <= I then
                begin
                     SetLength (AdpRows [AdpTot].IPAddressList, I * 2) ;
                     SetLength (AdpRows [AdpTot].IPMaskList, I * 2) ;
                end ;
            end ;
```

```
            AdpRows [AdpTot].IPAddressTot := I ;

        // беремо список IP адрес для GatewayList
            I := 0 ;
            PIpAddr := @AdapterInfo^.GatewayList ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].GatewayList [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].GatewayList) <= I then
                            SetLength (AdpRows [AdpTot].GatewayList, I * 2) ;
            end ;
            AdpRows [AdpTot].GatewayTot := I ;

        // беремо список IP адрес для GatewayList
            I := 0 ;
            PIpAddr := @AdapterInfo^.DHCPServer ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].DHCPServer [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].DHCPServer) <= I then
                            SetLength (AdpRows [AdpTot].DHCPServer, I * 2) ;
            end ;
            AdpRows [AdpTot].DHCPTot := I ;

        // беремо список IP адрес для PrimaryWINSServer
            I := 0 ;
            PIpAddr := @AdapterInfo^.PrimaryWINSServer ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].PrimWINSServer [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].PrimWINSServer) <= I then
                            SetLength (AdpRows [AdpTot].PrimWINSServer, I * 2)
;
            end ;
            AdpRows [AdpTot].PrimWINSTot := I ;

       // беремо список IP адрес для SecondaryWINSServer
            I := 0 ;
            PIpAddr := @AdapterInfo^.SecondaryWINSServer ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].SecWINSServer [I] := PIpAddr.IpAddress ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].SecWINSServer) <= I then
                            SetLength (AdpRows [AdpTot].SecWINSServer, I * 2) ;
            end ;
            AdpRows [AdpTot].SecWINSTot := I ;

            AdpRows [AdpTot].LeaseObtained := AdapterInfo^.LeaseObtained ;
            AdpRows [AdpTot].LeaseExpires := AdapterInfo^.LeaseExpires ;

            inc (AdpTot) ;
            if Length (AdpRows) <= AdpTot then
                        SetLength (AdpRows, AdpTot * 2) ;  // more memory
            AdapterInfo := AdapterInfo^.Next;
         end ;
         SetLength (AdpRows, AdpTot) ;
      end ;
  finally
      FreeMem( pBuf );
  end ;
end ;
```

```
procedure Get_AdaptersInfo( List: TStrings );
var
  AdpTot: integer;
  AdpRows: TAdaptorRows ;
  Error: DWORD ;
  I: integer ;
  //J: integer ;  jpt
  //S: string ;         id.
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  SetLength (AdpRows, 0) ;
  AdpTot := 0 ;
  Error := IpHlpAdaptersInfo(AdpTot, AdpRows) ;
  if (Error <> 0) then
      List.Add( SysErrorMessage( GetLastError ) )
  else if AdpTot = 0 then
      List.Add( ' Даних немає.'  )
  else
  begin
      for I := 0 to Pred (AdpTot) do
      begin
        with AdpRows [I] do
        begin
            //List.Add(AdapterName + ' -'  + Description ); // jpt : не корисне
            List.Add( Format(' %8.8x - %6s - %16s - %2d - %16s - %16s - %16s' ,
                [Index, AdaptTypes[aType], MacAddress, DHCPEnabled,
                GatewayList [0], DHCPServer [0], PrimWINSServer [0]] ) );
            {if IPAddressTot <> 0 then    // jpt : not useful
            begin
                S := ' '  ;
                for J := 0 to Pred (IPAddressTot) do
                        S := S + IPAddressList [J] + ' /'  + IPMaskList [J] + '
- '  ;
                List.Add(IntToStr (IPAddressTot) + '  IP Addresse(s): '  + S);
            end ;
            List.Add( '    '  ); }
        end ;
      end ;
  end ;
  SetLength (AdpRows, 0) ;
end ;

//---------------------------------------------------------------------------
{ зчитуємо кількість раундів, та час звертання до IP }
function Get_RTTAndHopCount( IPAddr: DWORD; MaxHops: Longint; var RTT: Longint;
  var HopCount: Longint ): integer;
begin
  if not GetRTTAndHopCount( IPAddr, @HopCount, MaxHops, @RTT ) then
  begin
    Result := GetLastError;
    RTT := -1; //
    HopCount := -1;
  end
  else
    Result := NO_ERROR;
end;

//---------------------------------------------------------------------------
{ ARP-таблиця – список відношень між віддаленими IP та віддаленими MAC-адресами.

}
procedure Get_ARPTable( List: TStrings );
var
  IPNetRow      : TMibIPNetRow;
  TableSize     : DWORD;
  NumEntries    : DWORD;
  ErrorCode     : DWORD;
```

```
  i             : integer;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  //  перший виклик: беремо довжину таблиці
  TableSize := 0;
  ErrorCode := GetIPNetTable( Nil, @TableSize, false );   //
  //
  if ErrorCode = ERROR_NO_DATA then
  begin
    List.Add( '  ARP-кеш пустий.'  );
    EXIT;
  end;
  // беремо таблицю
  GetMem( pBuf, TableSize );
  NumEntries := 0 ;
  try
  ErrorCode := GetIpNetTable( PTMIBIPNetTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMIBIPNetTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then //.
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        IPNetRow := PTMIBIPNetRow( PBuf )^;
        with IPNetRow do
          List.Add( Format( ' %8x - %12s - %16s - %10s' ,
                            [dwIndex, MacAddr2Str( bPhysAddr, dwPhysAddrLen ),
                            IPAddr2Str( dwAddr ), ARPEntryType[dwType]
                            ]));
        inc( pBuf, SizeOf( IPNetRow ) );
      end;
    end
    else
      List.Add( '  ARP-кеш пустий.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );

  // we _must_ restore pointer!
  finally
      dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( IPNetRow ) );
      FreeMem( pBuf );
  end ;
end;


//-----------------------------------------------------------------------------
procedure Get_TCPTable( List: TStrings );
var
  TCPRow        : TMIBTCPRow;
  i,
    NumEntries  : integer;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  DestIP        : string;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  RecentIPs.Clear;
  //  перший виклик : беремо розмір таблиці
  TableSize := 0;
  NumEntries := 0 ;
  ErrorCode := GetTCPTable(Nil, @TableSize, false );  //
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
```

```
    EXIT;

  // беремо розмір пам' яти, який потрібно та здійснюємо виклик знову
  GetMem( pBuf, TableSize );
  // беремо таблицю
  ErrorCode := GetTCPTable( PTMIBTCPTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin

    NumEntries := PTMIBTCPTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        TCPRow := PTMIBTCPRow( pBuf )^; // беремо наступний запис
        with TCPRow do
        begin
          if dwRemoteAddr = 0 then
            dwRemotePort := 0;
          DestIP := IPAddr2Str( dwRemoteAddr );
          List.Add(
            Format( ' %15s : %-7s -> %15s : %-7s -> %-16s' ,
            [IpAddr2Str( dwLocalAddr ),
            Port2Svc( Port2Wrd( dwLocalPort ) ),
              DestIP,
              Port2Svc( Port2Wrd( dwRemotePort ) ),
              TCPConnState[dwState]
              ] ) );
         //
            if (not ( dwRemoteAddr = 0 ))
            and ( RecentIps.IndexOf(DestIP) = -1 ) then
               RecentIPs.Add( DestIP );
        end;
        inc( pBuf, SizeOf( TMIBTCPRow ) );
      end;
    end;
  end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( TMibTCPRow ) );
  FreeMem( pBuf );
end;

//---------------------------------------------------------------------------
procedure Get_TCPStatistics( List: TStrings );
var
  TCPStats     : TMibTCPStats;
  ErrorCode    : DWORD;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  if NOT LoadIpHlp then exit ;
  ErrorCode := GetTCPStatistics( @TCPStats );
  if ErrorCode = NO_ERROR then
    with TCPStats do
    begin
      List.Add( ' Алгоритм повторної передачі: '  + TCPToAlgo[dwRTOAlgorithm] );
      List.Add( ' Мінімальний час виведення         : '  + IntToStr( dwRTOMin )
+ '  ms'  );
      List.Add( ' Максимальний час виведення         : '  + IntToStr( dwRTOMax )
+ '  ms'  );
      List.Add( ' Максимальне число підключень : '  + IntToStr( dwRTOAlgorithm )
);
      List.Add( ' Активні підключення             : '  + IntToStr( dwActiveOpens
) );
      List.Add( ' Пасивні підключення            : '  + IntToStr( dwPassiveOpens
) );
```

```
      List.Add( ' Невдала спроба відкриття     : '  + IntToStr( dwAttemptFails )
);
      List.Add( ' Відновлений зв'язок   : '  + IntToStr( dwEstabResets ) );
      List.Add( ' Поточний зв'язок .: '  + IntToStr( dwCurrEstab ) );
      List.Add( ' Отримано сегментів        : '  + IntToStr( dwInSegs ) );
      List.Add( ' Відправлено сегментів            : '  + IntToStr( dwOutSegs )
);
      List.Add( ' Ретрансльовано сегментів   : '  + IntToStr( dwReTransSegs ) );
      List.Add( ' Помилки входження          : '  + IntToStr( dwInErrs ) );
      List.Add( ' Скидання виведення          : '  + IntToStr( dwOutRsts ) );
      List.Add( ' Сукупні зв'язки   : '  + IntToStr( dwNumConns ) );
    end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
end;

function IpHlpTCPStatistics (var TCPStats: TMibTCPStats): integer ;
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetTCPStatistics( @TCPStats );
end;

//-------------------------------------------------------------------------
procedure Get_UDPTable( List: TStrings );
var
  UDPRow        : TMIBUDPRow;
  i,
    NumEntries  : integer;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;

  //  перший виклик : беремо розмір таблиці
  TableSize := 0;
  NumEntries := 0 ;
  ErrorCode := GetUDPTable(Nil, @TableSize, false );
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  // беремо потрібний розмір пам'яті, викликаємо знову
  GetMem( pBuf, TableSize );

  // беремо таблицю
  ErrorCode := GetUDPTable( PTMIBUDPTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMIBUDPTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        UDPRow := PTMIBUDPRow( pBuf )^; // беремо наступний запис
        with UDPRow do
          List.Add( Format( ' %15s : %-6s' ,
            [IpAddr2Str( dwLocalAddr ),
            Port2Svc( Port2Wrd( dwLocalPort ) )
              ] ) );
        inc( pBuf, SizeOf( TMIBUDPRow ) );
      end;
    end
    else
      List.Add( ' Даних немає.'  );
  end
  else
```

```
    List.Add( SyserrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( TMibUDPRow ) );
  FreeMem( pBuf );
end;

//-----------------------------------------------------------------------------
procedure Get_IPAddrTable( List: TStrings );
var
  IPAddrRow     : TMibIPAddrRow;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
  NumEntries    : DWORD;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  TableSize := 0; ;
  NumEntries := 0 ;
  //  перший виклик: беремо довжину таблиці
  ErrorCode := GetIpAddrTable(Nil, @TableSize, true );  //
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  GetMem( pBuf, TableSize );
  // беремо таблицю
  ErrorCode := GetIpAddrTable( PTMibIPAddrTable( pBuf ), @TableSize, true );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMibIPAddrTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) );
      for i := 1 to NumEntries do
      begin
        IPAddrRow := PTMIBIPAddrRow( pBuf )^;
        with IPAddrRow do
          List.Add( Format( ' %8.8x | %15s | %15s | %15s | %8.8d' ,
            [dwIndex,
            IPAddr2Str( dwAddr ),
              IPAddr2Str( dwMask ),
              IPAddr2Str( dwBCastAddr ),
              dwReasmSize
              ] ) );
        inc( pBuf, SizeOf( TMIBIPAddrRow ) );
      end;
    end
    else
      List.Add( ' Даних немає.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );

  // we must restore pointer!
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( IPAddrRow ) );
  FreeMem( pBuf );
end;

//-----------------------------------------------------------------------------
{ беремо дані з таблиці маршрутизатора Cisco}
procedure Get_IPForwardTable( List: TStrings );
var
  IPForwRow     : TMibIPForwardRow;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
  NumEntries    : DWORD;
begin
```

```
  if not Assigned( List ) then EXIT;
  List.Clear;
  TableSize := 0;

  //  перший виклик: беремо довжину таблиці
  NumEntries := 0 ;
  ErrorCode := GetIpForwardTable(Nil, @TableSize, true);
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  // беремо таблицю
  GetMem( pBuf, TableSize );
  ErrorCode := GetIpForwardTable( PTMibIPForwardTable( pBuf ), @TableSize,
true);
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMibIPForwardTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) );
      for i := 1 to NumEntries do
      begin
        IPForwRow := PTMibIPForwardRow( pBuf )^;
        with IPForwRow do
        begin
          if (dwForwardType < 1)
          or (dwForwardType > 4) then
                  dwForwardType := 1 ;   //  , враховуємо погані значення
          List.Add( Format(
            ' %15s | %15s | %15s | %8.8x | %7s |   %5.5d |  %7s |   %2.2d' ,
            [IPAddr2Str( dwForwardDest ),
            IPAddr2Str( dwForwardMask ),
              IPAddr2Str( dwForwardNextHop ),
              dwForwardIFIndex,
              IPForwTypes[dwForwardType],
              dwForwardNextHopAS,
              IPForwProtos[dwForwardProto],
              dwForwardMetric1
              ] ) );
        end ;
        inc( pBuf, SizeOf( TMibIPForwardRow ) );
      end;
    end
    else
      List.Add( ' Даних немає.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries * SizeOf( TMibIPForwardRow ) );
  FreeMem( pBuf );
end;

//-----------------------------------------------------------------------------
procedure Get_IPStatistics( List: TStrings );
var
  IPStats       : TMibIPStats;
  ErrorCode     : integer;
begin
  if not Assigned( List ) then EXIT;
  if NOT LoadIpHlp then exit ;
  ErrorCode := GetIPStatistics( @IPStats );
  if ErrorCode = NO_ERROR then
  begin
    List.Clear;
    with IPStats do
    begin
      if dwForwarding = 1 then
        List.add( ' Розблоковане пересилання      : '  + ' Так'  )
```

```
      else
        List.add( ' Розблоковане пересилання      : '  + ' Hi'  );
      List.add( ' Вбудований TTL             : '  + inttostr( dwDefaultTTL ) );
      List.add( ' Отримані датаграми       : '  + inttostr( dwInReceives ) );
      List.add( ' Помилка заголовку     (в)  : '  + inttostr( dwInHdrErrors ) );
      List.add( ' Адреса помилкова    (в)  : '  + inttostr( dwInAddrErrors ) );
      List.add( ' Датаграма переслана     : '  + inttostr( dwForwDatagrams ) );
//
      List.add( ' Невідомий протокол (в)  : '  + inttostr( dwInUnknownProtos )
);
      List.add( ' Датаграма відвергнута     : '  + inttostr( dwInDiscards ) );
      List.add( ' Датаграма встановлена     : '  + inttostr( dwInDelivers ) );
      List.add( ' Запит виведення           : '  + inttostr( dwOutRequests ) );
      List.add( ' Маршрутизація в маршрутизаторі Cisco відвергнута     : ' +
inttostr( dwRoutingDiscards ) );
      List.add( ' Немає маршрутів        (з)  : '  + inttostr( dwOutNoRoutes )
);
      List.add( ' Час виведення перебору      : '  + inttostr( dwReasmTimeOut )
);
      List.add( ' Запити перебору     : '  + inttostr( dwReasmReqds ) );
      List.add( ' Перебори здійснено : '  + inttostr( dwReasmOKs ) );
      List.add( ' Помилка перебору     : '  + inttostr( dwReasmFails ) );
      List.add( ' Фрагментація мережі успішна: '  + inttostr( dwFragOKs ) );
      List.add( ' Помилка фрагментації   : '  + inttostr( dwFragFails ) );
      List.add( ' Датаграми фрагментовано    : '  + inttostr( dwFRagCreates ) );
      List.add( ' Кількість інтерфейсів    : '  + inttostr( dwNumIf ) );
      List.add( ' Кількість IP-адрес  : '  + inttostr( dwNumAddr ) );
      List.add( ' маршрут в таблиці маршрутизації Cisco : '  + inttostr(
dwNumRoutes ) );
    end;
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
end;

function IpHlpIPStatistics (var IPStats: TMibIPStats): integer ;      //
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetIPStatistics( @IPStats );
end ;

//-------------------------------------------------------------------------
procedure Get_UdpStatistics( List: TStrings );
var
  UdpStats     : TMibUDPStats;
  ErrorCode    : integer;
begin
  if not Assigned( List ) then EXIT;
  ErrorCode := GetUDPStatistics( @UdpStats );
  if ErrorCode = NO_ERROR then
  begin
    List.Clear;
    with UDPStats do
    begin
      List.add( ' Datagrams (в)    : '  + inttostr( dwInDatagrams ) );
      List.add( ' Datagrams (з)   : '  + inttostr( dwOutDatagrams ) );
      List.add( ' No Ports          : '  + inttostr( dwNoPorts ) );
      List.add( ' Errors    (в)    : '  + inttostr( dwInErrors ) );
      List.add( ' UDP Listen Ports  : '  + inttostr( dwNumAddrs ) );
    end;
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
end;
```

```
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
function IpHlpUdpStatistics (UdpStats: TMibUDPStats): integer ;     //
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetUDPStatistics (@UdpStats) ;
end ;



//--------------------------------------------------------------------------------

procedure Get_ICMPStats( ICMPIn, ICMPOut: TStrings );

var
  ErrorCode     : DWORD;
  ICMPStats     : PTMibICMPInfo;

begin
  if ( ICMPIn = nil ) or ( ICMPOut = nil ) then EXIT;
  ICMPIn.Clear;
  ICMPOut.Clear;
  New( ICMPStats );
  ErrorCode := GetICMPStatistics( ICMPStats );
  if ErrorCode = NO_ERROR then
  begin
    with ICMPStats.InStats do
    begin
      ICMPIn.Add( ' Отримано повідослень    : '  + IntToStr( dwMsgs ) );
      ICMPIn.Add( ' Помилки ICMP               : '  + IntToStr( dwErrors ) );
      ICMPIn.Add( ' Розташування    : '  + IntToStr( dwDestUnreachs ) );
      ICMPIn.Add( ' Час перевищено        : '  + IntToStr( dwTimeEcxcds ) );
      ICMPIn.Add( ' Проблеми параметрів      : '  + IntToStr( dwParmProbs ) );
      ICMPIn.Add( ' Джерело відключено        : '  + IntToStr( dwSrcQuenchs ) );
      ICMPIn.Add( ' Перенаправлення            : '  + IntToStr( dwRedirects ) );
      ICMPIn.Add( ' Ехо запит          : '  + IntToStr( dwEchos ) );
      ICMPIn.Add( ' Ехо повтор          : '  + IntToStr( dwEchoReps ) );
      ICMPIn.Add( ' Запит мітки часу    : '  + IntToStr( dwTimeStamps ) );
      ICMPIn.Add( ' Повтор мітки часу    : '  + IntToStr( dwTimeStampReps ) );
      ICMPIn.Add( ' Запит маски адреси : '  + IntToStr( dwAddrMasks ) );
      ICMPIn.Add( ' Повтор маски адреси   : '  + IntToStr( dwAddrReps ) );
    end;
     //


    with ICMPStats.OutStats do


 begin
      ICMPOut.Add( ' Повідомлення відправлено        : '  + IntToStr( dwMsgs )
);
      ICMPOut.Add( 'Помилки ICMP            : '  + IntToStr( dwErrors ) );
      ICMPOut.Add( 'Розташування    : '  + IntToStr( dwDestUnreachs ) );
      ICMPOut.Add( 'Час перевищено        : '  + IntToStr( dwTimeEcxcds ) );
      ICMPOut.Add( 'Проблеми параметрів      : '  + IntToStr( dwParmProbs ) );
      ICMPOut.Add( 'Джерело відключено        : '  + IntToStr( dwSrcQuenchs ) );
      ICMPOut.Add( 'Перенаправлення            : '  + IntToStr( dwRedirects ) );
      ICMPOut.Add( 'Ехо запит          : '  + IntToStr( dwEchos ) );
      ICMPOut.Add( 'Ехо повтор          : '  + IntToStr( dwEchoReps ) );
      ICMPOut.Add( 'Запит мітки часу    : '  + IntToStr( dwTimeStamps ) );
      ICMPOut.Add( 'Повтор мітки часу    : '  + IntToStr( dwTimeStampReps ) );
      ICMPOut.Add( 'Запит маски адреси: '  + IntToStr( dwAddrMasks ) );
      ICMPOut.Add( 'Повтор маски адреси   : '  + IntToStr( dwAddrReps ) );
    end;
  end
  else
    IcmpIn.Add( SysErrorMessage( ErrorCode ) );
  Dispose( ICMPStats );
end;
```

```
//---------------------------------------------------------------------------
procedure Get_RecentDestIPs( List: TStrings );
begin
  if Assigned( List ) then
    List.Assign( RecentIPs )
end;

initialization

  RecentIPs := TStringList.Create;

finalization

  RecentIPs.Free;

end.
```

Файл NetConst.pas – ініціалізація констант

```
unit NetConst;


interface

resourcestring

  SShellLinkReadError = ' Помилка  читання' ;
  SShellLinkWriteError = ' Помилка запису' ;
  SShellLinkLoadError = ' Не можу завантажити %s' ;
  SShellLinkSaveError = ' Не можу зберегти %s' ;
  SShellLinkCreateError = ' Не можу ініціалізувати інтерфейс' ;
  SDynArrayIndexError = ' У масиві %s немає елемента з таким індексом (%d)' ;
  SDynArrayCountError = ' Перевищена довжина масиву (%d)' ;
  SSharedMemoryError = ' Не можу створити файл' ;
  SCannotInitTimer = ' Не можу ініціалізувати таймер' ;
  SPrinterIndexError = ' Принтер не доступний (%d)' ;
  SIndicesOutOfRange = ' Недопустимий індекс матриці [%d: %d]' ;
  SRowIndexOutOfRange = ' Недопустиме занчення рядка матриці (%d)' ;
  SColIndexOutOfRange = ' Недопустиме занчення стовбчика матриці (%d)' ;
  SNoAdminRights = ' У Вас немає прав адміністратора' ;

  SFileError = ' Помилка %s файл %s%s' ;
  SFileReading = ' читання' ;
  SFileWriting = ' запис' ;

 SFileError002 = '  - файл не знайдено' ;
  SFileError003 = '  - шлях не знайдено' ;
  SFileError004 = '  - не можу відкрити файл' ;
  SFileError005 = '  - немає доступу' ;
  SFileError014 = '  - не достатньо пам"яті' ;
  SFileError015 = '  - не можу знайти драйвер' ;
  SFileError017 = '  - не можу перемістити файл' ;
  SFileError019 = '  - носій захищений від запису' ;
  SFileError020 = '  - не можу знайти пристрій' ;
  SFileError021 = '  - пристрій не відкривається для читання' ;
  SFileError022 = '  - пристрій не може розпізнати команду' ;
  SFileError025 = '  - вказана область не знайдена' ;
  SFileError026 = '  - пристрій недоступний' ;
  SFileError027 = '  - сектор не знайдено' ;
  SFileError029 = '  - помилка запису на пристрій' ;
  SFileError030 = '  - помилка читання з пристрою' ;
  SFileError032 = '  - файл використовується іншою програмою' ;
  SFileError036 = '  - занадто багато відкритих файлів' ;
  SFileError038 = '  - досягнутий кінець файлу' ;
  SFileError039 = '  - диск переповнений' ;
  SFileError050 = '  - запит не підтримується' ;
  SFileError051 = '  - віддалений комп' ютер недоступний' ;
  SFileError052 = '  - у мережі знайдені ідентичні імена' ;
  SFileError053 = '  - мережний шлях не знайдено' ;
  SFileError054 = '  – мережа зайнята' ;
  SFileError055 = '  - ресурс мережі або пристрій недоступний' ;
  SFileError057 = '  - апаратна помилка в мережному адаптері' ;
  SFileError058 = '  - сервер не в змозі виконувати дану дію' ;
  SFileError059 = '  - помилка у мережі' ;
  SFileError064 = '  - недоступне мережне ім"я' ;
  SFileError065 = '  - немає доступу до мережі' ;
  SFileError066 = '  - невірно вказатий тип мережного ресурсу' ;
  SFileError067 = '  - не знайдене вказане мережне ім"я' ;
  SFileError070 = '  - відключейний сервер мережі' ;
  SFileError082 = '  - не можу створити файл чи каталог' ;
  SFileError112 = '  - не достатньо вільного місця на диску' ;
  SFileError123 = '  - в імені файлу вказано недопустимий символ' ;
  SFileError161 = '  - неправильно вказано шдях' ;
```

```
  SFileError183 = ‘  - файл не існує’ ;

  SCannotSetSize = ‘ Не можу змінити розмір файлу’ ;

  SUnableToCompress = ‘ Не можу заархівувати дані’ ;
  SUnableToDecompress = ‘ Не можу розархівувати дані’ ;

  SCannotFindNetwork = ‘ Не можу знайти  мережу’ ;

implementation

end.
```

Файл About.pas - довідка

```
unit About;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls;

type
  TForm2 = class(TForm)
    Image1: TImage;
    Memo1: TMemo;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.Button1Click(Sender: TObject);
begin
Form2.Close;
end;

end.
```