**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Центральноукраїнський національний технічний університет**

**Кафедра кібербезпеки та програмного забезпечення**

На правах рукопису

Ситнік Євген Юрійович

**Програмне забезпечення системи Network Function Virtualization для операторів зв'язку**

Спеціальність: 123 «Комп'ютерна інженерія»

Освітній ступінь: бакалавр

Науковий керівник:

**Дрєєв Олександр Миколайович** _____

(підпис) (дата)

кандидат технічних наук, доцент

**ДОПУЩЕНО ДО ЗАХИСТУ**

**Завідувач кафедри**

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.



Кропивницький – 2021

**Міністерство освіти і науки України**
**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**
**д.т.н., проф.**
**_____О.А.Смірнов**
« 11 » січня 2021 року

# З А В Д А Н Н Я
## НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

*Ситніку Євгену Юрійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи Network Function Virtualization для операторів зв'язку*

керівник роботи *Дрєєв Олександр Миколайович, канд. техн. наук, доцент*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 204-02 від 28.12.2020 року

2. Строк подання студентом роботи до захисту *22.05.2021 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи Network Function Virtualization для операторів зв'язку*

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

*1. Призначення та область використання.*
*2.Перегляд аналогічних існуючих систем.*
*3. Опис і обґрунтування проектних рішень.*
*4. Етапи програмування системи.*
*5. Впровадження системи в промислову експлуатацію.*
*6. Висновки*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

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

6. Дата видачі завдання « 11 » січня 2021 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної бакалаврської роботи | Строк виконання етапів кваліфікаційної бакалаврської роботи | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.03.2021 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.03.2021 р. | |
| 3. | Розробка моделі компонента | 20.03.2021 р. | |
| 4. | Розробка структур даних | 25.03.2021 р. | |
| 5. | Розробка алгоритмів зв’язкуу та відображення | 30.03.2021 р. | |
| 6. | Програмування алгоритмів | 10.04.2021 р. | |
| 7. | Оформлення ПЗ | 17.04.2021 р. | |
| 8. | Попередній захист роботи | 14.05.2021 р. | |
| | | | |
| | | | |
| | | | |
| | | | |

**Студент** _____ _____
( підпис ) (прізвище та ініціали)

**Керівник роботи** _____ _____
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

**Ситнік Є.Ю. Програмне забезпечення системи Network Function Virtualization для операторів зв'язку. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.**

В даній кваліфікаційній бакалаврській розроблено програмне забезпечення, яке призначено для системи Network Function Virtualization для операторів зв'язку.

Метою розробки є програмне забезпечення системи Network Function Virtualization для операторів зв'язку.

Результат роботи – програмна реалізація системи Network Function Virtualization для операторів зв'язку.

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 Delphi 10.4.1.

**Ключові слова**: комп'ютерна інженерія, Network Function Virtualization

# ABSTRACT

**Sytnik Ye.Yu. Network Function Virtualization software for telecom operators. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021**

This bachelor's degree software has been developed for Network Function Virtualization for telecom operators.

The purpose of the development is Network Function Virtualization software for telecom operators.

The result is a software implementation of the Network Function Virtualization system for telecom operators.

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 Delphi 10.4.1 environment.

**Keywords:** computer engineering, Network Function Virtualization

**ЗМІСТ**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ ..... 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ ..... 5
1.1 Призначення системи..... 5
1.2 Область застосування ..... 6
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ ..... 8
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми кваліфікаційної бакалаврської роботи..... 8
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування ..... 19
2.3 Розгорнута постановка завдання ..... 25
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ ..... 26
3.1 Опис функціонування системи ..... 26
3.2 Розробка структурної схеми..... 35
3.3 Розробка функціональної схеми ..... 39
3.4 Розробка діаграми процесів ..... 51
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ .... 54
4.1 Розробка блок-схем та опис алгоритмів функціонування системи ..... 54
4.2 Захист розробленого програмного забезпечення..... 68
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 70
6 ОСНОВНІ ВИСНОВКИ ..... 72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 74

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

| | | |
|---|---|---|
| ЦОД | – | Центр обробки даних |
| EMS | – | Система керування й адміністрування |
| KVM | – | Віртуалізація серверів |
| MANO | – | Management and orchestration |
| NFV | – | Network Function Virtualization, віртуалізація мережевих функцій |
| NFVI | – | Інфраструктура NFV, network function virtualization infrastructure |
| OSS/BSS | – | Стандартні системи операторів зв’язку |
| OSM | – | Open Source MANO |
| SBC | – | Прикордонні контролери сесій |
| SDN | – | Програмно-обумовлені мережі |
| VNF | – | Віртуальна мережева функція |

# ВСТУП

**Актуальність теми**. Мережі операторів постійно поповнюються новими пристроями. Нові сервіси ведуть до закупівлі нових пристроїв, пошуку місця для розміщення й додаткових потужностей по електроживленню. Власний пропрієтарний інтерфейс для кожного пристрою, проблеми інтеграції із уже існуючим залізом, веде до пошуку нових або навчанню наявних IT-фахівців. Це веде до росту витрат і збільшенню часу підключення й активації нових сервісів для всеїдних користувачів.

Network Function Virtualization (NFV) – віртуалізація мережевих функцій, яка дозволить переглянути підходи до створення архітектури мережі й дозволить перенести мережеві функції у віртуальний простір на базі стандартних серверів, розміщених у центрах обробки даних, мережевих вузлах або офісах корпоративних клієнтів, а також удвічі скоротити споживання електроенергії.

**Мета й завдання дослідження.** Метою роботи є програмне забезпечення системи Network Function Virtualization для операторів зв'язку.

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

– Огляд існуючих систем Network Function Virtualization для операторів зв'язку.

– Дослідження системи Network Function Virtualization для операторів зв'язку.

– Програмна реалізація системи Network Function Virtualization для операторів зв'язку.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі Network Function Virtualization для операторів зв’язку.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи Network Function Virtualization для операторів зв’язку, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Економічний висновок для телеком-індустрії наступний: існує вимога до підвищення інвестицій у телеком-інфраструктуру й це пов'язане з вимогами до більшої пропускної здатності каналів зв’язку. Одночасно є тенденція до скорочення середнього виторгу на абонента за рахунок розвитку конкуренції й демпінг із боку великих операторів у прагненні удержати частку ринку. Ці два збіжні графіки – витрати на інвестиції й зменшення доходів рано або пізно перетнуться, що приведе до банкрутств серед телеком-операторів і переділу ринку убік укрупнення.

Для нівелювання цієї погрози, великі телеком-гіганти повинні з однієї сторони підвищувати віддачу від своїх інвестицій, і з іншої сторони створювати нові сервіси для підвищення виторгу з одного абонента. Обидві завдання можна виконати за рахунок віртуалізації мережевих функцій. Більш щільне використання апаратних ресурсів при віртуалізації показало свій успіх на ринку обчислювальних потужностей – кілька віртуальних машин на одному фізичному сервері утилізують CPU, RAM і Storage набагато ефективніше. Запуск декількох віртуальних мережевих функцій на одному апаратному сервері теж знизить капітальні витрати на устаткування. Крім цього, дасть гнучкість у керуванні, дозволяючи швидше робіти відновлення ПЗ, усувати баги й запускати нові сервіси в короткий термін.

Для досягнення цих цілей потрібна система керуванням інфраструктурою віртуальних мережевих функцій – NFVI (network function virtualization infrastructure). Ця система повинна вміти в автоматизованому режимі запускати мережеві функції, відслідковувати їхній стан, масштабувати у випадку збільшення навантаження, перезапускати при аварійній поведінці, термінувати у

випадку відсутності необхідності. Усе це лягло в основу того, що назвали MANO – management and orchestration.

### 1.2 Область застосування

Великі гравці телеком-індустрії, усвідомлюючи неминучість руху в цьому напрямку, почали координувати свої зусилля! Це неймовірно в епоху капіталізму й закритості бізнес-моделей, які були присутні в 20 столітті, але видиме неминуче в 21 столітті, у якому темп змін росте експоненційно.

Почалося все з того, що учасники телеком-ринку стали створювати стандарти, по яких повинні реалізуватися програмні компоненти цієї індустрії. Під егідою європейського інституту стандартизації в телеком-сфері ETSI була створена робоча група для створення цілого списку стандартів, що описують самі різні процеси цієї ідеї.

Представництво компаній-контриб'юторів, які вносять свій внесок у створення стандартів, дуже значне. Подивитеся на список учасників стандарту ETSI MANO:

NEC, Tech Mahindra, Orange, Cisco, Amdocs, Wind River Systems, Alcatel-Lucent, HP, Oracle, Orange, Juniper, ZTE, Intel, Telecom Italia, Wipro, EnterpriseWEB, NSN, Huawei, ASSIA, Telefonica, China Unicom, AT&T, Verizon, Docomo, Deutsche Telekom, Sprint, Vodafone, 6Wind, BT, tech Mahindra, Sonus, Brocade, ETRI, Ooredoo, Fujitsu, Ericsson.

Отут і вендори, і телеком-оператори. При такому солідному составі очевидно, що прийняті стандарти будуть імплементовані як у програмних компонентах від вендорів, так і на мережах телеком-операторів. Варто помітити, що робота над стандартами ще триває, тому що при реалізації закладених підходів відбувається уточнення функціонала, формату даних, стикувальних інтерфейсів, процесів і іншого. Кожна компанія-контриб'ютор має право (і активно ним користується!) на внесення доробок у стандарт. Це допомагає

виробникам ПЗ додавати ті функції, які вони знають як реалізувати у вихідному коді. І це допомагає операторам, які знають реальні бізнес-кейси і як вони повинні бути вирішені за допомогою розроблювального ПЗ.

У підсумку була створена референсна архітектура компонентів MANO.

Нас же, у рамках цього роботи, цікавить, які ініціативи дали свій результат в open source середовищу. Комерційні пропрієтарні розробки вендорів, які заробляють на продажі ПЗ й ліцензій до них, складно оцінити поглядом з боку, не перебуваючи усередині ринку. При цьому опен-сорс проекти активно обговорюються і їх можна вивчати, не будучи вендором.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи Network Function Virtualization для операторів зв'язку, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

**ПЗ HPE Helion Openstack Carrier Grade для NFV**

Ви надаєте послуги зв'язку й прагнете створити ефективний ЦОД з використанням Openstack®? ПЗ HPE Helion Openstack Carrier Grade для NFV – це інструмент Openstack для ефективного розподілу й віртуалізації обчислювальних і мережевих ресурсів (NFV), що дозволяє постачальникам послуг зв’язку розгортати застосунки NFV на платформах з відкритим вихідним кодом. Платформа класу Carrier Grade вимагає інтеграції численних базових технологій, які винесені за рамки проекту Openstack. Серед основних прикладів можна назвати операційну систему хоста (Linux®), віртуалізацію серверів (KVM), віртуалізацію мережі (vSwitch) і контролери SDN. В основі застосунку лежать базові принципи HPE Helion Openstack Enterprise Edition, поліпшене по трьом ключовим напрямкам і функціям Carrier Grade: керованість, доступність і продуктивність.

**Основні характеристики**

**Відкрита інфраструктура, висока доступність, виняткова швидкість передачі даних**

HPE Helion Openstack Carrier Grade пропонує постачальникам послуг зв’язку хмарну платформу з відкритим вихідним кодом, яка забезпечує високу надійність і пропонує розширені функції самовідновлення для скорочення простоїв системи. Захист від виникнення єдиної крапки відмови. Максимальна доступність і надійність.

Керування життєвим циклом програмного забезпечення спрощує установку й керування в HPE Helion Openstack Carrier Grade. Резервування й швидке відновлення після збоїв системи керування.

Збільшуйте пропускну здатність за допомогою віртуального комутатора із прискорювачем DPDK, SR-IOV наскрізних рознімань PCI.

**Платформа для високопродуктивних віртуальних мережевих функцій (VNF)**

HPE Helion Openstack Carrier Grade гарантує близьку до номінальної швидкість передачі даних у мережі для VNF, включаючи vepc, vbras, vims, vfw і vrouter.

Висока продуктивність фізичних мережевих компонентів дозволяє скористатися всіма перевагами віртуалізації без шкоди для швидкодії.

**Функції керування для підвищення безпеки, керування програмними компонентами, планування ресурсів і адміністрування**

HPE Helion Openstack Carrier Grade підтримує установку відновлень без переривання роботи системи для зменшення часу планових простоїв.

Більш надійний захист завдяки різним поліпшенням, пов'язаним з ідентифікацією й керуванням доступом на рівні Openstack Keystone, посиленням мережевої безпеки й безпеки ОС, інструментами шифрування, аудитом, веденням журналів і установкою виправлень.

Керування парком устаткування через докладну інформаційну панель, що відбиває обчислювальні ресурси, системи зберігання, мережеві ресурси в хмарі. Динамічне відновлення списків активних попереджень за допомогою Openstack Horizon. Інтеграція з інструментами сторонніх виробників для керування парком устаткування.

Додаткова гнучкість завдяки розширеним ресурсам планування й адміністрування.

**Висока доступність і аварійне відновлення**

HPE Helion Openstack Carrier Grade підтримує прив'язку мережевих контролерів на рівні сервера. Це дозволяє використовувати всю доступну смугу пропущення й забезпечує захист від збоїв.

Інструменти високої доступності для екземплярів KVM, включаючи виявлення збоїв на обчислювальних вузлах, автоматичну евакуацію в просторі KVM і міграцію в реальному часі в простір KVM для досягнення високої доступності VNF.

**ПЗ HPE SBC Global Traffic**

Оптовий застосунок Session Border Controller припускає підключення прикордонних контролерів сесій (SBC) оптового оператора до оптових клієнтів за допомогою мереж IP. Кожному оптовому клієнтові призначають два SBC: основний і допоміжний. Перевірку стану й захист від збоїв повинні забезпечувати самі клієнти. Найчастіше при виявленні збою замість переходу на допоміжний SBC клієнти віддають перевагу альтернативного провайдера. Це приводить до втрати прибутки оптовим оператором. Крім того, оскільки для підтримки кожного оптового клієнта можна використовувати тільки один SBC, виникають серйозні обмеження по обсягах обслуговування кожного окремого клієнта. HPE SBC Global Traffic Manager дозволяє впоратися із цими складностями за рахунок впровадження розподіленого модуля балансування навантаження SBC, до якого підключаються оптові клієнти. HPE SBC Load Balancer – це повністю резервуємий застосунок рівня телекомунікаційних компаній, яке виділяє оптовим клієнтам віртуальні IP-адреси й розподіляє навантаження між декількома SBC згідно з політиками оператора.

**Основні характеристики**

**Автоматичний розподіл навантаження SBC для оптових клієнтів**

Модульна структура.

Просте й швидке додавання нових оптових клієнтів.

Просте додавання потужностей.

Спрощує перехід до віртуалізації.

**Висока доступність SBC**

Автоматична перевірка стану з використанням скриптів.

Швидке відновлення обслуговування.

Повне бачення системи для оптових клієнтів.

Застосунок забезпечує високу доступність і географічно розподілене резервування.

**Перевірені конфігурації віртуалізації мережевих функцій (NFV) HPE**

Прагнете впровадити застосунок віртуалізації мережевих функцій (NFV) HPE у вашу робочу мережу? Більшість постачальників послуг зв'язку (CSP) потребує технології, яка легко впроваджується, є високодоступною і підтримує керування життєвим циклом. Перевірені конфігурації віртуалізації мережевих функцій (NFV) HPE надають постачальникам послуг зв'язку надійну попередньо інтегровану платформу NFV, для якої доступна комплексна підтримка. Перевірені конфігурації NFV підготовлені разом із замовниками з обліком їх вибору апаратних і програмних компонентів сторонніх постачальників, опираючись на великі знання й передовий досвід HPE в області віртуалізації мережевих функцій. Перевірені конфігурації NFV забезпечують гнучкість у прийнятті рішень, прискорення процесу розгортання, зменшення ризиків і скорочення строків окупності, а також кваліфіковану технічну підтримку застосунків NFV з єдиним контактною особою. Такі конфігурації ідеально походять постачальникам послуг зв'язку, які вже вибрали певні сервери, програмне забезпечення й мережеві технології, але як і раніше прагнуть зробіти процес розгортання NFV швидким і безпроблемним.

Характеристики:

– Перевірена архітектура для телекомунікаційної хмари віртуалізації мережевих функцій (NFV) розроблялася разом з постачальниками послуг зв'язку й опирається на досвід HPE в області систем NFV і проектів підтвердження

концепцій NFV. При цьому використовуються кращі в галузі сервери, системи зберігання, мережеві й програмні продукти.

– Попередньо інтегровані й перевірені застосунки зі скриптами автоматизації для скорочення часу складання, спрощена специфікація матеріалів для простого замовлення відповідно до необхідного масштабу й попередньою конфігурацією, що відповідає вимогам замовника.

– Висококваліфікована технічна підтримка, у тому числі продуктів сторонніх постачальників. Кожний обіг розглядається індивідуально й надається безпосередній доступ до спеціалізованої групи підтримки HPE NFV для одержання обслуговування найвищого класу.

– Можливість поєднувати кращі в галузі продукти HPE з обраними компонентами сторонніх постачальників.

– Можна вибрати засіб керування віртуалізованою інфраструктурою (VIM): Helion Openstack Carrier Grade 4.0 або Red Hat Openstack Platform 10.

**Телекомунікаційні проекти HPE**

Схеми віртуалізації мережевих функцій (NFV) являють собою перевірені еталонні конфігурації на основі інфраструктурних платформ HPE, партнерських менеджерів віртуальної інфраструктури й програмно обумовлених мереж, адаптованих для конкретних сценаріїв використання NFV. Вони містять у собі набори інструментів і документацію, що спрощують установку й створення стека інфраструктури NFV (NFVI). Фахівці HPE підкреслюють важливість використання відкритих застосунків NFV-I, що полягають із компонентів декількох постачальників і підтримуючих галузеві стандарти.

**Основні характеристики**

**Поліпшення для кожного рівня стека інфраструктури NFV**

Масштабованість на основі модульності.

Надійність без єдиної крапки відмови.

Підвищена продуктивність

**Кращі методики розгортання HPE NFV**

Автоматизоване розгортання й налаштування конфігурації стека NFVI допомагають знизити складність і виключити людський фактор.

Реалізація принципів керування інфраструктурою за допомогою формальної моделі ("інфраструктура як код").

Архітектура, що забезпечує високу доступність телекомунікаційних застосунків.

**Комплексна продуктивність операторського класу**

Усе компоненти системи сертифіковані відповідно до вимог NEBS 3-го рівня.

Спеціальне налаштування з оптимальними конфігураціями BIOS і мережевого адаптера.

Визначення характеристик продуктивності систем зберігання й мережевих адаптерів.

**Підтримка й компонента різних постачальників для керування відкритою платформою**

Розроблене й протестоване в тісному технічному співробітництві із провідними партнерами в області VIM і SDN.

Підтримка широкої екосистемою системних інтеграторів і технологічних партнерів HPE.

Використовуються відкриті галузеві стандарти, такі як Redfish API.

**Система віртуалізації мережевих функцій (NFV) HPE**

Потрібний застосунок з повною віртуалізацією мережевих функцій? Система HPE NFV – це готова платформа інфраструктури віртуалізації мережевих функцій, яка дозволяє вибрати ПЗ для керування віртуальною інфраструктурою (HPE Helion Openstack Carrier Grade або Red Hat Openstack Platform) і систему зберігання (CEPH, HPE VSA, HPE 3PAR). Система NFV надає готові, інтегровані й перевірені застосунки для швидкого й простого розгортання з віртуалізацією мережевих функцій. Вони створені, щоб допомогти

постачальникам послуг зв’язку й мережевого устаткування перейти від експериментів і пробних версій до розгортання робочих застосунків з повною віртуалізацією мережевих функцій. Постачальники послуг можуть вибрати систему NFV потрібного масштабу, що включає в себе комплект повністю перевіреного й інтегрованого програмного й апаратного забезпечення. Це дозволяє їм відразу приступитися до роботи й розширюватися надалі в міру необхідності. Настроєні комплекти сумісні із продуктами сторонніх постачальників віртуалізованих мережевих служб (у рамках партнерських програм HPE OpenNFV і Red Hat) і надають широкі можливості, які опираються на інновації й перевірені застосунки.

**Основні характеристики**

**Простота розгортання**

Система HPE Network Functions Virtualization (NFV) спрощує розгортання інфраструктури NFV для постачальників послуг зв’язку, забезпечуючи зручність придбання, установки, експлуатації й підтримки на основі пакетів NFV і номерів деталей.

Він надає готові до використання конфігурації для прискорення розгортання в центрі обробки даних за допомогою погоджених засобів керування.

Програма – майстер установки підтримує установку всього програмного забезпечення.

Ви одержите готові автоматизовані служби для спрощеної реалізації різних процесів.

Код продукту NFV для технічної підтримки – це вихідний елемент для початку обслуговування системи.

**Архітектура з відкритим кодом**

Система HPE Network Functions Virtualization (NFV) забезпечує відкритість, тому клієнти часто вибирають її. Кращі у своєму класі застосунку для різних функцій дозволяють розгортати середовища, за допомогою яких постачальники послуг зв’язку можуть досягтися найкращих бізнес-результатів.

Застосунок, що використовує платформу з відкритим кодом, Carrier Grade заснований на Linux® Openstack і реалізації KVM, застосовуються додаткові компоненти з відкритим вихідним кодом.

Надає API-інтерфейси Openstack – запобігання прив'язки до одного постачальника, характерні для комерційних застосунків.

Підтримка декількох гостьових операційних систем для підвищення гнучкості.

Використовуйте партнерську програму OpenNFV або Red Hat для оперативного створення інноваційних послуг.

**Carrier Grade**

Система HPE Network Functions Virtualization (NFV) забезпечує високу доступність усіх апаратних і програмних компонентів, щоб виключити поява єдиних крапок відмови.

Надійні апаратні застосунки від Hewlett Packard Enterprise: сімейства серверів, систем зберігання й мережевих продуктів.

Поліпшений розподіл Openstack згідно з вимогами постачальників послуг.

Скорочення простоїв завдяки пакетній обробці даних віртуальних мереж у телекомунікаційних середовищах у режимі реального часу.

**Нові джерела доходу**

Система HPE Network Functions Virtualization (NFV) дозволяє створювати інноваційні моделі послуг з безліччю HPE OpenNFV і інших телекомунікаційних робочих навантажень Telco, а також одержувати доступ до технічного обслуговування, технічної підтримки й консультаціям HPE.

Підтримка використання постачальниками послуг зв’язку нових технологій, таких як «Інтернет речей».

Використовується в якості оптимальної платформи для різних застосунків віртуалізації мережевих функцій, наприклад vCPE, vEPC й vIMS.

**Open Source MANO**

Open Source MANO (OSM) розробляється робочою групою під керівництвом самого ETSI. При цьому лідируючу роль займають представники операторів Telefonica, BT, Telenor і вендорів Intel, RITF.io, Ubuntu, Vmware.

Архітектура OSM уже зараз дозволяє робіти дизайн мережевих сервісів, виконувати оркестрацію мережевих сервісів, управляти окремими VNF і управляти ресурсами NFVI, виділюваних під VNF. Доступне підключення SDN-контролерів для керування мережевим компонентом, необхідним для керування трафіком, який буде надходити в VNF.

Тому що цей проект просувається самим ETSI, тобто ймовірність, що він буде одним із самих успішних.

**Open-O**

Даний проект є китайською відповіддю на тренд по створенню системи оркестрації мережевих функцій. Список контриб'юторів говорить сам за себе: China Mobile, China Telecom, Huawei, ZTE, Hong Kong Telecom, Ericsson, Intel, Gigaspaces, Canonical, Infoblox, Redhat. І наявність у цьому списку західних вендорів пускай вас не бентежить – лідерами там є саме китайські компанії.

Перший реліз уже відбувся й доступний для завантаження.

Архітектура дозволяє створювати вокрфлоу сервісу за допомогою стандарту TOSCA, управляти мережами за допомогою SDN-оркестратора й мережевими функціями за допомогою NFV-оркестратора. Доступні коннектори до сторонніх SDN-контролерам і системами керування NFVI через VIM.

Очевидно, що китайський ринок телеком-операторів буде орієнтуватися на цей продукт і чи навряд ми побачимо розквіт інших ініціатив на цьому азіатському напрямку. А з обліком того, що китайський ринок мобільного й фіксованого зв'язку є самим великим у світі, те очевидно, що багато застосунків по керуванню гігантськими потоками даних саме на цьому ринку можуть показати себе із кращої сторони. Гігантські сплески трафіку й робота з Великими

Даними від великої кількості передплатників дозволить протестувати інноваційні підходи саме на цьому ринку.

**SONATA**

Цей проект з'явився за рахунок європейського гранту на НІОКР в області мереж 5G за назвою Horizon 2020 – ініціатива, при якій до 2020 року повинні з'явиться мережі 5G, які дозволять підвищити швидкість передачі даних для мобільних абонентів. При цьому ті сервіси, які будуть виявлятися передплатникам повинні мати маленьку затримку й широкі можливості. Частиною застосунку цього завдання є створення системи оркестрації мереж зв'язку й мережевих функцій. Проект SONATA повинен був надати можливість дизайну мережевих функцій і оперативне керування створеними мережевими функціями.

У партнерах проекту зазначені наступні компанії: Atos, NEC, Thales, Telefonica, Nokia, BT і кілька європейських університетів. Сам проект має дуже академічну спрямованість і глибоке пророблення з погляду технологій. Націленість у пріоритеті на обкатування підходів нового технологічного тренда, а не якнайшвидшу комерційну експлуатацію.

Перший реліз уже доступний для завантаження, хоча по даним github'a видно, що в поточну першу версію триває контриб’юція коду й відповідно не можна назвати цей реліз стабільним.

Функціонал представлений наступними компонентами:

SONATA SDK дозволяє робіти дизайн сервісів і тестування. SONATA Service Platform дозволяє запускати створені сервіси й управляти їхнім життєвим циклом.

**Що робіти телеком-операторам зараз?**

Що вже зараз можуть робіти оператори зв'язку, які прагнуть підготуватися до наступного технологічного тренда? Великі західні й азіатські телеком-оператори вже завершують стадії пілотування й proof of concept і переходять на поступове впровадження даних застосунків на продуктивний трафік. Нові

ділянки мережі оснащуються архітектурою віртуалізованих мережевих функцій. Нові сервіси проектують із використанням agile-підходу й на основі розподіленої інфраструктури. В Україні оператори найчастіше орієнтуються на застосунки, які поставляють західні або азіатські вендори, що пройшли обкатування на вже розгорнутих мережах. Очолити цей тренд і почати впроваджувати інновації одними з перших на нашому ринку – це прерогатива великих гравців, але ті, хто першим скористається новою хвилею, одержать максимум віддачі за рахунок раннього навчання персоналу й пошуку оптимального застосунку для своїх потреб. Почати можна з пілотування тих open source проектів, які представлені в огляді. Після успішного досвіду на відкритих і безкоштовних проектах, можна буде зробіти вибір комерційного пропрієтарного продукту, з обліком того, що найчастіше телеком-оператори концентрують свою експертизу на операційних процесах, а розробку й підтримку застосунку залишають за перевіреним вендором. Кого вибрати з вендорів, кожний оператор розв'яже сам, але такі звіти як Magic Quadrant by Garnter часто є тем вектором уваги, на який орієнтуються багато споживачів застосунків від вендорів. Недавній рейтинг від Gartner в області OSS систем показує, хто поставляє самі просунуті комерційні системи в цій галузі. Слід помітити, що під парасолькою OSS саме й розташовуються застосунки в області керування NFV-інфраструктурою й екосистемою VNF:

Видне, що у звіті Gartner крім «залізних» вендорів Nokia, Huawei, HPE, IBM і Ericsson так само присутні чисто софтварні компанії – Netcracker, Amdocs, Oracle. При виборі свого вендора слід мати у виді зацікавленість «залізних» компаній у попутному продажі своїх апаратних пристроїв у комплексі із програмними продуктами по керуванню «залізом». Софтварні компанії позбавлені цього недоліку й частіше інших є мульти-вендорними стосовно «заліза».

При цьому, можна вибрати на довгострокову перспективу й open source продукт, але потрібно продумати на обрії 3-5-7 років, хто буде надавати підтримку й писати патчи для багфікса.

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

**Delphi 10.4 Sydney**

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WEBView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент TWEBbrowser для iOS тепер реалізований на WkWEBview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними WEB-технологіями від Microsoft – новим WEBView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні

поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

**Істотне поліпшення Delphi Code Insight**

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

**Delphi Custom Managed Records**

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

**Єдине керування пам'яттю**

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь -якого heap-allocated класу C++, що значно знижує складність коду.

**Розширена підтримка бібліотек C++**

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

**Win 64-відладник і збирач для C++**

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш

стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

**Підвищення якості й швидкодії інструментів**

– Велика кількість поліпшень STL від Dinkumware.

– Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.

– Поліпшена підтримка Cmake.

– Велика кількість виправлень для підвищення стабільності і якості.

– Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

**Змінені стилі VCL для High DPI**

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

**Нові High DPI стилі й стилізація окремих VCL компонент**

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє

використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

**Поліпшена кроссплатформеність**

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент TWEBbrowser для iOS тепер реалізований на WkWEBview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

**Оновлений менеджер пакетів Getit**

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

**Універсальний інсталятор для установки Online і Offline**

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мів програмування й цільових платформ, мів інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

## 2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи Network Function Virtualization для операторів зв'язку.

В процесі розробки кваліфікаційної бакалаврської роботи необхідно виконати наступний обсяг роботи:

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

Якщо ми необхідне нам застосунок відв'яжемо від устаткування й змусимо працювати в ізольованому програмному контейнері замість «заліза», ми зможемо назвати цей застосунок віртуалізованим. Ідеї віртуалізації супроводжують It-індустрію с моменту появи перших ЕОМ – з 60их років минулого століття інженери вирішували завдання поділу одного великого мейнфрейма на кілька ізольованих проектів. В «нульових» людство знову зіштовхнулося зі схожим завданням – цього разу знадобилося ефективно розділяти серверні ресурси між безліччю їх споживачів-клієнтів. Крім самої можливості «нарізати» апаратний ресурс, віртуалізація несе колосальні вигоди.

Співіснування декількох застосунків на загальному залізі вигідніше, чим робота їх на виділених серверах – віртуалізованому застосунку для такої ж продуктивності потрібно менше серверів; такий застосунок більш компактний, витрачає менше електроенергії, вимагає менше мережевих портів і з'єднань. Нові способи доставки застосунків у вигляді шаблону віртуальної машини дозволяють покупцеві відмовитися від раніше апаратного сервера, що нав'язує, але при цьому одержувати всі переваги «коробкового» застосунку. Можливості сучасних гіпервізорів і систем керування привнесли багато нового в плані масштабування, відказостійкості, різних оптимізацій. Віртуалізація переформатувала IT-ринок: сформувалося поняття «хмарних» сервісів різних моделей – SaaS, IaaS, PaaS. Однак, закономірності програмно-прикладної сфери не повною мірою застосовні до сфери мережевої.

**Проблема, яку повинна вирішувати NFV-віртуалізація**

Сучасні телекомунікаційні мережі містять велику кількість фірмового устаткування, як правило, дуже спеціалізованого, заточеного під конкретну

операцію: один пристрій забезпечує NAT, інше обмежує швидкість доступу й уважає трафік, третє здійснює батьківський контроль і фільтрацію вмісту, ще одне відповідає за функції фаєрвола. Для такого поділу функцій між мережевою апаратурою є кілька причин: по-перше, кожний мережевий вендор, так уже зложилося, спеціалізується на декількох функціях, у яких дана конкретна компанія має свої наробітки й ноу-хау. Інша ймовірна причина зводиться до дуже специфічної начинки таких пристроїв, що має мало загального із серверами загального призначення. Тому, оператор-експлуатант таких застосунків мимоволі змушено за всяку ціну підтримувати таку «різноманітність». Запуск нової послуги вимагає установки нового комплекту устаткування, що підтримує необхідну функцію. Це, у свою чергу, вимагає виділення додаткових площ, додаткового електроживлення, логістики «заліза», його монтажу й пусконалаштовування.

Зараз очевидно, що мережа з фізичних «коробок», кожна з яких виконує лише одну-єдину функцію, це не сама оптимальна модель для розвитку. Крім високих капітальних і операційних витрат, такий дизайн дуже інертний, не дозволяє операторові нарощувати портфель надаваних послуг так швидко, як того вимагають акціонери. Мережа повинна бути набагато більш гнучкої й динамічної, сприяти впровадженню будь-якого нового сервісу, його швидкої активації по запиту абонента й звільненню зайнятих ресурсів при деактивації послуги.

**Приклади мережевих функцій, які можливо віртуалізувати, а які не дуже**

Функції, які виконують мережеві пристрої, значно різняться своїми характеристиками й можливостями по їхній віртуалізації. Деякі доцільно перенести в хмару, а інші – категорично не можна відокремлювати від мережевої апаратури. Передача даних так чи інакше пов'язана з переміщенням мережевих пакетів з однієї географічної крапки в іншу, і в кожній із цих крапок, на кожному

вузлі зв'язку, повинне, як мінімум, бути присутнім реальний мережевий пристрій з потрібною кількістю мережевих портів – комутатор або маршрутизатор.

**Пересилання – forwarding/routing/switching** – це сама примітивна операція над мережевим пакетом, яку виконує мережевий пристрій. По суті – копіювання з одного порту на інший, іноді із заміною певних полів фрейма або пакета. Дана операція на більшості платформ виконується спеціальним комплексом, «заточеним» під швидкий пошук відповідності в спеціальній, троїчній пам'яті TCAM і заміну потрібних полів пакета «на л ьоту». Спеціалізований ASIC, використовуваний у маршрутизаторах і комутаторах (його ще називають Network Processor / Network Processing Unit) для таких примітивних завдань більш пристосований – працює швидше, витрачає менше енергії, примітивний, компактний і дешевий.

Для порівняння, реалізація такого ж функціонала програмними засобами на general purpose CPU, зі зверненням до зовнішніх даних через складний стек уведення-виводу (через ядро й драйвер), різко зменшила б продуктивність, та й у форматі сервера неможливо було б забезпечити потрібну щільність портів.

Пересилання даних разом з усією низькорівневою логікою, яка обробляє кожний пакет, умовно виділяють у площину передачі даних, Data Plane. Майже вся сигналізація – протоколи виявлення сусідства, протоколи маршрутизації й керування трафіком, різні механізми балансування, запобігання петель, телеметрії, керування сервісами й смугою пропущення, авторизації, автентифікації й аккаунтингу – усі вони ставляться до площини керування – Control Plane. Чим інтенсивніше Control Plane і чому «тонше» Data Plane – тим більше доцільно віртуалізувати цю мережеву функцію, перенести її з мережевого заліза на сервер. І навпаки, чому «товстіше» Data Plane і чому примітивніше Control Plane – тем функція складніше у віртуалізації, і краще її залишити усередині мережевого пристрою.

**CE (Customer Edge)** маршрутизатори мають дуже широкий функціонал Data Plane – від простого роутингу до підтримки спеціальних методів

маршрутизації PBR/ABF, від примітивних ACL до складного Zone-Based фаєрволлингу, такі роутери якої підтримують різні стандарти енкапсуляції й туннелювання (GRE/IPIP/MPLS/OTV), трансляцію мережевих адрес (NAT, PAT), складні QoS-політики (policing, shaping, marking, scheduling). Функціонал Control Plane теж дуже широкий – від примітивних ARP/ND до найскладніших фіч типу MPLS Traffic Engineering. Подібний Branch-Маршрутизатор, з навантаженням кілька десятків або сотень мегабіт у секунду може бути прекрасно віртуалізований – на промисловому сервері можна розмістити десяток-іншийтаких елементів.

**P/PE маршрутизатори**, що мають схожий функціонал, призначені для установки мережі оператора, і їх data-plane обробляє колосальна кількість трафіку: від сотень гігабіт до десятків терабіт у секунду. При цьому вони повинні мати максимальну щільність портів. Чи варто говорити, що сервер традиційної архітектури не розрахований на Такі обсяги данних, которими оперують Маршрутизатори провайдерського класу. Їх Dataplane занадто «товстий» і не цікавий для віртуалізації. За аналогією з операторськими роутерами, майже неможлива віртуалізація високопродуктивних комутаторів, що широко використовуються в ЦОДах.

Єдине, що іноді може або повинне бути віртуальним стосовно а таким велетням – лише площина керування, повністю або частково. Data Plane у цьому випадку повністю залишається на пристрої, а саме пересилання пакетів виконується по таблицях, запрограмованих зовнішньою системою. Такою системою може бути не тільки традиційний контролер SDN у вузькому розумінні (працюючий по Openflow стандарту) – це може бути будь-який оркестратор, медіатор, драйвер мережевого елемента з підтримкою будь-якого доступного інтерфейсу (CLI, Netconf, SNMP).

Під цю же категорію підпадає такий традиційний елемент, як скажемо, **BGP Route-Reflector**, «що розливає» по мережі таблиці маршрутизації BGP. Такі елементи добре підходять для віртуалізації, але на мережі оператора їх лічені

одиниці. NFV-концепція стосується скоріше динамічних, абонентських, масових, а не інфраструктурних сервісів.

Забігаючи вперед скажу, що винесений Control Plane і таке поняття як SDN співвідносяться з NFV у трохи іншому ключі – скоріше як елемент мережевої інфраструктури для самого NFV, а не як функція.

**CPE (Customer Premises Equipment або Residential Gateway) –** та сама коробочка, установлювана у квартирі абонента, куди приходить кабель від оператора. Вона роздає інтернет на всі пристрої у квартирі і є центральною крапкою його домашньої мережі: на ній працює DHCP-сервер, кешуючий DNS-сервер, клієнт протоколу, по якому «спілкується» із провайдером: PPPoE/L2TP/PPTP/DHCP/802.1x. З dataplane-фіч визначальними також є NAT і Firewall. Від коробочки з портом для включення операторського кабелю в кожному разі нікуди не подіятися, і вимоги до пристрою залишаються колишніми: маленький, симпатичний, безпроблемний, дешевий девайс. Ці сподівання давно вловили виробники електроніки, і на нашому ринку доступні такі пристрої на будь-який смак і гаманець. Треба віддати належне вендорам найпоширеніших чипсетів таких «мильниць»: їх обчислювальна здатність може бути дуже високої, а такі ресурсномісткі функції як NAT і комутація можуть бути реалізовані апаратно.

Такі пристрої на даний момент не відповідають вимогам операторів скоріше через недолік функціонала: провайдер найчастіше не може оцінити їхній стан і завантаження, вони часто є вузьким місцем для надання послуги; оператор не бачить локальну мережу абонента й не управляє налаштуваннями CPE, що утрудняє діагностику; відсутність необхідних фіч на цьому елементі утрудняє запуск нових послуг; відновлення ПЗ цих пристроїв, як правило, проводиться несвоєчасно, що привносить додаткові ризики. Я певен, що найближчим часом нас чекає переосмислення моделі таких послуг, як батьківський контроль, інтернет-кінозал, керування розумним будинком, бекап і реплікація, зберігання й перегляд відеозаписів з домашніх камер відеоспостереження, сервісів на зразок «

робочого стола школяра» і т.д., і значну роль у цьому зіграє CPE, віртуалізовані в домені оператора. Наявність індивідуального абонентського vcpe-контейнера, через який проходить абонентський трафік, імовірно, змінить принцип обліку споживання послуг: об'єктом тарифікації, нарешті, зможе стати не тільки трафік, а обчислювальний або дискові ресурс. «Коробочка з антеною» у такій послузі залишиться на своєму місці в абонентській квартирі, але буде мати самий мінімальний набір функцій.

**BNG/BRAS** – так називаються роутери-шлюзи, через які абоненти оператора виходять в Інтернет. Саме ці маршрутизатори виконують «дозування» і облік споживаних послуг відповідно до тарифного плану по кожному з користувачів. На ці пристрої доводиться колосальне обчислювальне навантаження – кожного абонента необхідно авторизувати в AAA-системі по протоколу RADIUS, одержати унікальні правила й атрибути по кожній із сесій щоб створити для кожного свій, унікальний сервіс зі своїми класами й обмеженнями, правилами обліку й адресації, політиками QoS і ACL. Цей елемент відповідає за призначення адрес, підтримує роботу сигналізації ARP/ND/PPP/LCP/DHCP з десятками й сотнями тисяч абонентських пристроїв. По кожній сесії BNG повинен розуміти скільки трафіку потрапило в той або інший клас, скільки часу тривала сесія, кому перекрити доступ в Інтернет, кого «загорнути» на портал, а кому побільшати швидкість. Чи потрібно говорити, що ці залізяки повинні мати дуже продуктивним Control Plane BNG має мережеві процесори ще більш складні, чим на P або PE платформах! Крім пересилання пакетів ця розумна залізка витворяє з ними більш цікаві «вправи»: кожна абонентська сесія програмується в мережевий процесор, по кожній може вестися по кілька лічильників, у кожної можуть бути свої правила, свої класифікатори, свої швидкості доступу.

Така «пакетна магія» неможлива без додаткових апаратних засобів: сотень тисяч лічильників і величезних обсягів спеціальних типів пам'яті: TCAM'а для класифікаторів і списків доступу, пакетної пам'яті для організації черг. Деякі

технології абонентського доступу використовують туннелюваня (L2TP/PPPoE), і тоді на BNG покладає ще й енкапсуляція-декапсуляція даних, а чим більше операцій виконується над мережевим пакетом, тим більше переваг має ASIC перед x86. Втім, віртуалізований BNG може бути цікавий при сумарних обсягах трафіку не більш декількох одиниць Gbps, з дуже динамічними сервісами й більшою кількістю сесій і bringup/teardown подій. Гарний приклад – Wifi на транспорті: з малим обсягом трафіку, високою динамікою, з наявністю додаткових сервісів начебто баннерної реклами, які можна розмістити на тому ж кластері серверів, де розгорнуть vBNG.

**NAT – трансляція мережевих адрес** – у зв'язку з виниклим дефіцитом публічних адрес, що перебувають у розпорядженні операторів зв’язку, ця фіча стала в останні роки затребувана не тільки в Enterprise і SMB, але й в Sp-сегменті. Nat-функціонал ставиться до одній і самих ресурсномістких Data Plane функцій і вимагає значних обчислювальних ресурсів. Складність її пов'язана з величезною кількістю з'єднань, що одночасно відслідковуються. Відкриття лише однієї цієї сторінки (з усіма об'єктами й картинками) створила, використовувала й закрила сотні трансляцій на NAT пристрої. Звичайно ж, у тому випадку, якщо ваш оператор використовує NAT. Клієнти пірінгових файлообмінних мереж генерують багато тисяч трансляцій на один мегабіт смуги.

Розвиток спрощених режимів NAT (так званих Carrier-Grade режимів) дозволило добитися збільшення продуктивності, однак виконання NAT на мережевих процесорах сьогодні скоріше виключення, ніж правило. Великий обсяг трафіку, який повинен обслуговувати комплекс NAT сьогодні реалізується на високопродуктивних серверах, установлюваних у слот одного із центральних маршрутизаторів на мережі. Даний підхід дозволяє уникнути зовнішніх мережевих з'єднань – комплекс підключається прямо на backplane роутера й виглядає для маршрутизатора як ще один лінійний модуль.

Ще одна яскрава характеристика NAT – товщина – його можливо розділити на більш дрібні частини шляхом дроблення пулу адрес, але це зробіть

його менш ефективним. Чим він більш централізований – тим краще використовується доступний адресний ресурс. Через товщину, більші обсяги трафіку й великого обчислювального навантаження NAT є цікавим кейсом для концепції розподіленого NFV (dNFV), коли мережеві функції логічно централізовані (маємо єдиний пул адрес і ресурсів і єдину крапка керування), але при цьому територіально розподілені по вузлах і перебувають топологічно близько до вузлів транзиту, що обслуговується трафіку.

**Security:** Функціонал мережевої безпеки по своїх властивостях близький до NAT – також характеризується величезною кількістю й динамікою станів (Stateful FW, NGFW, IDS, IPS, Антивірус, Антиспам), більшим споживанням пам'яті й ресурсів CPU, що робіть його цікавим для віртуалізації. На відміну від NAT, цей сервіс не можна назвати товстим. Його можна ділити аж до співвідношення 1 абонент на 1 контейнер, тому дана послуга вважається одним із самих явних use-case для NFV.

**DPI –** глибока інспекція пакетів – функція, що дозволяє операторові зв'язку обмежувати або визначати рівень сервісу (квоту, швидкість) для певного типу застосунку або вмісту. Одним із прикладів може служити така розповсюджена послуга «батьківський контроль», при активації якої організує проходження абонентського трафіку через фільтр на підставі класу відвідуваного ресурсу. Data Plane цього сервісу звичайно реалізується програмними засобами, а цікава операторові продуктивність може починатися з досить малих величин – це теж один з яскравих юз-кейсів.

**MMSC/IMS – IP Multimedia Subsystem** – «прикордонний» елемент для надання голосових сервісів, трансляції телебачення, стрімінгу відео для таких застосунків, як домашній кінотеатр або відеоспостереження, різних мультимедійних меню й сервісу обміну миттєвими повідомленнями. Відмінно доповнює портфоліо операторських послуг.

Інфраструктурні елементи мобільних мереж GSM/UMTS/LTE/Wifi: бездротові контролери, EPC, IMS, CSN, SMSC, PCC, Gwc, Gwu, PGW, SGW, HSS,

PCRF – ці функції мають високі вимоги по обчислювальних ресурсах, а за обсягом переданих даних – скоріше помірні. Лише деякі з них мають ряд особливостей, пов'язані із забезпеченням необхідних параметрів QoS.

Для повноти картини згадаю такі застосунки, як DHCP, DNS, HTTP-сервери (з найширшим вибором доступних WEB-застосунків), HTTP Proxy (включаючи різні оптимізаторів, прискорювачі й URL-фільтри), SIP Proxy і т.д. Їх можна відносити до мережевих функцій, можна сперечатися й відносити їх до Application-рівню моделі OSI, але, у кожному разі, вони затребувані й легко віртуалізуємі, і також можуть служити складовими «цеглинками» послуг нового покоління.

**Розвиток і поточний статус стандарту NFV**

Отже, із самими функціями більш-менш ясно. Залишилося зрозуміти яким образом їх можливо комбінувати один з одним, як забезпечувати ізоляцію мережевих з'єднань між цими віртуальними елементами, безпеку, масштабованість і відказостійкість, як зробіти систему прозорою й керованою, зрештою, яке вибрати устаткування – обчислювальне й мережеве – і як його налаштувати. Заради відповідей на ці питання сім найбільших закордонних провайдерів (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica і Verizon) вирішили об'єднати свої зусилля в дослідженнях і в 2013 р. створили робочу групу під егідою Європейського інституту по стандартизації в області електзастосунку ETSI. Пізніше до їхніх експериментів приєдналися десятки виробників мережевого устаткування й ПЗ, і на даний момент в NFV ISG входить кілька сотень учасників.

На мережах учасників-операторів були організовано кілька тестових зон і розподілені теми досліджень і найцікавіші питання для вивчення. Практичні наробітки фіксуються, узагальнюються й публікуються для наступного коментування й виправлення й за результатами роботи формулюються нові питання. Ще рік тому технологія NFV була мало наповнена конкретикою, але з

кожною новою публікацією мозаїка майбутнього стандарту виглядає усе повніше.

Примітно, що успішний досвід «пілотування» найчастіше переймає й тиражується « як є», і навіть незважаючи на «пропрієтарність» і поки ще відсутність повноцінного відкритого стандарту, ряд вендорів, що бажають зайняти цю затребувану нішу, уже надає не тільки самі віртуальні елементи, але й цілісні застосунки NFV, різного масштабу, різного ступеня готовності й повноти, по-різному реалізовані.

В 2014 році до вендорсько-провайдерских ініціатив приєдналося Opensource співтовариство Linux Foundation, чим був даний старт проекту платформи з відкритим вихідним кодом – OPNFV, який повинен розширити інтерес індустрії до використання загальнодоступних і відкритих продуктів у цій ніші. Наприклад, для мережі підійшов би OpenvSwitch і Opendaylight, для рівня віртуалізації – Openstack, у якості сховища – Ceph, у ролі гіпервізора – KVM, хостова й гостьова ОС – Linux. Сьогодні доступний перший реліз цього продукту, Arno. Активності по стандартизації NFV також помічені в таких галузевих організаціях, як, наприклад, MEF або TM Forum.

### 3.2 Розробка структурної схеми

На думку експертів Vodafone Research впровадження NFV в операторів зв'язку приблизно скоротить OPEX на 60% протягом трьох років, CAPEX – на 59% протягом п'яти років. При цьому очікується, що завантаження устаткування підвищиться на 20-30%.

Network Function Virtualization є доповненням до програмно-обумовлених мереж (SDN), але може реалізовуватися як спільно, так і окремо друг від друга. Давайте розберемося, у чому принципові відмінності між ними?

Архітектуру віртуалізації мережевих функцій (NFV), розроблену ETSI, описує документ ETSI GS NFV-0010 V0.1.7 і в даній роботі вона представлена у вигляді структурної схеми:

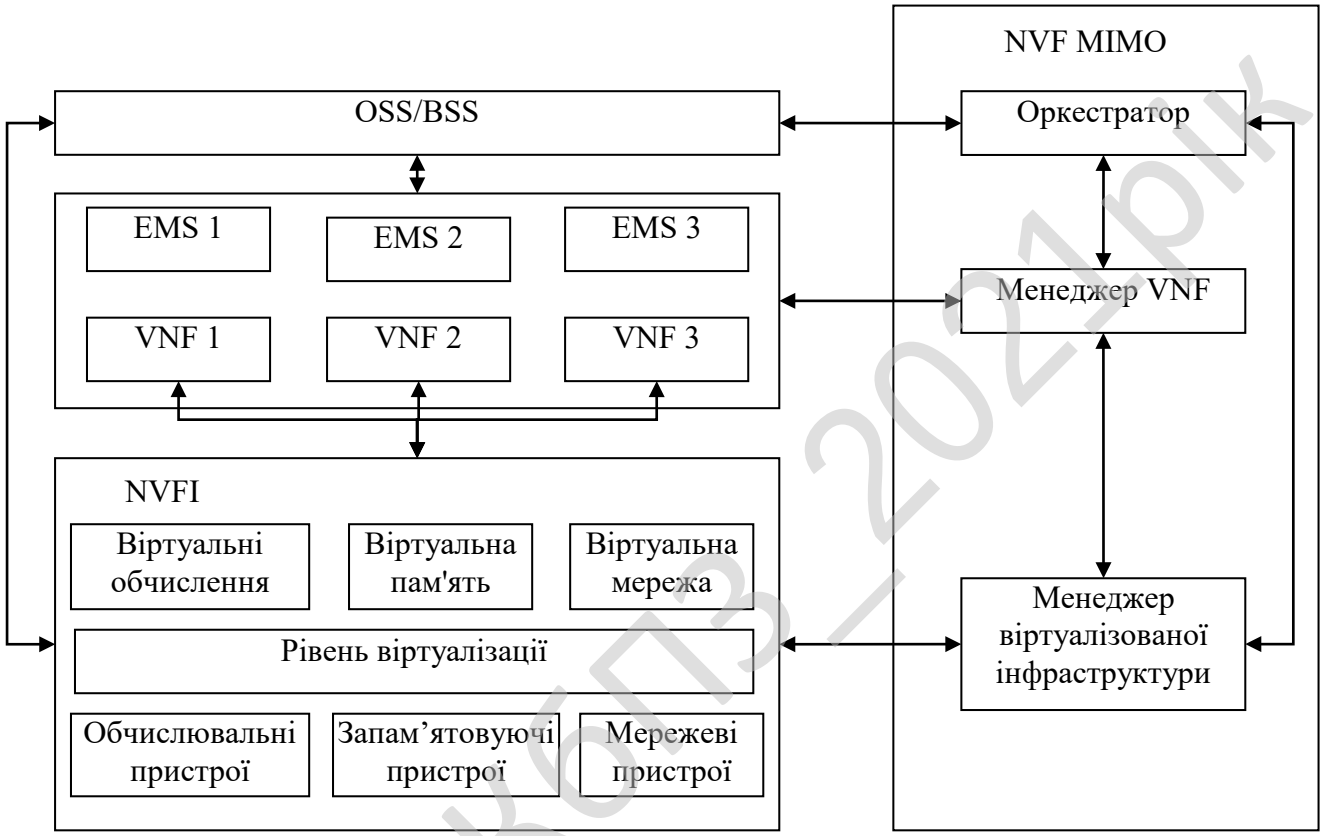


Рисунок 3.1 – Структурна схема системи

Архітектура NFV включає наступні основні елементи:

– OSS/BSS стандартні системи операторів зв'язку – біллінг, чарджинг, тарифікація, CRM, самообслуговування абонентів і т.д. Вони як були, так і залишаються, і пов'язані з NFV тільки одним специфікованим інтерфейсом (Os-Ma).

– VNF (Virtual Network Function) – віртуальна мережева функція, наприклад: DNS, DHCP, комутатор, маршрутизатор, балансувальник або базова станція і т.д.

– EMS (Element Management System) – система керування й адміністрування однією або декількома VNF.

– NFVI (Network Function Virtualization Infrastructure) – інфраструктура NFV: апаратні й програмні ресурси у фізичному й віртуальному виді, на яких працюють віртуальні мережеві функції VNF. Можуть перебувати локально або бути розподіленими територіально й з'єднані мережею оператора.

– Hardware Resourses (апаратні ресурси, тобто обчислювальні, мережеві й ресурси зберігання) – у це фізична частина інфраструктури NFVI – будь-який стандартний комутатор, або фізичний сервер, або пристрій зберігання і т.д.

– NFV Orchestrator (оркестратор NFV) – адміністрування інфраструктури NFV програмними ресурсами, створення закінченої послуги з декількох VNF.

– VNF Manager – менеджер відповідає за життєвий цикл VNF: інсталяція, активація, масштабування, відновлення й термінування. Може відповідати за роботу однієї або декількох VNF.

– Virtualized Infrastructure Manager (менеджер віртуальної інфраструктури) – відповідає за взаємодію віртуальної мережевої функції з апаратними й програмними ресурсами й інвентаризацію наявних ресурсів, а також відповідає за збір подій і питання продуктивності.

До факторів, що стримують розвиток NFV, можна віднести наступні:

– Стандарти й сумісність. Стандарти NFV ще не до кінця розроблені, що привносить різнобій у вендорні реалізації, викликає проблеми сумісності реалізацій NFV різних вендорів, а також ризики в стратегіях операторів і сервісів-провайдерів.

– Технічна зрілість. Хоча більшість нових NFV-технологій тільки перебуває в стані тестування, багато операторів зв’язку вже починають впроваджувати NFV.

– Неясність цінності для бізнесу. Перші комерційні реалізації NFV з'явилися тільки в 2015 р. Цінність для бізнесу від реалізації NFV поки лежить у теоретичній площині й реальних підтверджувальних кейсів усе ще недостатньо.

– Проблеми міграції. Міграція традиційної мережевої інфраструктури до архітектури NFV є складним і багатоступінчастим завданням. У цей час оператори й вендори устаткування поки не мають систематичного досвіду подібних переходів і переконливих «історій успіху».

– Проблеми організаційної структури. У даний момент у структурі практично будь-якого оператора зв'язку департамент інформаційний технологій (IT) і технічний департамент мережі зв'язку оператора (CT) організаційно розділені. Тим часом, NFV і SDN ставляться саме до сфери IT. Тому потрібно не тільки трансформація базової мережі оператора, але і його організаційної структури. А це досить нетривіальне й хворобливе завдання. Однак, після її завершення, як внутрішня корпоративна мережа оператора, так і його базова мережа, будуть мати єдину інфраструктуру, що приведе як зниженню накладних витрат, так і до підвищення ефективності бізнесу.

– Моніторинг продуктивності і якісних параметрів мережі. Першопрохідники й піонери зіштовхнулися із проблемою, що існуючі системи моніторингу не адаптовані під завдання NFV. Якщо ми ведемо мову про часткову заміну устаткування на стандартні сервера, то й апаратні застосунки для моніторингу каналів зв'язку повинні бути теж замінені на програмний застосунок для установки на стандартний сервер. І такі застосунки вже з'являються, наприклад, Truspeed NFV від Viavi Solutions.

Перехід до NFV можна зрівняти тільки з переходом від аналогових телефонних станцій до цифрових на початку 70-х років, коли виробникам устаткування знадобилося така ж переорієнтація й таке ж переоснащення. У той час також був багато дискусій на тему про те, чому «цифра» краще?. И такі приклади можна приводити у великій кількості, але віртуалізація мережевих функцій дозволить відповідати на вимоги користувачів у частині нових послуг суттєво швидше й зможе скоротити витрати на модернізацію мережі.

### 3.3 Розробка функціональної схеми

Розглянемо методику системного аналізу Network Function Virtualization для операторів зв’язку. Приведемо функціональну схему розробки структури Network Function Virtualization для операторів зв’язку, визначимо параметри Network Function Virtualization для операторів зв’язку. Сформулюємо завдання оптимізації структури Network Function Virtualization для операторів зв’язку по технічних параметрах. Представимо алгоритми й методи рішення.

Необхідний рівень функціонування інформаційної системи (ІС) пов'язаний із забезпеченням параметрів, значення яких задаються у відповідності зі специфікою діяльності організації й особливостями використовуваних ІС у даній діяльності (таблиця 3.1).

Забезпечення необхідних параметрів функціонування ІС залежить від технічних характеристик програмно-апаратного комплексу (ПАК), використовуваного для зберігання інформаційного ресурсу при Network Function Virtualization для операторів зв’язку.

Таблиця 3.1 – Параметри, що характеризують необхідний рівень функціонування ІС

| Позначення | Коментар |
|---|---|
| $k_v$ | збільшення обсягів оброблюваної інформації, (% приросту в од. часу) |
| $k_z$ | збільшення числа користувачів (% приросту) |
| t | середній час очікування користувача (час реакції, мс.) |
| $v_{кр}$ | критична маса інформації, що може бути загублена при збоях системи без порушення діяльності (% від загального обсягу за од. часу) |

Розміщення інформації повинне здійснюватися з однієї сторони на тих носіях, характеристики яких задовольняють заданим параметрам Network

Function Virtualization для операторів зв’язку. З іншого боку, необхідне зниження витрат на його зберігання. Домогтися оптимального розподілу ресурсів зберігання при зниженні витрат на зберігання й супровід інформації можна, формуючи мережеву віртуалізацію NFV в рамках моделі ETSI у вигляді багаторівневої блокової структури.

Із цією метою необхідно класифікувати інформацію, формовану в ході діяльності організації, по ступені її значимості й застосовувати інструменти керування розміщенням даних на пристроях зберігання відповідно до цієї класифікації.

Припустимі фінансові витрати дозволяють класифікувати інформацію на наступні класи:

– критична;

– важлива;

– неважлива.

Інформація в період свого існування проходить наступні стадії: створення, обробка, зберігання, архівування, видалення. За цей час міняється її актуальність і затребуваність, відповідно змінюється приналежність до одному з перерахованих класів.

Діяльність організації розглядається як множина інформаційних процесів $A_i$, $i = 1..m$, у ході яких використовується Network Function Virtualization для операторів зв’язку.

Для розрахунку вхідних параметрів структури Network Function Virtualization для операторів зв’язку пропонується функціонально-орієнтований підхід, що виконується в наступні етапи:

– збір статистики використання Network Function Virtualization для операторів зв’язку і-м процесом (етап 1):

$v_i$ – сумарний обсяг, Мб;

$k_i$ – число користувачів;

$z_i$ – число операцій уведення-виводу за од. часу;

$v_{zi}$ – обсяг інформації, переданої за одну операцію уведення-виводу, Мбіт;

– визначення состава Network Function Virtualization для операторів зв'язку для і-го процесу за критерієм припустимі фінансові витрати (етап 2) і відповідному кожному класу параметрів:

$n_{ij}$ – період зміни цінності Network Function Virtualization для операторів зв'язку для і-го процесу й j -го класу Network Function Virtualization для операторів зв'язку;

$m_i$ – період існування Network Function Virtualization для операторів зв'язку, використовуваного і-м -процесом;

– обробка відомостей (етап 3) з використанням методів математичної статистики, що дозволяють визначити прогнозні значення параметрів Network Function Virtualization для операторів зв'язку на майбутній період: обсяг, число операцій уведення-виводу, число користувачів;

– формування Network Function Virtualization для операторів зв'язку у вигляді багаторівневої структури (етап 4). Число й зміст рівнів визначається на підставі відомостей, отриманих на етапі 1 і 2, а також на підставі вимог до ІС. Для кожного рівня задається множина процесів E і відповідних їм Network Function Virtualization для операторів зв'язку, розташовуваних на р-м рівні.

$$E_p = \{A_{k,} v_{kj} (k = 1,q)\}, \quad (3.1)$$

де q – число процесів.

Розрахунок значень параметрів (етап 5) для кожного рівня обчислюється по формулах (3.2) – (3.4), які є вихідними даними для проектування оптимальної структури зберігання інформаційного ресурсу при Network Function Virtualization для операторів зв'язку.

Уведемо наступні позначення, що характеризують технічні параметри:

– P – загальна кількість рівнів Network Function Virtualization для операторів зв'язку;

– $V_p$ – сумарний обсяг ресурсу для р-го рівня, Мб;

– $Ps_p$ – пропускна здатність для р-го рівня, Мбіт/сек;

– $Z_p$ – число операцій уведення-виводу для p-го рівня;

– t – час доступу, сек;

– $k_{raid}$ – коефіцієнт використання додаткового простору RAID масивом.

Сумарний обсяг ресурсу для p-го рівня обчислюється за формулою:

$$V_p = k_{raid} * \sum_{k=1}^{q} v_k . \quad (3.2)$$

Пропускна здатність для p-го рівня обчислюється за формулою:

$$Ps_p = \left[ \sum_{k=1}^{q} (z_k * v_{zk}) \right] / t \ . \quad (3.3)$$

Число операцій уведення-виводу для p-го рівня обчислюється за формулою:

$$Z_p = \sum_{k=1}^{q} z_k . \quad (3.4)$$

Дані параметри використовуються для проектування оптимальної структури Network Function Virtualization для операторів зв'язку, вибору апаратних ресурсів, їхнє налаштування, конфігурування, оцінки витрат.

Розробку плану структури Network Function Virtualization для операторів зв'язку розглянемо як завдання, що полягає в оптимізації параметрів багаторівневого середовища зберігання інформаційного ресурсу при Network Function Virtualization для операторів зв'язку із заданими локальними характеристиками кожного рівня й у той же час об'єднаними сукупністю обмежень на все середовище зберігання. Оптимальним планом є номенклатура дискових масивів і кількість зовнішніх запам'ятовувальних пристроїв (ЗЗП), складових дисковий масив, при мінімальній сумарній вартості зберігання. У такій постановці ми приходимо до завдання математичного програмування із блокової (багаторівневої) структурою.

З обліком вищесказаного дамо формалізований опис завдання. Нехай маємо P-рівнів і $m_p$, p = 1..P параметрів, що характеризують ресурс, наявність кожного i-го параметра становить найменше $b_{pi}$ і найбільше $B_{pi}$, i = 1…$m_p$, значення у відповідних одиницях вимірів. Ці параметри призначені для

формування $n_p$ типів дискових масивів. Кожна одиниця j-го типу дискового масиву містить $a_{ij}$ одиниць і-го параметра ресурсу. Потрібно визначити, які типи дискових масивів і яка кількість дисків необхідно для формування багаторівневого середовища зберігання з найкращими показниками для прийнятого критерію оптимальності – вартості F.

Позначимо через $x_{pj}$ – кількість одиниць j-го типу дискових масивів на р-м рівні, тоді математичну постановку завдання можна записати у вигляді:

$$F = \sum_{p=1}^{P} \sum_{j=1}^{n_p} v_p \, c_{jp} \, x_{jp} \;\; \overrightarrow{j} \; \mathrm{m} \tag{3.5}$$

при обмеженнях

$$\sum_{p=1}^{P} \sum_{j=1}^{n_p} a_{ij} x_{pj} \le B_i , \quad \mathrm{i} = 1..\mathrm{m}_0; \tag{3.6}$$

$$\sum_{p=1}^{P} \sum_{j=1}^{n_p} a_{ij} x_{pj} \ge b_i \quad \mathrm{i} = 1..\mathrm{m}_0; \tag{3.7}$$

$$\sum_{p=1}^{P} \sum_{j=1}^{n_p} x_{pj} \le b_0 ; \tag{3.8}$$

$$\sum_{j=1}^{n_p} a_{ij} x_{pj} \le B_i , \; \mathrm{i} = 1..\mathrm{m_p}, \; \mathrm{p} = 1..\mathrm{P}; \tag{3.9}$$

$$\sum_{j=1}^{n_p} a_{ij} x_{pj} \ge b_i , \; \mathrm{i} = 1..\mathrm{m_p}, \; \mathrm{p} = 1..\mathrm{P} ; \tag{3.10}$$

$$x_j \in \Omega, \qquad \mathrm{j} = 1..\mathrm{n_p}, \; \mathrm{p} = 1..\mathrm{P}, \tag{3.11}$$

де

– P – загальна кількість локальних блоків;

– $m_0$ – число обмежень у блоці-зв'язуванню;

– $n_p$ – число змінних в р – м локальному блоці;

– $m_p$ – число обмежень в р-м локальному блоці;

– $v_{pj}$ – обсяг диска j-типу в р-м локальному блоці;

– $c_{pj}$ – вартість зберігання інформації на диску j-типу в р-м локальному блоці;

– $b_0$ – загальна кількість дисків зберігання інформаційного ресурсу при Network Function Virtualization для операторів зв’язку;

– $b_{pi}$ – найменше значення параметра Network Function Virtualization для операторів зв’язку;

– $B_{pi}$ – найбільше значення параметра Network Function Virtualization для операторів зв’язку;

– $\Omega$ – множина цілих, позитивних чисел.

Умови (3.6), (3.7), (3.8) описують блок-зв'язування, (3.9), (3.10) – окремі блоки (рівні), (3.11) – умова цілочисельного значення змінної $x_{pj}$.

Критична інформація актуальна для організацій і повинна розташовуватися на високопродуктивному дисковому масиві (High HDD). Важлива інформація розташовується на дисках середньої продуктивності (Middle HDD). Архівні дані, довгостроково незатребувані, розташовуються на менш продуктивних носіях (Low HDD). Відповідно, для зберігання даних класів інформації необхідна 3-хрівнева структура зберігання.

Розглянемо параметри математичної моделі для 3-хрівневої структури Network Function Virtualization для операторів зв’язку. Значення параметрів $b_i$ і $B_i$ визначаються по формулах (3.2)-(.34) і задаються для кожного рівня (блоку) індивідуально. Значення $a_{ij}$ визначаються технічними характеристиками дискових пристроїв провідних виробників.

Поряд із завданням (3.5)-(3.11), розглянута завдання, що полягає в оптимізації кількості невикористаного ресурсу і-го параметра на кожному р-м рівні $y_{pj}$ при заданій вартості F* = 150 тис. грн. Позначивши через $Y_{pj}$ кожний відповідний локальний критерій, приходимо до завдання багатокритеріальної оптимізації. Процес рішення в цьому випадку неминуче пов'язаний з експертними оцінками як самих критеріїв, так і взаємини між ними.

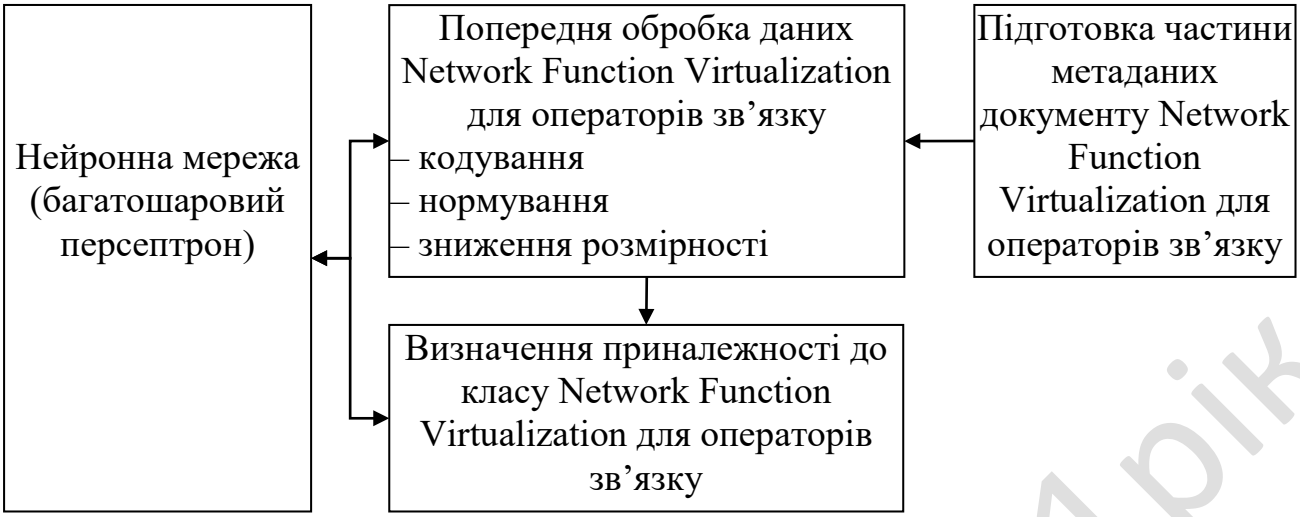


Рисунок 3.2 – Функціональна схема системи

Як показують розрахунки, введення багаторівневої структури зберігання інформаційного ресурсу при Network Function Virtualization для операторів зв’язку ефективно при досягненні обсягів кілька сотень гігабайт. У цьому випадку, адміністратори даних повинні мати можливість прогнозувати ріст обсягів з урахуванням вимог, які змінюються, вчасно планувати витрати на оснащенні апаратними засобами.

Наведена модель дозволяє здійснювати прогноз і планування витрат. Разом з тим, вирішується завдання оптимального використання апаратних ресурсів.

Розглянемо процес розробки інтелектуального класифікатора, побудованого на базі нейронної мережі (НМ). Описується й обґрунтовується структура нейронної мережі для класифікації Network Function Virtualization для операторів зв’язку, алгоритм її навчання. Під класифікацією Network Function Virtualization для операторів зв’язку розуміється віднесення об'єкта до одному із заздалегідь відомих класів: критична, важлива, неважлива.

Класифікація Network Function Virtualization для операторів зв’язку здійснюється з використанням програмного комплексу й включає наступні етапи обробки інформації.

Перший і другий етапи формують метадані Network Function Virtualization для операторів зв'язку, тобто дані про Network Function Virtualization для операторів зв'язку, які формуються частково користувачем, частково програмою.

Для здійснення класифікації виділені ознаки, що характеризують групу, до якої належить Network Function Virtualization для операторів зв'язку:

– $x_1$ – обсяг;

– $x_2$ – інтенсивність використання;

– $x_3$ – ким створюється;

– $x_4$ – інтенсивність зміни;

– $x_5$ – дата створення;

– $x_6$ – найменування підрозділу;

– $x_7$ – тип ресурсу;

– $x_8$ – дата зміни;

– $x_9$ – найменування програми.

Третій етап, попередня обробка вхідних даних, є найбільш трудомістким етапом класифікації й включає:

– кодування, тому що використовуваний алгоритм класифікації працює тільки із числовою інформацією, тоді як ознаки $x_3$, $x_6$, $x_7$, $x_9$ приймають значення з дискретного набору. У цьому випадку пропонується використовувати двійковий тип кодування $n \rightarrow n$, n – число категорій;

– нормування, тобто приведення до єдиного масштабу числової змінної на діапазон розкиду її значень. Для нормування пропонується використовувати статистичні характеристики середнє й дисперсія, тому що ознаки $x_1$, $x_2$ $x_4$, $x_5$ мають великий розкид значень;

– зниження розмірності, тобто відбір найбільш інформативних ознак. У ході експериментів виявлені ознаки, які найбільш впливають на точність роботи класифікатора: $x_3$, $x_6$, $x_7$, $x_2$, $x_5$.

Четвертий етап виконує класифікацію Network Function Virtualization для операторів зв'язку на підставі вхідного вектора $X = \{x_3, x_6, x_7, x_2, x_5\}$ до одному із заданої множини класів Y:

– критична;

– важлива;

– неважлива.

Етапи 3 – 4 виконуються автоматично з використанням інтелектуального класифікатора. Розробка інтелектуального класифікатора включає формування навчальної множини, вибір оптимальної архітектури НМ і алгоритму її навчання.

Для побудови класифікатора використовується багатошаровий персептрон, а точніше його двошарова реалізація.

На вхід нейронної мережі подається p-мірний вектор метаданих $X = \{x_i, \ i = 1,2,\ldots p\}$...Оптимальна кількість нейронів на схованому шарі H підбирається експериментально.

У вихідному шарі число нейронів збігається із числом класів M, тому що пропонується для кожного класу визначити свій нейрон. Між вхідним і схованим шарами, а також між схованим і вихідним шарами використовується повнозв'язна структура.

З урахуванням цих доповнень опишемо прийняті на рисунку 3.5 позначення:

– p – розмірність вихідних даних (кількість метаданих, використовуваних для класифікації);

– H – число нейронів на схованому шарі;

– $x_i$ – компонента вхідного вектора ознак, $i = 1,\ldots,p$;

– $x_0 = 1$ – постійний вплив, використовуваний для роботи нейронної мережі;

– $w_{ji}$ – вагарні коефіцієнти між вхідним і схованим шарами, $i = 0,1,\ldots,p$; $j = 1,\ldots,H$;

– vjk – вагові коефіцієнти між схованим і вихідним шарами, j = 0,1,…,H; k = 1…m;

– zj – значення виходу j-го нейрона схованого шару; $z_0 = 1$, j = 1,…,H;

– yk – значення виходу k-го нейрона мережі, k = 1…m...

Значення виходів $z_j$ і $y_k$ визначаються по формулах (3.10), (3.11).

$$z_j = f_1(\sum_{i=0}^{p} w_{j\,i} x_i) \tag{3.10}$$

$$y(x) = f_2(\sum_{j=1}^{H}(v_j z_j) = f_2(\sum_{j}^{H} v_j f_1(w_{j1}x_1 + w_{j2}x_2 + ... + w_{jp}x_p + w_{j0}) + v_0) \ . \tag{3.11}$$

Як функція активації $f_1(x)$ для нейронів схованого шару й $f_2(x)$ для нейронів на виході мережі використовуємо сигмоїдну функцію, позначивши її як f(x):

$$f(x) = \frac{1}{1 + e^{-ax}} . \tag{3.12}$$

Функціонування нейронної мережі залежить від величин, що характеризують зв'язки, тому, задавшись архітектурою нейронної мережі, необхідно знайти оптимальні значення всіх змінних коефіцієнтів w.

Оцінка якості роботи нейронної мережі визначається величиною функції обчислення помилки E, як середньоквадратичне відхилення поточних виходів від необхідних для кожної навчальної пари із заданої вибірки навчальної множини S: {X,Y}.

Процес навчання розглядається як завдання багатомірної оптимізації, що складається в пошуку оптимальних значень вагових коефіцієнтів w:

$$E(w) = \frac{1}{2}\sum_{k}(y_{k,s} - d_{k,s})^2 , \tag{3.13}$$

де

y – поточне значення k-нейрона для s-навчальної пари;

d – необхідне значення k-нейрона для s – навчальної пари.

У випадку визначення помилки для всього множини навчальних пар величина E обчислюється за формулою:

$$E = \frac{1}{N}\sum_{s=1}^{N} E_s(w) \to \min, \tag{3.14}$$

де N – потужність навчальної вибірки.

Тоді навчання нейронної мережі зводиться до знаходження вагових коефіцієнтів w, які давали мінімальне значення цільової функції E для всієї навчальної вибірки:

$$E(w) = \frac{1}{N}\sum_{s=1}^{N}(\frac{1}{2}\sum_{k=1}(y_{k,s} - d_{k,s})^2 = \frac{1}{N}\sum_{s=1}^{N}\left[\frac{1}{2}\sum_{k=1}^{m}(f_2\sum_{1}^{5}(v_j f_1(w_{j1}x_1 + w_{j2}x_2 ... + w_{jp}x_p + w_{j0}) + v_0) - d_{k,s})^2)\right] \to \min. \tag{3.15}$$

Пошук мінімуму функції здійснюється з використанням наступних алгоритмів: методу зворотного поширення помилки, методу сполучених градієнтів, генетичного алгоритму. У таблиці 3.5 наведені результати навчання при наступних параметрах моделі НМ p = 5, H = 8, m = 3, N = 63.

Щонайкраще завдання навчання вирішують генетичний алгоритм і метод сполучених градієнтів. Для поліпшення роботи генетичного алгоритму в процесі експериментів підібрані параметри, при яких досягаються прийнятні параметри швидкості збіжності і якості рішення.

Проведені дослідження підтвердили ефективність застосування нейронних мереж для класифікації Network Function Virtualization для операторів зв’язку. При певних налаштуваннях НМ можна домогтися результатів, коли ймовірність правильного розпізнавання становить 97%. Помилки виникають тільки на 2 векторах з 63.

Використання інтелектуального класифікатора дозволить значно знизити тимчасові витрати, пов'язані з обслуговуванням Network Function Virtualization для операторів зв’язку, знизити втрати, пов'язані з людським фактором.

Далі з використанням отриманих моделей, методик і алгоритмів:

1. Проведено оптимізацію структури Network Function Virtualization для операторів зв’язку.

2. Отримано прогнозну оцінку росту обсягу Network Function Virtualization для операторів зв’язку і визначені параметри оптимальної структури Network

Function Virtualization для операторів зв’язку з урахуванням змін обсягів Network Function Virtualization для операторів зв’язку.

3. Отримано оцінку ефективності пропонованих рішень оптимізації структури Network Function Virtualization для операторів зв’язку.

Оптимізація Network Function Virtualization для операторів зв’язку включає комплекс організаційних і апаратно-програмних рішень, спрямованих на забезпечення необхідного рівня функціонування ІС. У цьому зв'язку розроблений ряд організаційних мір, спрямованих на керування Network Function Virtualization для операторів зв’язку, що підвищить ефективність його використання.

Розроблено методичні рекомендації, що регламентують процес формування й зберігання інформаційного ресурсу при Network Function Virtualization для операторів зв’язку.

Розроблено модель захисту, що включає перелік груп користувачів і відповідні їм права доступу.

Оцінка ефективності пропонованих рішень по оптимізації структури Network Function Virtualization для операторів зв’язку виконана з урахуванням витрат, які несе організація протягом існування Network Function Virtualization для операторів зв’язку.

Визначення витрат по використанню Network Function Virtualization для операторів зв’язку проведено на підставі сукупної вартості володіння (СВВ) інформаційними ресурсами. Для оцінки СВВ використовується модель, що відбиває повний перелік статей витрат, пов'язаних із впровадженням і обслуговуванням ІС протягом строку функціонування.

$$C_{C\,C\,B} = \sum_{1}^{7} c_k \,, \tag{3.16}$$

де $c_k$– витрати на k-у статтю витрат ІС.

В умовах росту обсягів Network Function Virtualization для операторів зв’язку і збільшення числа інформаційних процесів, збільшуються витрати,

пов'язані з адмініструванням і людським фактором, питома вага яких становить 42%.

Проведений аналіз інформаційних систем і відповідних інформаційних ресурсів дозволив виявити динаміку росту й динаміку зміни цінності Network Function Virtualization для операторів зв’язку. За останні 3 роки Network Function Virtualization для операторів зв’язку збільшився в 3,5 рази, з них на неструктуровану інформацію доводиться 58%. Разом з тим, аналіз Network Function Virtualization для операторів зв’язку показав, що 30% інформації не використовується більше 12 місяців.

Проведені розрахунки СВВ зберігання інформаційного ресурсу при Network Function Virtualization для операторів зв’язку до оптимізації й після оптимізації його структури на період з 2011 по 2017 р. показали наступні результати.

Аналіз витрат і оцінка ефективності оптимізації структури Network Function Virtualization для операторів зв’язку, що ставляться до розробки, впровадженню й функціонуванню ІС, зроблений на основі даних відділу інформаційних технологій освітньої установи й відомостей про вартість устаткування провідного виробника систем зберігання даних.

Таким чином, використання оптимізації структури Network Function Virtualization для операторів зв’язку дозволить знизити поточні витрати, пов'язані з його обслуговуванням, а також одноразові витрати на апаратні ресурси.

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

### 3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3.

Розроблена діаграма взаємодії процесів використовується для представлення та візуалізації процесів обробки даних тобто структурного проектування бакалаврської роботи.

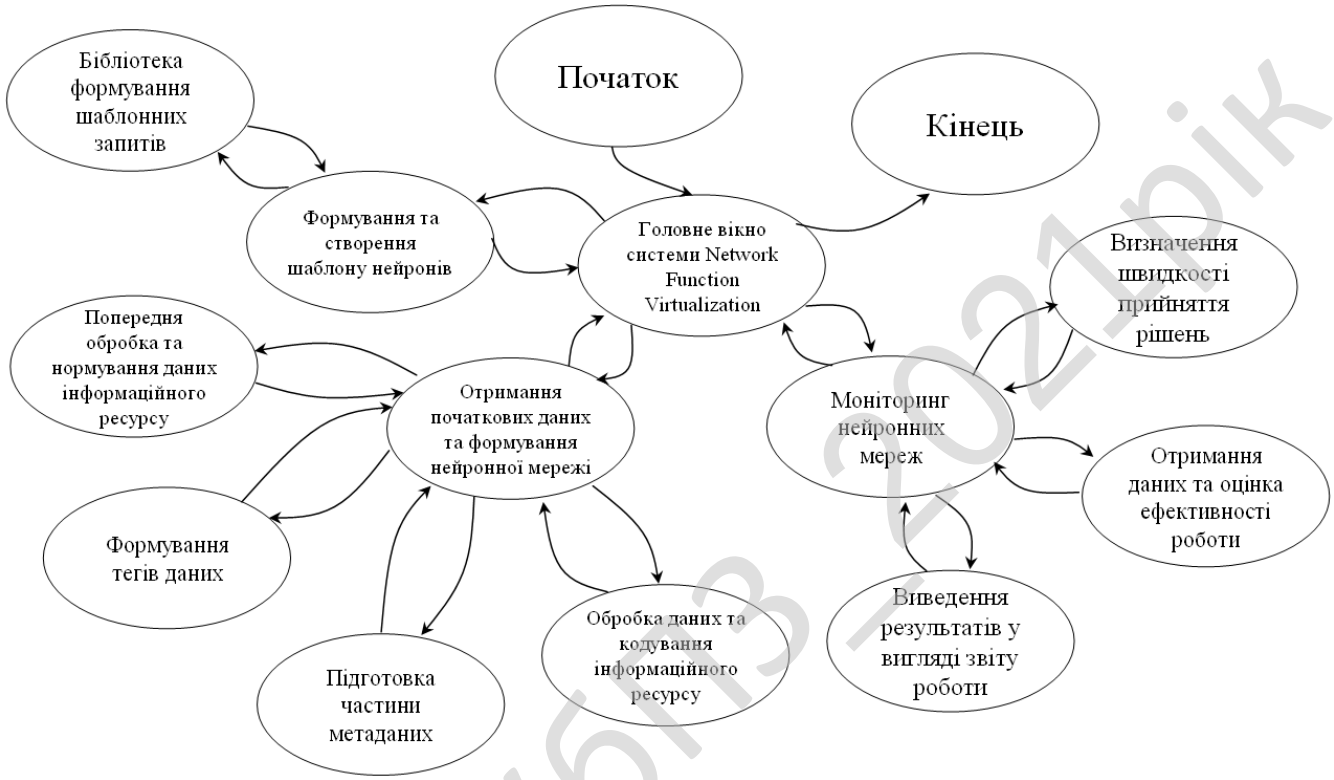


Рисунок 3.3 – Діаграма взаємодії процесів

Основні складові елементи діаграми взаємодії процесів це потоки даних:

– Репозиторії, потік сховища даних.

– Потоки зовнішні по відношенню до системи сутності.

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

– Потоки даних гібридні між елементами трьох попередніх типів.

Відповідно до документації 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.

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється. Таким чином у результаті після розгляду, вищеописаної системи, схеми структурної, функціональної, діаграми взаємодії процесів перейдемо до опису та розгляду блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

# 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1. З рисунку 4.1 видно, що після запуску програми відбуваються наступні дії:

– Виведення системи Network Function Virtualization;

– Запит оптимізація структури;

– Вибір шаблонів нейронів оптимізації структури інформаційного ресурсу;

– Завдання властивостей полів в рамках моделі;

– Створення шарів та додавання до них нейронів;

– Навчання нейронної мережі оптимізації структури інформаційного ресурсу;

– Формування та розпізнавання структури IP;

– Виведення результатів обробки даних;

– Запит аналізу мережі;

– Вибір нейронної мережі оптимізації структури IP;

– Тестування нейронної мережі оптимізації структури IP;

– Визначення швидкості прийняття рішень та оцінка ефективності роботи;

– Виведення результатів в форматі моделі ETSI;

– Запит бібліотеки шаблонів;

– Виведення існуючих шаблонів даних;

– Редагування та створення нових шаблонів;

– Запит завершення роботи системи.

З рисунку 4.2 видно, що відбуваються наступні дії:

– Виведення діалогового вікна вибору образів оптимізації для операторів зв’язку;

– Ініціалізація нейронної мережі системи Network Function Virtualization;

– Відкрити файл з образами з послідуючою реалізацією;

– Введення вхідних даних;

– Формування початкового вектору

– Створення вектора, що буде подаватися на вхід;

– Показати вектор у вигляді сітки;

– Подати вектор на вхід нейронної мережі;

– Розпізнавання вектору;

– Зчитати результати з вихідного шару нейронної мережі;

– Вивести отримані результати;

– Знищити вектор.

Розглянемо реалізацію модуля пошуку файлі, який необхіден для отримання початкових даних та формування нейронної мережі. Вихідний код:

```
// Пошук файлів
unit find;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, FileCtrl, ComCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    ListBox1: TListBox;
    Button2: TButton;
    DateTimePicker1: TDateTimePicker;
    Label2: TLabel;
    Bevel1: TBevel;
    ListBox2: TListBox;
    Edit2: TEdit;
```

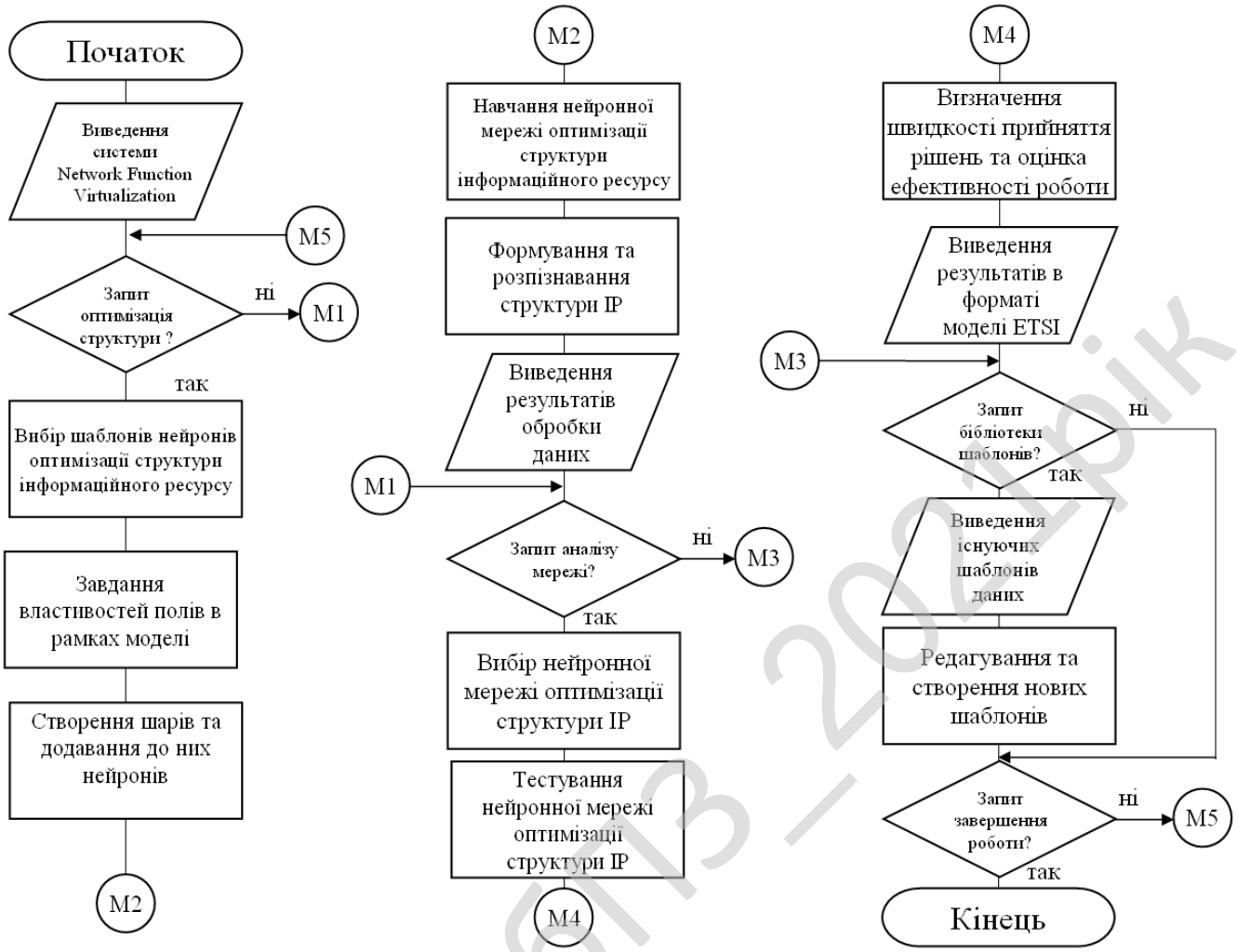


Рисунок 4.1 – Блок-схема основної програми

```
    Label3: TLabel;
    Button3: TButton;
    Edit3: TEdit;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    Bevel2: TBevel;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure CheckBox2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
  private
```

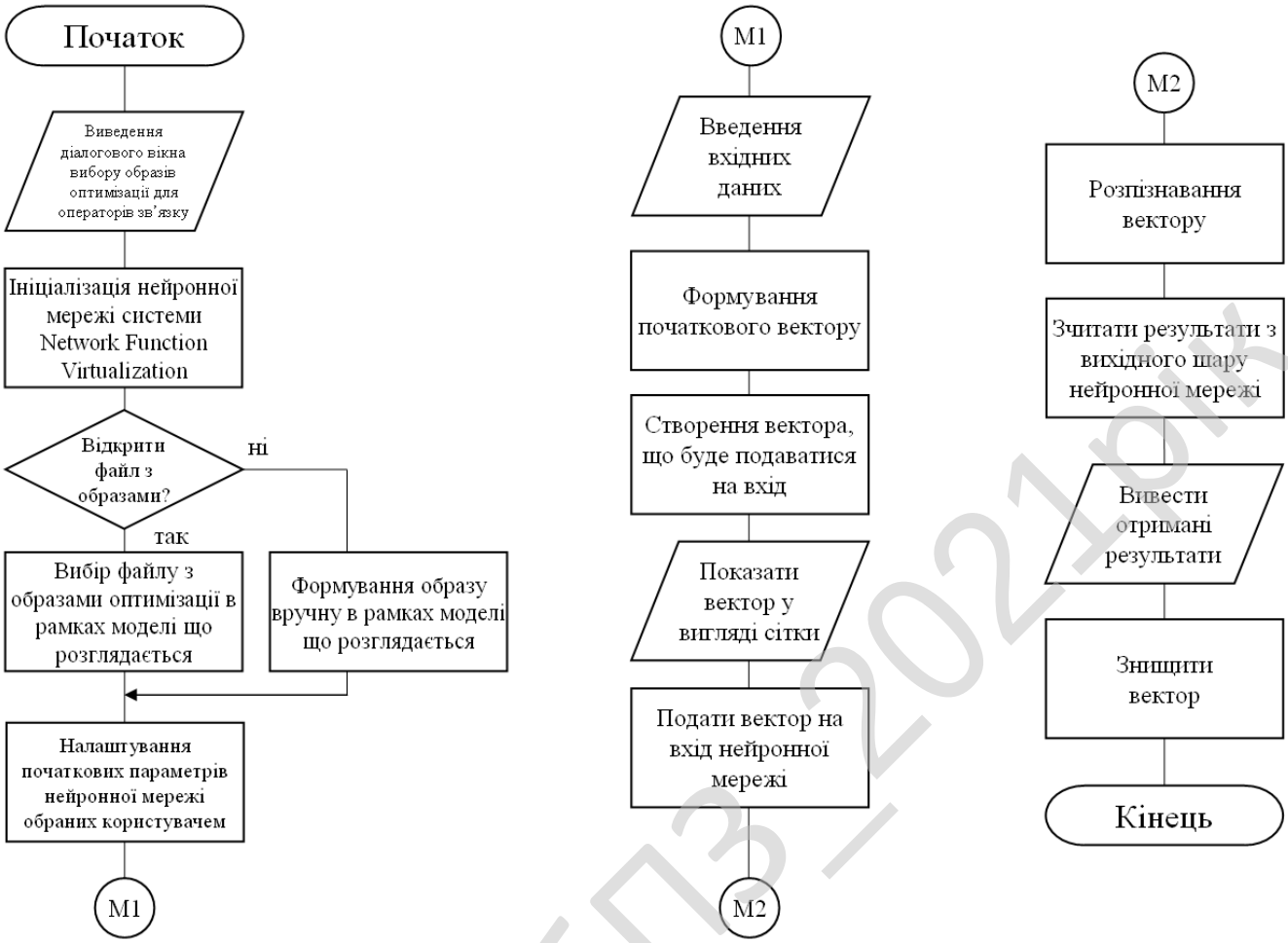


Рисунок 4.2 – Блок-схема роботи підпрограми

```
    { Private declarations }
  public
    { Public declarations }
    procedure FindFile(Dir:String);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
 Dir1,Dir2:string;
```

```
 SR:TSearchRec;
 FindRes:Integer;
// змінна для записи результату пошуку
begin
 SelectDirectory('Просмотр каталога',Dir1,Dir2);
 ListBox1.Clear;
 FindRes:=FindFirst(Dir2+'\*.*',faAnyFile,SR);
 while FindRes=0 do
// поки ми знаходимо файли (каталоги),
// то виконувати цикл
// якщо знайдений елемент каталог і
// він має назву "." або "..", тоді:
// продовжити пошук
begin
   if ((SR.Attr and faDirectory)=faDirectory) and
   ((SR.Name='.')or(SR.Name='..')) then
    begin
     FindRes:=FindNext(SR);
     Continue;
// продовжити цикл
    end;
{
якщо у файлу (каталогу) дата створення менше, ніж встановлено в DateTimePicker1,
то продовжити пошук}

   ListBox1.Items.Add(SR.Name+'                  '+Dir2+'              '
     +IntToStr(SR.Size)+'  Bite');
// додавання в список знайденого елемента
   if Application.Terminated then Break;
   Application.ProcessMessages;
   FindRes:=FindNext(SR);
// продовження пошуку за заданими умовами
  end;
 FindClose(SR);
// закриваємо пошук
{
Атрибути шуканого елемента:
  faReadOnly - Файли "тільки читання".
  faHidden - Приховані файли.
  faSysFile - Системні файли.
  faVolumeID - Файл мітки диска (максимум 11 символів).
  faDirectory - Атрибут ознаки каталогу.
```

```
  faArchive - Звичайний файл.
  faAnyFile - Пошук по усім вищесказаним атрибутам.
}
 Label2.Caption:='Items: '+IntToStr(ListBox1.Items.Count);
end;

procedure TForm1.FindFile(Dir:String);
var
 i:integer;
 SR:TSearchRec;
 FindRes:Integer;
 Dir1,Dir2:string;
begin
SelectDirectory('Просмотр каталогу',Dir1,Dir2);
Label4.Caption:='Path: - '+Dir2+'*.*';
 i:=0;
 FindRes:=FindFirst(Dir+'*.*',faAnyFile,SR);
 while FindRes=0 do
  begin
   if ((SR.Attr and faDirectory)=faDirectory) and
   ((SR.Name='.')or(SR.Name='..')) then
     begin
      FindRes:=FindNext(SR);
      Continue;
     end;
   // якщо знайдений каталог, то
   if ((SR.Attr and faDirectory)=faDirectory) then
    begin
     // входимо в процедуру пошуку з параметрами поточного каталогу +
     // каталог, що ми знайшли
     FindFile(Dir+SR.Name+'\');
     FindRes:=FindNext(SR);
     Continue; // продолжить цикл
    end;
   ListBox2.Items.Add(SR.Name+'       -       '+Dir+'       -       '
                                          +IntToStr(SR.Size)+' - Bite');
    if Application.Terminated then Break;
   Application.ProcessMessages;
   FindRes:=FindNext(SR);
  end;
 FindClose(SR);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 ListBox2.Clear;
// очищення списку файлів
 FindFile(Edit2.Text);
// пошук файлів з початковими умовами, заданих в Edit1
 Label3.Caption:='Items: '+IntToStr(ListBox2.Items.Count);
end;

procedure TForm1.Button3Click(Sender: TObject);
var
 s:string;
begin
 s:=Edit3.Text;
 with ListBox1 do
 ItemIndex:=Perform(LB_SELECTSTRING, ItemIndex, LongInt(S));
//////////////////
 with ListBox2 do
 ItemIndex:=Perform(LB_SELECTSTRING, ItemIndex, LongInt(S));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 SendMessage(ListBox1.Handle,LB_SetHorizontalExtent,1000,0);
 SendMessage(ListBox2.Handle,LB_SetHorizontalExtent,1000,0);
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
 Listbox2.Sorted:=true;
 ListBox2.Items.BeginUpdate;
 SendMessage(ListBox2.Handle,LB_SetHorizontalExtent,1000,0);
 ListBox2.Items.EndUpdate;
end;

procedure TForm1.CheckBox2Click(Sender: TObject);
begin
 Listbox1.Sorted:=true;
 SendMessage(ListBox1.Handle,LB_SetHorizontalExtent,1000,0);
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
var
 i:integer;
begin
 for i:=ListBox1.Items.Count-1 downto 0 do
  if ListBox1.Selected[i] then
   begin
    ListBox1.Items.Delete(i);
    ListBox1.Selected[i]:=true;
   end;
 for i:=ListBox2.Items.Count-1 downto 0 do
  if ListBox2.Selected[i] then
   begin
    ListBox2.Items.Delete(i);
    ListBox2.Selected[i]:=true;
   end;
end;

end.
```

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

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

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

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших

комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

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

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

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

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

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

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

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

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

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

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

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

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

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка ПЗ. Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб -сервера. Нарешті, управління даними здійснюється сервером даних.

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

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

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

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

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

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а

також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

Розглянемо 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/апаратне 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 такий 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.

### 4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використати фіналіста конкурсу AES – шифр Rijndael. Він є нетрадиційним блоковим шифром, оскільки не використовує мережу Фейштеля для криптоперетворень. Алгоритм представляє кожний блок кодуємих даних у вигляді двовимірного масиву байт розміром 4x4, 4x6 або 4x8 залежно від установленої довжини блоку. Далі на відповідних етапах перетворення відбуваються або над незалежними стовпцями, або над незалежними рядками, або взагалі над окремими байтами в таблиці. Всі перетворення в шифрі мають строге математичне обґрунтування. Сама структура й послідовність операцій дозволяють виконувати даний алгоритм ефективно як на 16-бітних так і на 64-бітних процесорах. У структурі алгоритму закладена можливість паралельного виконання деяких операцій, що на багатопроцесорних робочих станціях може ще підняти швидкість шифрування в 4 рази. Алгоритм складається з деякої кількості раундів (від 10 до 14 – це залежить від розміру блоку й довжини ключа), у яких послідовно виконуються наступні операції : ByteSub – Таблична підстановка 8x8 біт (рисунок 4.3).

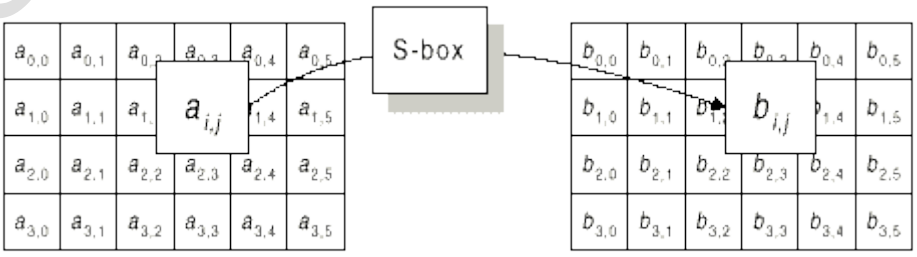


Рисунок 4.3 – Таблична підстановка 8x8 біт

ShiftRow – зрушення рядків у двовимірному масиві на різні зсуви (рисунок 4.4). MixColumn – математичне перетворення, що перемішує дані усередині стовпця (рисунок 4.5). AddRoundKey – додавання матеріалу ключа операцією XOR (рисунок 4.6). В останньому раунді операція перемішування стовпців відсутня, що робіть всю послідовність операцій симетричною.

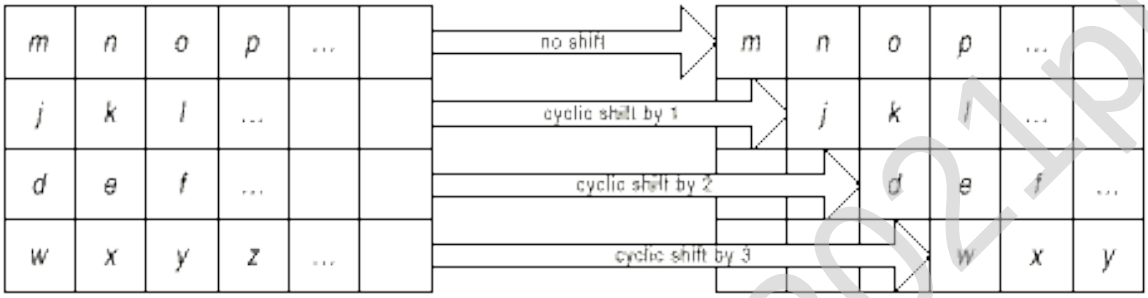


Рисунок 4.4 – Зрушення рядків у двовимірному масиві на різні зсуви

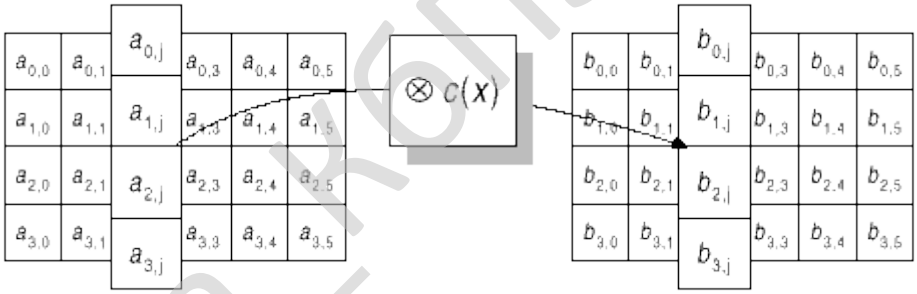


Рисунок 4.5 – Математичне перетворення, що перемішує дані усередині стовпця

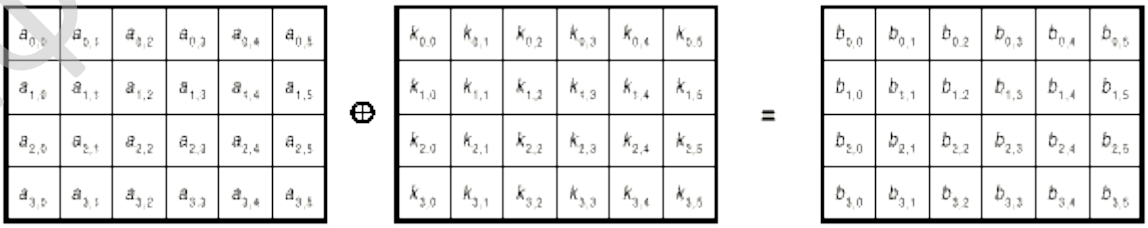

Рисунок 4.6 – Додавання матеріалу ключа операцією XOR

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. З рисунку головного вікна можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок підрозділів ПЗ: NFV; Даних обробки; Форматів обробки.

– Верхнього меню: Файл; Формат; Автозавантаження; Налаштування віртуалізації NFV; Довідка.

– Вспливаючі вікна відповідних підрозділів ПЗ відносно обраних функцій.


Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення. 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.

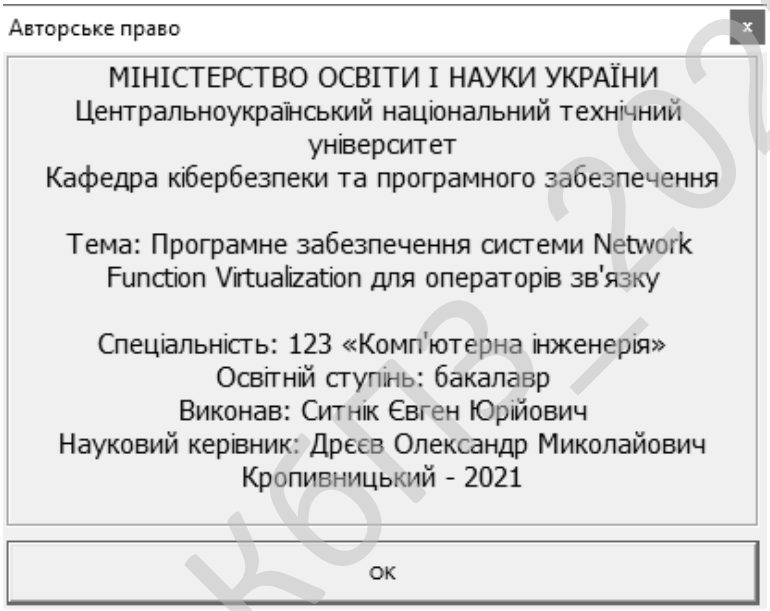


Рисунок 5.2 – Авторське право

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.

# 6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної бакалаврської роботи, призначено для системи Network Function Virtualization для операторів зв’язку.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем Network Function Virtualization для операторів зв’язку.

– Досліджена система Network Function Virtualization для операторів зв’язку.

– На основі отриманих результатів досліджень створена програмна реалізація системи Network Function Virtualization для операторів зв’язку.

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання Network Function Virtualization для операторів зв’язку.

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 Delphi 10.4.1. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи Network Function Virtualization для операторів зв’язку. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на

загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм AES.

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

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Королев А.В. Адаптивная маршрутизация в корпоративных сетях / А.В. Королев, Г.А. Кучук, А.А. Пашнев. – Х.: ХВУ, 2003. – 224 с.

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

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

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

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

6. Мохамад Гани Абу Таам Разработка математической GERT-модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / А.А.Смирнов, Мохамад Гани Абу Таам // Информационные системы в управлении, образовании, промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2014. – 498 с.

7. Мохамад Гани Абу Таам Метод управления доступом в интеллектуальных узлах коммутации / Мохамад Гани Абу Таам, А.А.Смирнов // Информационные технологии и защита информации в информационно-коммуникационных системах: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2015. – 486 с.

8. Мохамад Гани Абу Таам Математическая GERT-модель технологии передачи метаданных в облачные антивирусные системы / В.В.Босько,

А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Збірник наукових праць "Системи обробки інформації". – Випуск 1(117). – Х.: ХУПС – 2014. – С. 137-141.

9. Мохамад Гани Абу Таам Структурно-логическая GERT-модель технологии распространения компьютерных вирусов / А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Системи управління, навігації та зв'язку. – Випуск 1(29). – П.: ПНТУ. – 2014. – С. 120-125.

10. Мохамад Гани Абу Таам Сравнительные исследования математических моделей технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, А.В. Коваленко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 9(125). – Х.: ХУПС – 2014. – С. 105-110.

11. Мохамад Гани Абу Таам Математическая модель интеллектуального узла коммутации с обслуживанием информационных пакетов различного приоритета / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 4 (41). – Харків: ХУПС. – 2014. – С. 48-52.

12. Мохамад Гани Абу Таам Исследование показателей качества функционирования интеллектуальных узлов коммутации в телекоммуникационных системах и сетях / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 4(17). – Харків: ХУПС. – 2014. – С.90-95.

13. Мохамад Гани Абу Таам Усовершенствованный алгоритм управления доступом к «облачным» телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 1(126). – Х.: ХУПС – 2015. – С. 150-153.

14. Мохамад Гани Абу Таам Анализ и исследование методов управления сетевыми ресурсами для обеспечения антивирусной защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Системи озброєння і військова техніка. – Випуск 3(43) – Х.: ХУПС – 2015. – С. 100-107.

15. Мохамад Гани Абу Таам Исследование эффективности метода управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 3(19). – Х.: ХУПС. – 2015. – С. 134-141.

16. Mohamad Abou Taam Method of controlling access to intellectual switching nodes of telecommunication networks and systems / A.A. Smirnov, Mohamad Abou Taam, S.A. Smirnov // International Journal of Computational Engineering Research (IJCER). – Volume 5, Issue 5. – India. Delhi. – 2015. – P. 1-7.

17. Мохамад Гани Абу Таам GERT-модель технологии передачи данных в облачные антивирусные системы / А.А. Смирнов, В.В. Босько, Мохамад Гани Абу Таам // Збірник тез доповідей науково-практичної конференції «Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку». м. Харків. 12-13 березня 2014 р. – Харків. АВВ МВС. – 2014. – С. 18-19.

18. Мохамад Гани Абу Таам Математическое моделирование технологии передачи сигнатур в облачные антивирусные системы / Мохамад Гани Абу Таам, А.А. Смирнов // Збірник тез VI міжнародної науково-практичної конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 17-18 квітня 2014 р. – Харків: ХНЕУ. – 2014. – С. 260.

19. Мохамад Гани Абу Таам Анализ требований к качеству обслуживания в информационно-телекоммуникационных системах / А.А. Смирнов, Мохамад Гани Абу Таам // Збірник тез XVI міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 11-12 квітня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 124-126.

20. Мохамад Гані Абу Таам Дослідження та реалізація GERT-моделі технології розповсюдження комп’ютерних вірусів для захисту телекомунікаційних систем / Мохамад Гані Абу Таам, С.А. Смирнов // Збірник

тез науково-практичної конференції «Інформаційні технології та комп'ютерна інженерія». м. Кіровоград. 4 грудня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 168.

21. Мохамад Гани Абу Таам Исследование математических моделей технологии распространения компьютерных вирусов / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник наукових праць міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 25-28 лютого 2015 р. – Київ: Європейський університет. – 2015. – С. 90-91.

22. Мохамад Гани Абу Таам Метод управления доступом к «облачным» ресурсам для защиты телекоммуникационных систем / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез всеукраїнської науково-практичної конференції «Інформаційна безпека держави, суспільства та особистості». м. Кіровоград. 16 квітня 2015. – Кіровоград: КНТУ. – 2015. – С. 50-52.

23. Мохамад Гани Абу Таам Разработка метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник тез VII міжнародної науково-практичної конференції "Проблеми і перспективи розвитку ІТ-індустрії". м. Харків. 17-18 квітня 2015 р. – Харків: ХНЕУ. – 2015. – С. 14.

24. Мохамад Гани Абу Таам Реализация метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам // Збірник тез XVII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 17-18 квітня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 91-92.

25. Мохамад Гани Абу Таам Реализация математической модели интеллектуального узла коммутации для обеспечения защищенности телекоммуникационной сети / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез II Міжнародної науково-практичної Інтернет-конференції «Інформаційна та економічна безпека» (INFECO-2015)». м. Харків. 21-22 травня 2015 р. – Харків: ХІБС УБС НБУ. – 2015. – С. 20-24.

26. Мохамад Гани Абу Таам Разработка математической модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Сборник тезисов XI международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 01 – 06 июня 2015 г – Варна. ТУВ. – 2015. – С. 488-491

27. Мохамад Гани Абу Таам Метод управления доступом к облачным телекоммуникационным ресурсам для обеспечения защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Комп'ютерні технології та інформаційна безпека». м. Кіровоград. 2-3 липня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 4-5.

28. Мохамад Гани Абу Таам Имитационная модель системы управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез першої всеукраїнської науково-практичної конференції «Перспективні напрями захисту інформації». м. Затока. 7-9 вересня 2015 р. – Одеса: ОНАЗ. – 2015. – С. 90-94.

29. МСЭ-Т Рекомендация G.101. Международные телефонные соединения и цепи – Общие определения //11/2003. [Електронний ресурс]. – Режим доступу до ресурсу: http://www. telecom61.ru›SharedFiles/Download.aspx? …pageid=106

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

31. 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.

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

33. 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.

34. 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

35. 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

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

37. Семенов С.Г. Математическая модель мультисервисного канала связи на основе экспоненциальной GERT-сети / С.Г. Семенов, Є.В. Мелешко, Я.В. Ілюшко // Системи озброєння і військова техніка. – Х.:ХУ ПС. – 2011. –Вип. 3(27). – С. 64-67.

38. Семенов С.Г. Математична модель системи криптографічного захисту електронних повідомлень на основі GERT-мережі / С.Г. Семенов, О.О. Сур // Системи управління, навігації та зв'язку. – К.:ЦНДІ навігації і управління. – 2012. – Том 1. Вип. 1(21). – С. 131-137

39. Семенов С.Г. Исследования вероятностно-временных характеристик мультисервисного канала связи с использованием математического аппарата GERT-сети / С.Г. Семенов, В.В. Босько, І.А. Березюк // Системи обробки інформації. – Х.: ХУ ПС. – 2012. – Том 1. Вип. 3(101). – С. 139-142.

40. Семенов С.Г. Моделирование защищенного канала связи с использованием экспоненциальной GERT-сети / С.Г. Семенов,

А.А. Можаев // Информатика, математическое моделирование, экономика. – Смоленськ.: Смоленский филиал АНО ВПО ЦС РФ "Российский университет кооперации". – 2012. – Том.1. – С. 152-160.

41. Семенов С.Г. Методика математического моделирования защищенной ИТС на основе многослойной GERT-сети / С.Г. Семенов // Вісник Національного технічного університету «Харківський політехнічний інститут». – Х.:НТУ «ХПІ». – 2012. –№62 (968). – С 173-181.

42. Семенов С.Г. Защита данных в компьютеризированных управляющих системах / С.Г. Семенов, В.В. Давыдов, С.Ю. Гавриленко. – LAP Lambert Academic Publishing GmbH & Co. KG (Саарбрюккен, Германия), 2014. – 236 с.

43. 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.

44. 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.

45. 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.

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

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

48. 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.

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

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

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

52. Elwalid Routing and Protection in GMPLS Networks: From Shortest Paths to Optimized Designs / A. Elwalid, D. Mitra, I. Saniee, and I. Widjaja. //Journal of lightwave technology. – 2003. – №21(11), P. 2828-28-38.

53. A.B. Bagula Online Traffic Engineering: The Least Interference Optimization Algorithm / A.B. Bagula, M. Botha, and A.E Krzesinski. // IEEE Communications Society – 2004, P. 1232-1236.

54. Anees. Shaikh Evaluating the Impact of Stale Link State on Quality-of-Service Routing / Anees Shaikh, Jennifer Rexford, and Kang G. Shin. // IEEE/ACM Transactions on Networking. – 2001. – №9(2), P. 162-176.

Додаток А
(обов'язковий)

**Технічне завдання**

**Зміст**


1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи Network Function Virtualization для операторів зв’язку.

## 2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 204-02 від 28.12.2020 року).

## 3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи Network Function Virtualization для операторів зв’язку.

## 4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи 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;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

### 5.2 Показники призначення

Система повинна забезпечувати:

– системи Network Function Virtualization для операторів зв’язку;

– 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 забезпечення.

### 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 Delphi 10.4.1.

### 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 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 81 аркуш.

## 8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 5.06.2021 р.

Додаток Б

(обов’язковий)

**Міністерство освіти і науки України**

**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ Дрєєв О.М.

***Програмне забезпечення системи Network Function Virtualization для операторів зв’язку***

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 64

Літера: РП

Кропивницький – 2021 року

**Hopf.pas - Блок оптимізації структури системи Network Function Virtualization для операторів зв'язку**

```
unit Hopf;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, Neural_network_Comp, Neural_network_Types, Db, DBTables, ExtCtrls,
DBCtrls, StdCtrls,
  ToolWin, ComCtrls;

type
  TForm1 = class(TForm)
    Table: TTable;
    btnExecute: TButton;
    DBNavigator: TDBNavigator;
    DataSource: TDataSource;
    btnEdit: TButton;
    stgDatabase: TStringGrid;
    stgInput: TStringGrid;
    stgOutput: TStringGrid;
    NeuralNetHopf: TNeuralNetHopf;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    Bevel: TBevel;
    TableLETTERS: TStringField;
    dbMain: TDatabase;
    procedure DataSourceDataChange(Sender: TObject; Field: TField);
    procedure FormActivate(Sender: TObject);
    procedure GridClick(Sender: TObject);
    procedure btnEditClick(Sender: TObject);
    procedure btnExecuteClick(Sender: TObject);
    procedure GridDrawCell(Sender: TObject; ACol, ARow: Integer;
      Rect: TRect; State: TGridDrawState);
    procedure FormCreate(Sender: TObject);
  public
    { Public declarations }
    procedure AddPattern(Value: string);
    procedure Clear(Grid: TStringGrid);
    procedure Init;
    procedure ShowMatrix(Grid: TStringGrid; Value: string);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

// Показати вектор у вигляді сітки
procedure TForm1.ShowMatrix(Grid: TStringGrid; Value: string);
var
  i, j: integer;
begin
  Clear(Grid);
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
    begin
      try
        if Value[i * Grid.RowCount + j + 1] = '1' then
          Grid.Cells[i, j] := '1'
        else
```

```
          Grid.Cells[i, j] := ' '
      except
        Grid.Cells[i, j] := ' '
      end;
    end;
end;

// Очистити сітку
procedure TForm1.Clear(Grid: TStringGrid);
var
  i, j: integer;
begin
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
      Grid.Cells[i, j] := ' ';
end;

// Показати символ з таблиці
procedure TForm1.DataSourceDataChange(Sender: TObject; Field: TField);
begin
  ShowMatrix(stgDatabase, TableLETTERS.Value);
end;

// Ініціалізація мережі для системи Network Function Virtualization для
операторів зв’язку значеннями з таблиці
procedure TForm1.Init;
begin
  // Очистити мережу від зразків
  NeuralNetHopf.ResetPatterns;

  // Додати зразки з таблиці до мережі для системи Network Function
Virtualization для операторів зв’язку
  Table.First;
  while not Table.Eof do
  begin
    AddPattern(TableLETTERS.AsString);
    Table.Next;
  end;
  Table.First;

  // Ініціалізувати ваги
  NeuralNetHopf.InitWeights;
  // Мережа підготовлена до розпізнавання
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  Clear(stgDatabase);
  Clear(stgInput);
  Clear(stgOutput);
  Init;
end;

procedure TForm1.GridClick(Sender: TObject);
begin
  with Sender as TStringGrid do
    if Cells[Col, Row] = '1' then
      Cells[Col, Row] := ' '
    else
      Cells[Col, Row] := '1'
end;

// Додавання нового образа до мережі для системи Network Function Virtualization
для операторів зв’язку
procedure TForm1.AddPattern(Value: string);
var
  i: integer;
  xVector: TVectorInt;
begin
```

```
  SetLength(xVector, stgDatabase.RowCount * stgDatabase.ColCount);

  // Перетворення символьного рядка у вектор
  for i := 1 to stgDatabase.RowCount * stgDatabase.ColCount do
    try
      if TableLETTERS.AsString[i] = '1' then
        xVector[i - 1] := 1
      else
        xVector[i - 1] := -1;
    except
      xVector[i - 1] := -1;
    end;


  NeuralNetHopf.AddPattern(xVector);
end;

procedure TForm1.btnEditClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  xString := '';
  for i := 0 to stgDatabase.ColCount - 1 do
    for j := 0 to stgDatabase.RowCount - 1 do
      if stgDatabase.Cells[i, j] = '1' then
        xString := xString + '1'
      else
        xString := xString + ' ';
  Table.Edit;
  TableLETTERS.AsString := xString;
  Table.Post;
end;

// Розпізнавання символу
procedure TForm1.btnExecuteClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  // Подаємо сигнали на вихід мережі для системи Network Function Virtualization
для операторів зв’язку
  for i := 0 to stgInput.ColCount - 1 do
    for j := 0 to stgInput.RowCount - 1 do
      if stgInput.Cells[i, j] = '1' then
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := 1
      else
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := -1;

  // Запуск процесу розпізнавання
  NeuralNetHopf.Calc;

  // Перетворення виходів мережі для системи Network Function Virtualization для
операторів зв’язку до рядка
  xString := '';
  for i := 1 to stgOutput.RowCount * stgOutput.ColCount do
    if NeuralNetHopf.Layers[1].Neurons[i - 1].Output = 1 then
      xString := xString + '1'
    else
      xString := xString + ' ';

  // Відобразити результат
  ShowMatrix(stgOutput, xString);
end;

procedure TForm1.GridDrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
begin
  with Sender as TStringGrid do
```

```
  begin
    Canvas.Brush.Color := clBlack;
    if Cells[ACol,ARow] <> ' ' then
      Canvas.FillRect(Rect)
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  dbMain.Params.Values['Path'] := ExtractFilePath(Application.ExeName);
  dbMain.Open;
  Table.Open;

end;

end.
```

**Neural_network_Editor.pas – розробка нейронних мереж для системи Network Function Virtualization для операторів зв'язку**

```
unit Neural_network_Editor;

interface

uses
  Classes,
  DesignIntf, DesignEditors,
  Neural_network_Comp, Neural_network_EditorForm,
  SysUtils, Dialogs, Controls, Neural_network_EditorFieldsForm;

type
  TNeuronsInLayerFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

  TFileNameFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    procedure Edit; override;
  end;

  TOptionsFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

procedure Register;

implementation

function TOptionsFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TOptionsFieldsProperty.Edit;
var
  i: integer;
  xNeuralNetExtended: TNeuralNetExtended;
  frmNeuroFields: TfrmNeuroFields;
  xCurrent: integer;
begin
  frmNeuroFields := TfrmNeuroFields.Create(nil);
  try
    with frmNeuroFields do
    begin
      xNeuralNetExtended := (GetComponent(0) as TNeuralNetExtended);
      for i := 0 to xNeuralNetExtended.AvailableFieldsCount - 1 do
        ltbFieldName.Items.Add(xNeuralNetExtended.Fields[i].Name);
      xCurrent := 0;
      rdgFieldType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].Kind;
      rdgNormType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].NormType;
      edtAlpha.Text := FloatToStr(xNeuralNetExtended.Fields[xCurrent].Alpha);
      sttMin.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMin);
      sttMax.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMax);
      NeuralNetExtended := xNeuralNetExtended;
```

```
      ShowModal;
    end;
  except
    frmNeuroFields.Free;
  end;
end;

function TOptionsFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TNeuralNetExtended) do
    if AvailableFieldsCount > 1 then
    begin
      for i := 0 to AvailableFieldsCount - 1 do
        Result := Result + Fields[i].Name + ',';
      Result[Length(Result)] := ']'
    end
    else
      Result[Length(Result) + 1] := ']';
end;

function TNeuronsInLayerFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

function TNeuronsInLayerFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TNeuralNetBP) do
    if LayerCount > 0 then
    begin
      for i := 0 to LayerCount - 1 do
        Result := Result + IntToStr(LayersBP[i].NeuronCount) + ',';
      Result[Length(Result)] := ']';
    end
    else
      Result[Length(Result) + 1] := ']';
end;

procedure TNeuronsInLayerFieldsProperty.Edit;
var
  i: integer;
  xPreviousCount: integer;
  xNeuralNetBP: TNeuralNetBP;
  frmNeuronsInLayer: TfrmNeuronsInLayer;
  xChangesMade: boolean;
begin
  xChangesMade := False;
  frmNeuronsInLayer := TfrmNeuronsInLayer.Create(nil);
  try
    with frmNeuronsInLayer do
    begin
      xNeuralNetBP := (GetComponent(0) as TNeuralNetBP);
      speLayers.Value := xNeuralNetBP.LayerCount;
      stgNeuronsInLayer.RowCount := xNeuralNetBP.LayerCount + 1;
      for i := 0 to xNeuralNetBP.LayerCount - 1 do
      begin
        stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
        stgNeuronsInLayer.Cells[1, i + 1] :=
IntToStr(xNeuralNetBP.LayersBP[i].NeuronCount);
      end;
      if ShowModal = mrOk then
```

```
      begin
        xNeuralNetBP.ResetLayers;
        if speLayers.Value = 0 then
        begin
          xNeuralNetBP.ResetLayers;
          Exit;
        end;
        if xNeuralNetBP.LayerCount <> speLayers.Value then
          xChangesMade := True
        else
          for i := 0 to xNeuralNetBP.LayerCount - 1 do
            if xNeuralNetBP.LayersBP[i].NeuronCount <>
StrToInt(stgNeuronsInLayer.Cells[1, i + 1]) then
            begin
              xChangesMade := True;
              Break;
            end;
        if xChangesMade then
        begin
          if xNeuralNetBP.LayerCount = speLayers.Value then
            for i := 0 to speLayers.Value - 1 do
              xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1]
          else
            if xNeuralNetBP.LayerCount < speLayers.Value then
            begin
              for i := 0 to xNeuralNetBP.LayerCount - 1 do
                xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1];
              for i := xNeuralNetBP.LayerCount to speLayers.Value - 1 do
                xNeuralNetBP.AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i +
1]));
            end
            else
            begin
              if speLayers.Value > 0 then
                for i := 0 to speLayers.Value - 1 do
                  xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i
+ 1];
              xPreviousCount := xNeuralNetBP.LayerCount -  speLayers.Value;
              for i := 1 to xPreviousCount do
                xNeuralNetBP.DeleteLayer(xNeuralNetBP.LayerCount - 1);
            end;
          if not xNeuralNetBP.AutoInit then
            xNeuralNetBP.AutoInit := True;
        end;
      end;
    end;
  except
    frmNeuronsInLayer.Free;
  end;
end;

function TFileNameFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TFileNameFieldsProperty.Edit;
var
  xOpenDialog: TOpenDialog;
begin
  xOpenDialog := TOpenDialog.Create(nil);
  if UpperCase(GetName) = 'FILENAME' then
    xOpenDialog.Filter := 'Нейронна мережа(*.nnw)|*.nnw|всі файли (*.*)|*.*';
  if UpperCase(GetName) = 'SOURCEFILENAME' then
    xOpenDialog.Filter := 'Текстові файли (*.txt)|*.txt|всі файли (*.*)|*.*';

  if xOpenDialog.Execute then
```

```
  begin
    if UpperCase(GetName) = 'FILENAME' then
      (GetComponent(0) as TNeuralNetExtended).FileName := xOpenDialog.FileName;
    if UpperCase(GetName) = 'SOURCEFILENAME' then
      (GetComponent(0) as TNeuralNetExtended).SourceFileName :=
xOpenDialog.FileName;
  end;
  xOpenDialog.Free;
end;

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(TStrings), TNeuralNetBP, 'NeuronsInLayer',
TNeuronsInLayerFieldsProperty);
  RegisterPropertyEditor(TypeInfo(TFileName), TNeuralNetExtended, '',
TFileNameFieldsProperty);
  RegisterPropertyEditor(TypeInfo(string), TNeuralNetExtended, 'Options',
TOptionsFieldsProperty);
end;

end.
```

**Neural_network_EditorForm.pas – редактор нейронних мереж для системи Network Function Virtualization для операторів зв'язку**

```
unit Neural_network_EditorForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, StdCtrls, ExtCtrls, Neural_network_Comp, Spin, Neural_network_Types;

type
  TfrmNeuronsInLayer = class(TForm)
    Bevel1: TBevel;
    btnOk: TButton;
    btnCancel: TButton;
    stgNeuronsInLayer: TStringGrid;
    Label1: TLabel;
    speLayers: TSpinEdit;
    procedure stgNeuronsInLayerGetEditMask(Sender: TObject; ACol,
      ARow: Integer; var Value: String);
    procedure stgNeuronsInLayerSetEditText(Sender: TObject; ACol,
      ARow: Integer; const Value: String);
    procedure speLayersChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmNeuronsInLayer: TfrmNeuronsInLayer;

implementation

{$R *.DFM}

procedure TfrmNeuronsInLayer.stgNeuronsInLayerGetEditMask(Sender: TObject;
  ACol, ARow: Integer; var Value: String);
begin
  Value := '0000';
end;

procedure TfrmNeuronsInLayer.stgNeuronsInLayerSetEditText(Sender: TObject;
  ACol, ARow: Integer; const Value: String);
begin
{  with stgNeuronsInLayer do
    try
      Cells[ACol, ARow] := IntToStr(StrToInt(trim(Value)));
    except
      Cells[ACol, ARow] := IntToStr(DefaultNeuronCount);
    end;}
end;

procedure TfrmNeuronsInLayer.speLayersChange(Sender: TObject);
var
  i: integer;
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
  for i := 1 to stgNeuronsInLayer.RowCount do
    if trim(stgNeuronsInLayer.Cells[1, i]) = '' then
    begin
      stgNeuronsInLayer.Cells[0, i] := IntToStr(i);
      stgNeuronsInLayer.Cells[1, i] := IntToStr(DefaultNeuronCount);
    end;
end;
```

```
procedure TfrmNeuronsInLayer.FormCreate(Sender: TObject);
begin
  stgNeuronsInLayer.Cells[0,0] := '# шару';
  stgNeuronsInLayer.Cells[1,0] := 'нейронів';
end;

end.
```

**Neural_network_EditorFieldsForm.pas – редактор властивостей полів нейроної мережі для системи Network Function Virtualization для операторів зв'язку**

```
unit Neural_network_EditorFieldsForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Neural_network_Comp;

type
  TfrmNeuroFields = class(TForm)
    ltbFieldName: TListBox;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    Bevel1: TBevel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    sttMin: TStaticText;
    sttMax: TStaticText;
    edtAlpha: TEdit;
    btnOk: TButton;
    btnCancel: TButton;
    Bevel2: TBevel;
    Label4: TLabel;
    procedure ltbFieldNameClick(Sender: TObject);
    procedure rdgFieldTypeClick(Sender: TObject);
    procedure rdgNormTypeClick(Sender: TObject);
    procedure edtAlphaChange(Sender: TObject);
  private
    { Private declarations }
  public
    NeuralNetExtended: TNeuralNetExtended;
    { Public declarations }
  end;

var
  frmNeuroFields: TfrmNeuroFields;

implementation

{$R *.DFM}

procedure TfrmNeuroFields.ltbFieldNameClick(Sender: TObject);
begin
  rdgFieldType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  rdgNormType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  edtAlpha.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  sttMin.Caption :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin);
  sttMax.Caption :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax);
end;

procedure TfrmNeuroFields.rdgFieldTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
```

```
  else
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        NeuralNetExtended.Fields[i].Kind := rdgFieldType.ItemIndex;
end;

procedure TfrmNeuroFields.rdgNormTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
  else
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        NeuralNetExtended.Fields[i].NormType := rdgNormType.ItemIndex;
end;

procedure TfrmNeuroFields.edtAlphaChange(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha :=
StrToFloat(edtAlpha.Text);
end;

end.
```

**Neural_network_Types.pas – ініціалізація базових констант**

```
unit Neural_network_Types;

interface

const

  DefaultAlpha = 1;              // Параметр активаційної функції
  DefaultEpochCount = 10000;     // Кількість етапів для навчання
  DefaultErrorValue = 0.05;      // Помилка за замовчуванням для всіх видів
  DefaultHopfLayerCount = 2;     // Кількість шарів у мережі для системи Network
Function Virtualization для операторів зв’язку Хопфилда
  DefaultLayerCount = 0;         // Мінімальна кількість шарів у мережі для
системи Network Function Virtualization для операторів зв’язку back-propagation
  DefaultMaxIterCount = 10;      // Максимальна кількість ітерацій в алгоритмі
Хопфилда
  DefaultMomentum = 0.9;         // Імпульс - момент
  DefaultNeuronCount = 0;        // Кількість нейронів у прихованому шарі за
замовчуванням
  DefaultPatternCount = 0;       // Кількість прикладів
  DefaultTeachRate = 0.1;        // Швидкість навчання
  DefaultTeachIdentCount = 100;  // Необхідний відсоток розпізнаних прикладів з
навчальної множини
  DefaultTestIdentCount = 100;   // Необхідний відсоток розпізнаних прикладів з
тестової множини
  DefaultUseForTeach = 100;      // Відсоток прикладів використовуваних як
навчальна множина
  DefaultDeltaBarAcceleratingConst = 0.095;
  DefaultDeltaBarDecFactor = 0.85;
  DefaultRPropInitValue = 0.1;
  DefaultRPropMaxStepSize = 50;
  DefaultRPropMinStepSize = 1 E-6;
  DefaultRPropDecFactor = 0.5;
  DefaultRPropIncFactor = 1.2;
  DefaultSuperSABDecFactor = 0.5;
  DefaultSuperSABIncFactor = 1.05;

  SensorLayer = 0;               // Сенсорний шар
  BiasNeuron = 1;                // Зсув у мережі для системи Network Function
Virtualization для операторів зв’язку back-propagation

  Separators = ['.', ','];
  Letters    = ['a'..'z'];
  Capitals   = ['A'..'Z'];
  DigitChars = ['0'..'9'];
  SpaceChar = #9;

resourcestring

  SFieldNorm = 'Не існує типу нормалізації %d';
  SFieldKind = 'Не існує типу поля %d';
  SFieldIndexRange = 'Неправильно зазначений номер поля %d';
  SInFieldCount = 'Неправильно встановлена кількість вхідних полів';
  SInNeuronCount = 'Неправильно встановлена кількість вхідних нейронів';
  SInVectorCount = 'Неправильно встановлена розмірність вхідного вектора';
  SLayerRangeIndex = 'Неправильно зазначений номер шару %d';
  SNeuronRangeIndex = 'Неправильно зазначений номер нейрона %d';
  SNeuronCount = 'Неправильно зазначена кількість нейронів';
  SOutFieldCount = 'Неправильно встановлена кількість вихідних полів';
  SOutNeuronCount = 'Неправильно встановлена кількість вихідних нейронів';
  SOutVectorCount = 'Неправильно встановлена розмірність вихідного вектора';
  SPatternRangeIndex = 'Вихід за межі масиву прикладів %d';
  SStreamCannotRead = 'Помилка читання з потоку';
  SWeightRangeIndex = 'Неправильно зазначений номер ваги %d';
  SWrongFileName = 'Неправильно зазначене ім'я файлу %s';
  SCannotBeNumber = 'Помилка, вираз %s неможливо привести до числового типу';
```

```
  SBPStopCondition = 'Не задана умова зупинки процесу навчання';

type

  TVectorInt = array of integer;
  TVectorFloat = array of double;
  TVectorString = array of string;
  TMatrixInt = array of array of integer;
  TMatrixFloat = array of array of double;
  TNormalize = (nrmLinear, nrmSigmoid, nrmAuto,  nrmNone,
                nrmLinearOut, nrmAutoOut);
  TNeuroFieldType = (fdInput, fdOutput, fdNone);

implementation


end.
```

**Neural_network_Comp.pas – формування бібліотеки шаблонів та дослідження нейронних мереж для системи Network Function Virtualization для операторів зв'язку**

```
unit Neural_network_Comp;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, PumpData, Neural_network_Types, IniFiles, Math;

type

  // Класи виключень
  EInOutDimensionError = class(Exception);
  ENeuronCountError  = class(Exception);
  ENeuronNotEqualFieldError  = class(Exception);
  EBPStopCondition  = class(Exception);

  // Процедурні типи
  TActivation = function (Value: double): double of object;

  // Упереджуюче оголошення класів
  TNeuron = class;
  TLayer = class;

  // Базовий клас нейрона

  TNeuron = class(TObject)
  private
    FOutput: double;
    // Вектор ваг
    FWeights: TVectorFloat;
    // Покажчик на шар, у якому перебуває нейрон
    Layer: TLayer;
    function GetWeights(Index: integer): double;
    procedure SetWeights(Index: integer; Value: double);
    procedure SetWeightCount(const Value: integer);
  public
    constructor Create(ALayer: TLayer); virtual;
    destructor Destroy; override;
    // Ініціалізація ваг
    procedure InitWeights;  virtual;
    // Зважена сума
    procedure ComputeOut(const AInputs: TVectorFloat); virtual;
    property Output: double read FOutput write FOutput;
    property WeightCount: integer  write SetWeightCount;
    property Weights[Index: integer]: double read GetWeights write SetWeights;
  end;

  // Клас нейрона для мережі для системи Network Function Virtualization для
операторів зв'язку Хопфілда

  TNeuronHopf = class(TNeuron)
  public
    procedure ComputeOut(const AInputs: TVectorFloat); override;
  end;

  // Клас нейрона для мережі для системи Network Function Virtualization для
операторів зв'язку

  TNeuronBP = class(TNeuron)
  private
    // Локальна помилка
    FDelta: double;
    // Значення швидкості навчання на попередньому етапі
```

```
    FLearningRate: TVectorFloat;
    // Значення частинної похідної на попередньому етапі
    FPrevDerivative: TVectorFloat;
    // Значення корекції ваги на попередньому етапі
    FPrevUpdate: TVectorFloat;
    // Функція активації
    FOnActivation: TActivation;
    // Похідна функції активації
    FOnActivation: TActivation;
    function GetPrevUpdate(Index: integer): double;
    function GetPrevDerivative(Index: integer): double;
    function GetLearningRate(Index: integer): double;
    function GetPrevUpdateCount: integer;
    procedure SetPrevDerivative(Index: integer; const Value: double);
    procedure SetPrevDerivativeCount(const Value: integer);
    procedure SetDelta(Value: double);
    procedure SetPrevUpdate(Index: integer; Value: double);
    procedure SetPrevUpdateCount(const Value: integer);
    procedure SetLearningRate(Index: integer; const Value: double);
    procedure SetLearningRateCount(const Value: integer);
  public
    destructor Destroy; override;
    procedure ComputeOut(const AInputs: TVectorFloat); override;
    property Delta: double read FDelta write SetDelta;
    property LearningRate[Index: integer]: double read GetLearningRate write
SetLearningRate; //
    property LearningRateCount: integer write SetLearningRateCount;
    property PrevDerivativeCount: integer write SetPrevDerivativeCount;
    property PrevDerivative[Index: integer]: double read GetPrevDerivative write
SetPrevDerivative; //
    property PrevUpdateCount: integer read GetPrevUpdateCount write
SetPrevUpdateCount;
    property PrevUpdate[Index: integer]: double read GetPrevUpdate write
SetPrevUpdate;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
  end;

  // Базовий клас шару
  TLayer = class(TPersistent)
  private
    FNumber: integer;
    // Розмірність NeuronCount
    FNeurons: array of TNeuron;
    function GetNeurons(Index: integer): TNeuron;
    function GetNeuronCount: integer;
    procedure SetNeurons(Index: integer; Value: TNeuron);
    procedure SetNeuronCount(Value: integer);
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); virtual;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property Neurons[Index: integer]: TNeuron read GetNeurons write SetNeurons;
    property NeuronCount: integer read GetNeuronCount write SetNeuronCount;
  end;

  // Клас шару для мережі для системи Network Function Virtualization для
операторів зв’язку Хопфілда
  TLayerHopf = class(TLayer)
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
  end;

  // Клас шару для мережі для системи Network Function Virtualization для
операторів зв’язку
  TLayerBP = class(TLayer)
  private
    function GetNeuronsBP(Index: integer): TNeuronBP;
    procedure SetNeuronsBP(Index: integer; Value: TNeuronBP);
```

```
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property NeuronsBP[Index: integer]: TNeuronBP read GetNeuronsBP write
SetNeuronsBP;
  end;

  // Базовий клас мережі для системи Network Function Virtualization для
операторів зв'язку
  TNeuralNet = class(TComponent)
  private
    // Масив шарів
    FLayers: array of TLayer;
    // Число вибірок
    FPatternCount: integer;
    // Розмірність FPatternCount, InputNeuronCount
    FPatternsInput: TMatrixFloat;
    // Розмірність FPatternCount, OutputNeuronCount
    FPatternsOutput: TMatrixFloat;
    function GetLayers(Index: integer): TLayer;
    function GetOutputNeuronCount: integer;
    function GetPatternsOutput(PatternIndex: integer; OutputIndex: integer):
double;
    function GetPatternsInput(PatternIndex: integer; InputIndex: integer):
double;
    procedure SetLayers(Index: integer; Value: TLayer);
    procedure SetPatternsInput(PatternIndex: integer; InputIndex: integer;
Value: double);
    procedure SetPatternsOutput(PatternIndex: integer; InputIndex: integer;
Value: double);
  protected
    function GetLayerCount: integer; virtual;
    function GetInputNeuronCount: integer; virtual;
    procedure Clear; virtual;
    procedure ResizeInputDim; virtual;
    procedure ResizeOutputDim; virtual;
    procedure SetPatternCount(const Value: integer); virtual;
    procedure SetLayerCount(Value: integer); virtual;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  public
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); virtual; abstract;
    procedure AddPattern(const AInputs: TVectorFloat; const AOutputs:
TVectorFloat); overload; virtual;
    procedure DeleteLayer(Index: integer); virtual; abstract;
    procedure DeletePattern(Index: integer); virtual;
    procedure Init(const ANeuronsInLayer: TVectorInt); overload; virtual;
    property InputNeuronCount: integer read GetInputNeuronCount;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Layers[Index: integer]: TLayer read GetLayers write SetLayers;
    property OutputNeuronCount: integer read GetOutputNeuronCount;
    property PatternsInput[PatternIndex: integer; InputIndex: integer]: double
 read GetPatternsInput write SetPatternsInput;
    property PatternsOutput[PatternIndex: integer; InputIndex: integer]: double
read GetPatternsOutput write SetPatternsOutput;
    procedure ResetPatterns; virtual;
  end;

  // Клас мережі для системи Network Function Virtualization для операторів
зв'язку Хопфілда
  TNeuralNetHopf = class(TNeuralNet)
  private
    FAutoInit: boolean;
    FInputNeuronCount: integer;
    FMaxIterCount: integer;
    FPatternCount: integer;
    FPatterns: TMatrixInt;
    FOnAfterInit: TNotifyEvent;
```

```
    FOnBeforeInit: TNotifyEvent;
    FOnPatternRecognized: TNotifyEvent;
    function GetInput(Index: integer): double;
    function GetPatterns(InputIndex: integer; PatternIndex: integer): integer;
    function Stabled: boolean;
    procedure SetInput(Index: integer; Value: double);
    procedure SetPatterns(InputIndex: integer; PatternIndex: integer; Value:
integer);
  protected
    function GetInputNeuronCount: integer; override;
    function GetLayerCount: integer; override;
    procedure SetInputNeuronCount(Value: integer);
    procedure SetPatternCount(const Value: integer); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddPattern(const ANewPattern: TVectorInt); reintroduce; overload;
    procedure Calc; virtual;
    procedure DeletePattern(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure InitWeights; virtual;
    procedure ResetPatterns; override;
    procedure ResizePatternsDim; virtual;
    property Input[Index: integer]: double read GetInput write SetInput;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Patterns[InputIndex: integer; PatternIndex: integer]: integer read
GetPatterns write SetPatterns;
  published
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property InputNeuronCount: integer read GetInputNeuronCount write
SetInputNeuronCount;
    property MaxIterCount: integer read FMaxIterCount write FMaxIterCount;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnPatternRecognized: TNotifyEvent read FOnPatternRecognized write
FOnPatternRecognized;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  end;

  // Клас мережі для системи Network Function Virtualization для операторів
зв’язку
  TNeuralNetBP = class(TNeuralNet)
  private
    // Коефіцієнт крутості граничної сигмоїдальної функції
    FAlpha: double;
    // Прапор автоініціалізації топології мережі для системи Network Function
Virtualization для операторів зв’язку
    FAutoInit: boolean;
    // Прапор продовження навчання
    FContinueTeach: boolean;
    // Бажаний вихід нейромережі розмірність OutputNeuronCount
    FDesiredOut: TVectorFloat;
    // Прапор зупинки при досягненні FEpochCount
    FEpoch: boolean;
    // Лічильник етапів (пред'явлення мережі для системи Network Function
Virtualization для операторів зв’язку всіх прикладів з навчальної вибірки)
    FEpochCount: integer;
    // Номер поточної епохи
    FEpochCurrent: integer;
    // Значення помилки, при якій приклад вважається розпізнаним
    FIdentError: double;
    // Значення максимальної помилки на навчальній множині
    FMaxTeachResidual: double;
    // Значення максимальної помилки на тестовій множині
    FMaxTestResidual: double;
    // Значення середньої помилки на навчальній множині
    FMidTeachResidual: double;
    // Значення середньої помилки на тестовій множині
    FMidTestResidual: double;
```

```
    // Помилка на навчальній множині
    FTeachError: double;
    // Коефіцієнт інерційності
    FMomentum: double;
    // Кількість нейронів у шарах
    FNeuronsInLayer: TStrings;
    // Подія після ініціалізації
    FOnAfterInit: TNotifyEvent;
    FOnAfterNeuronCreated: TNotifyEvent;
    // Подія після навчання
    FOnAfterTeach: TNotifyEvent;
    // Подія до ініціалізації
    FOnBeforeInit: TNotifyEvent;
    // Подія до початку навчання
    FOnBeforeTeach: TNotifyEvent;
    // Подія після проходження одного етапу
    FOnEpochPassed: TNotifyEvent;
    // Число прикладів у навчальній безлічі
    FPatternCount: integer;
    // Масив утримуючий псевдовипадкову послідовність
    FRandomOrder: TVectorInt;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTeachCount: integer;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTestCount: integer;
    // Прапор зупинки навчання
    FStopTeach: boolean;
    FTeachStopped: boolean;
    // Коефіцієнт швидкості навчання - величина градієнтного кроку
    FTeachRate: double;
    // Число прикладів у тестовій множині
    FTestSetPatternCount: integer;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatterns: TMatrixFloat;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatternsOut: TMatrixFloat;
    function GetDesiredOut(Index: integer): double;
    function GetLayersBP(Index: integer): TLayerBP;
    function GetTestSetPatterns(InputIndex, PatternIndex: integer): double;
    function GetTestSetPatternsOut(InputIndex, PatternIndex: integer): double;
    procedure NeuronCountError;
    procedure NeuronsInLayerChange(Sender: TObject);
    procedure SetAlpha(Value: double);
    procedure SetDesiredOut(Index: integer; Value: double);
    procedure SetEpochCount(Value: integer);
    procedure SetLayersBP(Index: integer; Value: TLayerBP);
    procedure SetMomentum(Value: double);
    procedure SetTeachRate(Value: double);
    procedure SetTestSetPatternCount(const Value: integer);
    procedure SetTestSetPatterns(InputIndex, PatternIndex: integer; const Value:
double);
    procedure SetTestSetPatternsOut(InputIndex, PatternIndex: integer; const
Value: double);
    // Перетасування набору даних
    procedure Shuffle;
  protected
    function GetLayerCount: integer; override;
    function GetOutput(Index: integer): double; virtual;
    // Активаційна функція
    function Activation(Value: double): double; virtual;
    // Похідна активаційної функції
    function Activation(Value: double): double; virtual;
    // Середня квадратична помилка
    function QuadError: double; virtual;
    // Підстроювання ваг
    procedure AdjustWeights; virtual;
    // Розраховує  локальну помилку - дельту
    procedure CalcLocalError; virtual;
```

```
    // Перевірка мережі для системи Network Function Virtualization для
операторів зв’язку на тестовій множині
    procedure CheckTestSet; virtual;
    procedure DoOnAfterInit; virtual;
    procedure DoOnAfterNeuronCreated(ALayerIndex, ANeuronIndex: integer);
virtual;
    procedure DoOnAfterTeach; virtual;
    procedure DoOnBeforeInit; virtual;
    procedure DoOnBeforeTeach; virtual;
    procedure DoOnEpochPassed; virtual;
    // Ініціалізація ваг мережі для системи Network Function Virtualization для
операторів зв’язку псевдовипадковими значеннями
    procedure InitWeights; virtual;
    // Пред'явлення мережі для системи Network Function Virtualization для
операторів зв’язку вхідних значень приклада
    procedure LoadPatternsInput(APatternIndex :integer); virtual;
    // Пред'явлення мережі для системи Network Function Virtualization для
операторів зв’язку вхідних значень приклада
    procedure LoadPatternsOutput(APatternIndex :integer); virtual;
    // Поширює сигнал у прямому напрямку
    procedure Propagate; virtual;
    // Установка значень за замовчуванням
    procedure SetDefaultProperties; virtual;
    procedure SetPatternCount(const Value: integer); override;
    // Струс мережі для системи Network Function Virtualization для операторів
зв’язку
    procedure ShakeUp; virtual;
    property TeachStopped: boolean read FTeachStopped write FTeachStopped;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); override;
    procedure Compute(AVector: TVectorFloat); virtual;
    procedure DeleteLayer(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure ResetLayers; virtual;
    procedure TeachOffLine; virtual;
    property DesiredOut[Index: integer]: double read GetDesiredOut write
SetDesiredOut;
    property EpochCurrent: integer read FEpochCurrent;
    property IdentError: double read FIdentError write FIdentError;
    property LayersBP[Index: integer]: TLayerBP read GetLayersBP write
SetLayersBP;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Output[Index: integer]: double read GetOutput;
    property StopTeach: boolean read FStopTeach write FStopTeach;
    property TeachError: double read FTeachError;
    property MaxTeachResidual: double read FMaxTeachResidual;
    property MaxTestResidual: double read FMaxTestResidual;
    property MidTeachResidual: double read FMidTeachResidual;
    property MidTestResidual: double read FMidTestResidual;
    property RecognizedTeachCount: integer read FRecognizedTeachCount;
    property RecognizedTestCount: integer read FRecognizedTestCount;
    property TestSetPatternCount: integer read FTestSetPatternCount write
SetTestSetPatternCount;
    property TestSetPatterns[InputIndex: integer; PatternIndex: integer]: double
read GetTestSetPatterns write SetTestSetPatterns;
    property TestSetPatternsOut[InputIndex: integer; PatternIndex: integer]:
double read GetTestSetPatternsOut write SetTestSetPatternsOut;
  published
    property Alpha: double read FAlpha write SetAlpha;
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property ContinueTeach: boolean read FContinueTeach write FContinueTeach;
    property Epoch: boolean read FEpoch write FEpoch;
    property EpochCount: integer read FEpochCount write SetEpochCount;
    property Momentum: double read FMomentum write SetMomentum;
    property NeuronsInLayer: TStrings read FNeuronsInLayer write
FNeuronsInLayer;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
```

```
    property OnAfterNeuronCreated: TNotifyEvent read FOnAfterNeuronCreated write
FOnAfterNeuronCreated;
    property OnAfterTeach: TNotifyEvent read FOnAfterTeach write FOnAfterTeach;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnBeforeTeach: TNotifyEvent read FOnBeforeTeach write
FOnBeforeTeach;
    property OnEpochPassed: TNotifyEvent read FOnEpochPassed write
FOnEpochPassed;
    property PatternCount: integer read FPatternCount write SetPatternCount;
    property TeachRate: double read FTeachRate write SetTeachRate;
  end;

  // Клас мережі для системи Network Function Virtualization для операторів
зв'язку back-propagation TNeuralNetExtended }
  TNeuralNetExtended = class(TNeuralNetBP)
  private
    // Файл даних
    FNeuroDataSource: TNeuroDataSource;
    // Ім'я файлу даних *.txt
    FSourceFileName: TFileName;
    // Ім'я конфігураційного файлу *.nnw
    FFileName: TFileName;
    // Конфігураційний файл
    FNnwFile: TIniFile;
    // Поля
    FFields: TNeuroFields;
    // Кількість доступних полів
    FAvailableFieldsCount: integer;
    FMaxTeachError: boolean;
    FMaxTeachErrorValue: double;
    FMaxTestError: boolean;
    FMaxTestErrorValue: double;
    FMidTeachError: boolean;
    FMidTeachErrorValue: double;
    FMidTestError: boolean;
    FMidTestErrorValue: double;
    FOptions: string;
    FSettingsLoaded: boolean;
    FTestAsValid: boolean;
    FTeachIdent: boolean;
    FTeachIdentCount: integer;
    FTestIdent: boolean;
    FTestIdentCount: integer;
    FUseForTeach: integer;
    FIdentError: double;
    FRealOutputIndex: TVectorInt;
    FRealInputIndex: TVectorInt;
    function GetFields(Index: integer): TNeuroField;
    function GetInputFieldCount: integer;
    function GetOutputFieldCount: integer;
    function GetRealInputIndex(Index: integer): integer;
    function GetRealOutputIndex(Index: integer): integer;
    procedure SetFields(Index: integer; Value: TNeuroField);
    procedure SetFileName(Value: TFilename);
    procedure SetAvailableFieldsCount(Value: integer);
    procedure SetUseForTeach(const Value: integer);
    procedure SetTeachIdentCount(const Value: integer);
    procedure SetRealOutputIndex(Index: integer; const Value: integer);
    procedure SetRealOutputIndexCount(const Value: integer);
    procedure SetRealInputIndex(Index: integer; const Value: integer);
    procedure SetRealInputIndexCount(const Value: integer);
  protected
    function GetOutput(Index: integer): double; override;
    procedure DoOnBeforeTeach; override;
    procedure DoOnEpochPassed; override;
    procedure SetDefaultProperties; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
```

```
    procedure ComputeUnPrepData(AVector: TVectorFloat);
    // Завантажує дані з текстового файлу
    procedure LoadDataFrom;
    // Завантажує настроювання мережі для системи Network Function
Virtualization для операторів зв’язку
    procedure LoadNetwork;
    // Завантажує настроювання мережі для системи Network Function
Virtualization для операторів зв’язку
    procedure LoadPhase1;
    // Завантажує настроювання мережі для системи Network Function
Virtualization для операторів зв’язку
    procedure LoadPhase2;
    // Завантажує настроювання мережі для системи Network Function
Virtualization для операторів зв’язку
    procedure LoadPhase4;
    // Нормалізує набір даних
    procedure NormalizeData;
    // Зберігає настроювання мережі для системи Network Function Virtualization
для операторів зв’язку
    procedure SaveNetwork;
    // Зберігає настроювання мережі для системи Network Function Virtualization
для операторів зв’язку
    procedure SavePhase1;
    // Зберігає настроювання мережі для системи Network Function Virtualization
для операторів зв’язку
    procedure SavePhase2;
    // Зберігає настроювання мережі для системи Network Function Virtualization
для операторів зв’язку
    procedure SavePhase4;
    // Навчання нейронної мережі для системи Network Function Virtualization для
операторів зв’язку
    procedure Train;
    property AvailableFieldsCount: integer read FAvailableFieldsCount write
SetAvailableFieldsCount;
    property Fields[Index: integer]: TNeuroField read GetFields write SetFields;
    property InputFieldCount: integer read GetInputFieldCount;
    property OutputFieldCount: integer read GetOutputFieldCount;
    property SettingsLoaded: boolean read FSettingsLoaded write FSettingsLoaded;
    property RealOutputIndex[Index: integer]: integer read GetRealOutputIndex
write SetRealOutputIndex;
    property RealOutputIndexCount: integer write SetRealOutputIndexCount;
    property RealInputIndex[Index: integer]: integer read GetRealInputIndex
write SetRealInputIndex;
    property RealInputIndexCount: integer write SetRealInputIndexCount;
    property NnwFile: TIniFile read FNnwFile write FNnwFile;
  published
    property FileName: TFileName read FFileName write SetFileName;
    property IdentError: double read FIdentError write FIdentError;
    property MaxTeachError: boolean read FMaxTeachError write FMaxTeachError;
    property MaxTeachErrorValue: double read FMaxTeachErrorValue write
FMaxTeachErrorValue;
    property MaxTestError: boolean read FMaxTestError write FMaxTestError;
    property MaxTestErrorValue: double read FMaxTestErrorValue write
FMaxTestErrorValue;
    property MidTeachError: boolean read FMidTeachError write FMidTeachError;
    property MidTeachErrorValue: double read FMidTeachErrorValue write
FMidTeachErrorValue;
    property MidTestError: boolean read FMidTestError write FMidTestError;
    property MidTestErrorValue: double read FMidTestErrorValue write
FMidTestErrorValue;
    property Options: string read FOptions write FOptions;
    property SourceFileName: TFileName read FSourceFileName write
FSourceFileName;
    property TestAsValid: boolean read FTestAsValid write FTestAsValid;
    property TeachIdent: boolean read FTeachIdent write FTeachIdent;
    property TeachIdentCount: integer read FTeachIdentCount write
SetTeachIdentCount;
    property TestIdent: boolean read FTestIdent write FTestIdent;
    property TestIdentCount: integer read FTestIdentCount write FTestIdentCount;
```

```
    property UseForTeach: integer read FUseForTeach write SetUseForTeach;
  end;


procedure Register;

implementation

{$R *.RES}

{ TNeuron }

constructor TNeuron.Create(ALayer: TLayer);
begin
  inherited Create;
  // покажчик на шар у якому перебуває нейрон
  Layer := ALayer;
end;

destructor TNeuron.Destroy;
begin
  WeightCount := 0;
  FWeights := nil;
  Layer := nil;
  inherited;
end;

procedure TNeuron.ComputeOut(const AInputs: TVectorFloat);
var
  i: integer;
begin
  FOutput := 0;
  // Підраховується зважена сума нейрона
  for i := Low(AInputs) to High(AInputs) do
    FOutput := FOutput + FWeights[i] * AInputs[i];
end;

function TNeuron.GetWeights(Index: integer): double;
begin
  try
    Result := FWeights[Index];
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
end;

procedure TNeuron.InitWeights;
var
  i: integer;
begin
  // Ініціалізація ваг нейрона
  for i := Low(FWeights) to High(FWeights) do
    FWeights[i] := Random
end;

procedure TNeuron.SetWeightCount(const Value: integer);
begin
  SetLength(FWeights, Value);
end;

procedure TNeuron.SetWeights(Index: integer; Value: double);
begin
  try
    FWeights[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
```

```
end;

{ Кінець опису TNeuron }

{ TNeuronHopf }

procedure TNeuronHopf.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // гранична функція
  if FOutput >= 0 then
    FOutput := 1
  else
    FOutput := -1
end;

{ Кінець опису TNeuronHopf}

{ TNeuronBP }

destructor TNeuronBP.Destroy;
begin
  FOnActivation := nil;
  FOnActivation := nil;
  PrevUpdateCount := 0;
  FPrevUpdate := nil;
  inherited;
end;

function TNeuronBP.GetLearningRate(Index: integer): double;
begin
  Result := FLearningRate[Index];
end;

function TNeuronBP.GetPrevDerivative(Index: integer): double;
begin
  Result := FPrevDerivative[Index];
end;

function TNeuronBP.GetPrevUpdateCount: integer;
begin
  Result := High(FPrevUpdate) + 1;
end;

function TNeuronBP.GetPrevUpdate(Index: integer): double;
begin
  Result := FPrevUpdate[Index];
end;

procedure TNeuronBP.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // Задає зсув нейрона
  FOutput := FOutput + Weights[High(AInputs) + 1];
  FOutput := OnActivation(FOutput);
end;

procedure TNeuronBP.SetDelta(Value: double);
begin
  FDelta := Value;
end;

procedure TNeuronBP.SetLearningRate(Index: integer; const Value: double);
begin
  FLearningRate[Index] := Value;
end;

procedure TNeuronBP.SetLearningRateCount(const Value: integer);
begin
```

```
  SetLength(FLearningRate, Value)
end;

procedure TNeuronBP.SetPrevUpdate(Index: integer; Value: double);
begin
  FPrevUpdate[Index] := Value;
end;

procedure TNeuronBP.SetPrevUpdateCount(const Value: integer);
begin
  SetLength(FPrevUpdate, Value)
end;

procedure TNeuronBP.SetPrevDerivative(Index: integer; const Value: double);
begin
  FPrevDerivative[Index] := Value;
end;

procedure TNeuronBP.SetPrevDerivativeCount(const Value: integer);
begin
  SetLength(FPrevDerivative, Value)
end;

{ Кінець опису TNeuronBP }

{ TLayer }

procedure TLayer.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayer).FNumber;
  NeuronCount := (Source as TLayer).NeuronCount;
  // Створюються нейрони
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

constructor TLayer.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  inherited Create;
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

destructor TLayer.Destroy;
var
  i: integer;
begin
  for i := 0 to NeuronCount - 1 do
    FNeurons[i].Free;
  NeuronCount := 0;
  FNeurons := nil;
  inherited;
end;

function TLayer.GetNeuronCount: integer;
begin
  Result := High(FNeurons) + 1;
end;

function TLayer.GetNeurons(Index: integer): TNeuron;
begin
  Result := FNeurons[Index];
end;
```

```
procedure TLayer.SetNeuronCount(Value: integer);
begin
  if Value <> High(FNeurons) + 1 then
    SetLength(FNeurons, Value);
end;

procedure TLayer.SetNeurons(Index: integer; Value: TNeuron);
begin
  try
    FNeurons[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
  end;
end;

{ TLayerHopf }

constructor TLayerHopf.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronHopf.Create(Self);
end;

{ TLayerBP }

procedure TLayerBP.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayerBP).FNumber;
  NeuronCount := (Source as TLayerBP).NeuronCount;
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

constructor TLayerBP.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

destructor TLayerBP.Destroy;
begin
  inherited;
end;

function TLayerBP.GetNeuronsBP(Index: integer): TNeuronBP;
begin
  Result := FNeurons[Index] as TNeuronBP;
end;

procedure TLayerBP.SetNeuronsBP(Index: integer; Value: TNeuronBP);
begin
  FNeurons[Index] := Value as TNeuronBP;
end;

{ TNeuralNet }

destructor TNeuralNet.Destroy;
```

```
begin
  Clear;
  SetLength(FPatternsInput, 0, 0);
  FPatternsInput := nil;
  SetLength(FPatternsOutput, 0 ,0);
  FPatternsOutput := nil;
  FLayers := nil;
  inherited;
end;

procedure TNeuralNet.Clear;
var
  i, xCount: integer;
begin
  xCount := LayerCount;
  if  xCount > 0 then
  begin
    for i := 0 to xCount - 1 do
      FLayers[i].Free;
    LayerCount := 0;
  end;
end;

function TNeuralNet.GetInputNeuronCount: integer;
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNet.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNet.GetLayers(Index: integer): TLayer;
begin
  Result := FLayers[Index];
end;

function TNeuralNet.GetOutputNeuronCount: integer;
begin
  Result := Layers[LayerCount - 1].NeuronCount;
end;

function TNeuralNet.GetPatternsInput(PatternIndex: integer; InputIndex:
integer): double;
begin
  Result := FPatternsInput[PatternIndex, InputIndex];
end;

procedure TNeuralNet.AddPattern(const AInputs: TVectorFloat; const AOutputs:
TVectorFloat);
var
  i: integer;
begin
  if InputNeuronCount <> High(AInputs) + 1 then
    raise EInOutDimensionError.Create(SInVectorCount);
  if OutputNeuronCount <> High(AOutputs) + 1 then
    raise EInOutDimensionError.Create(SOutVectorCount);
  PatternCount := PatternCount + 1;
  ResizeInputDim;
  ResizeOutputDim;
  for i := Low(AInputs) to High(AInputs) do
    PatternsInput[PatternCount - 1, i] := AInputs[i];
  for i := Low(AOutputs) to High(AOutputs) do
    PatternsOutput[PatternCount - 1, i] := AOutputs[i];
end;

procedure TNeuralNet.DeletePattern(Index: integer);
var
```

```
  i, j: integer;
begin
  try
    // видаляє вхідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to InputNeuronCount - 1 do
        FPatternsInput[i, j] := FPatternsInput[i + 1, j];
    // видаляє вихідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to OutputNeuronCount - 1 do
        FPatternsOutput[i, j] := FPatternsOutput[i + 1, j];
    Dec(FPatternCount);
    ResizeInputDim;
    ResizeOutputDim;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNet.ResetPatterns;
begin
  FPatternCount := DefaultPatternCount;
  ResizeInputDim;
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
  ResizeInputDim;
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternsOutput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsOutput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.SetPatternsInput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsInput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.Init(const ANeuronsInLayer: TVectorInt);
var
  i,j: integer;
begin
  LayerCount := High(ANeuronsInLayer) + 1;
  // FLayers[0] нульовий шар і виконує роль розподільного,
  // використовується тільки поле Output
  FLayers[0] := TLayer.Create(0, ANeuronsInLayer[0]);
  // для нульового шару не потрібні вагові коефіцієнти
  for i := 1 to LayerCount - 1 do
  begin
    FLayers[i] := TLayer.Create(i, ANeuronsInLayer[i]);
    for j := 0 to ANeuronsInLayer[i] - 1 do
    with FLayers[i].FNeurons[j] do
      // задає кількість елементів у векторі ваг нейрона j в
      //  шарі i рівним кількості виходів попереднього шару
      WeightCount := FLayers[ i-1].NeuronCount;
  end;
end;
```

```
procedure TNeuralNet.ResizeInputDim;
begin
  SetLength(FPatternsInput, FPatternCount, InputNeuronCount)
end;

procedure TNeuralNet.ResizeOutputDim;
begin
  SetLength(FPatternsOutput, FPatternCount, OutputNeuronCount)
end;

procedure TNeuralNet.SetLayerCount(Value: integer);
begin
  SetLength(FLayers, Value);
end;

procedure TNeuralNet.SetLayers(Index: integer; Value: TLayer);
begin
  try
    FLayers[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;

{  TNeuralNetHopf }

constructor TNeuralNetHopf.Create(AOwner: TComponent);
begin
  inherited;
  PatternCount := DefaultPatternCount;
  InputNeuronCount := DefaultNeuronCount;
  MaxIterCount := DefaultMaxIterCount;
  AutoInit := False;
end;

destructor TNeuralNetHopf.Destroy;
begin
  FOnAfterInit := nil;
  FOnBeforeInit := nil;
  FOnPatternRecognized := nil;
  SetLength(FPatterns, 0, 0);
  FPatterns := nil;
  inherited;
end;

function TNeuralNetHopf.GetInput(Index: integer): double;
begin
  Result := Layers[1].Neurons[Index].Output;
end;

function TNeuralNetHopf.GetInputNeuronCount: integer;
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNetHopf.GetLayerCount: integer;
begin
  Result := DefaultHopfLayerCount;
end;

function TNeuralNetHopf.GetPatterns(InputIndex: integer; PatternIndex: integer):
integer;
begin
  Result := FPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetHopf.Stabled: boolean;
var
```

```
  i: integer;
begin
  // Порівнює вихідні значення попередньої
  //  ітерації зі значеннями поточної
  Result := True;
  for i := 0 to InputNeuronCount - 1  do
    if FLayers[1].FNeurons[i].FOutput <> FLayers[0].FNeurons[i].FOutput then
    begin
      Result := False;
      Exit
    end;
end;

procedure TNeuralNetHopf.AddPattern(const ANewPattern: TVectorInt);
var
  i: integer;
begin
  if InputNeuronCount <> High(ANewPattern)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  PatternCount := PatternCount + 1;
  ResizePatternsDim;
  for i := 0 to FInputNeuronCount - 1 do
    FPatterns[FPatternCount - 1, i] := ANewPattern[i];
  if AutoInit then
    InitWeights;
end;

procedure TNeuralNetHopf.Calc;
var
  i: integer;
  xCurrentIter: integer;
  xArray: TVectorFloat;
begin
  SetLength(xArray, InputNeuronCount);
  // Цикл працює поки не стабілізуються виходи
  xCurrentIter := 0;
  repeat
    for i := 0 to InputNeuronCount - 1 do
    begin
      // Запам'ятовує попередній крок ітерації, для
      //  цього використовується нульовий шар
      Layers[SensorLayer].Neurons[i].Output := Layers[1].Neurons[i].Output;
      xArray[i] := Layers[1].Neurons[i].Output;
    end;
    for i := 0 to InputNeuronCount - 1 do
      with Layers[1].Neurons[i] do
        // Розраховується новий стан нейронів і аксонів
        ComputeOut(xArray);
    Inc(xCurrentIter);
  until Stabled or (MaxIterCount = xCurrentIter);
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetHopf.DeletePattern(Index: integer);
var
  i, j: integer;
begin
  try
    for i := Index to FPatternCount - 2 do
      for j := 0 to FInputNeuronCount - 1 do
        FPatterns[i, j] := FPatterns[i + 1, j];
    Dec(FPatternCount);
    ResizePatternsDim;
    if AutoInit then
      InitWeights;
  except
```

```
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNetHopf.Init;
var
  i, j: integer;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
  LayerCount := DefaultHopfLayerCount;
  for i := 0 to LayerCount - 1 do
    FLayers[i] := TLayerHopf.Create(i, FInputNeuronCount);
  // Для нульового шару не потрібні вагові коефіцієнти
  for j := 0 to FInputNeuronCount - 1 do
    with FLayers[1].FNeurons[j] do
      // задає кількість елементів у векторі
      WeightCount := FInputNeuronCount;
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TNeuralNetHopf.InitWeights;
var
  i, j, k : integer;
begin
  // Ініціалізує вагову матрицю
  for i := 0 to InputNeuronCount - 1 do
    for j := 0 to InputNeuronCount - 1 do
    with Layers[1].Neurons[i] do
    begin
      Weights[j] := 0;
      if i <> j then
        for k := 0 to PatternCount - 1 do
          Weights[j] := Weights[j]+ Patterns[k, i] * Patterns[k, j]
    end;
end;

procedure TNeuralNetHopf.ResetPatterns;
begin
  PatternCount := DefaultPatternCount;
  if AutoInit then
    InitWeights;
end;

procedure TNeuralNetHopf.ResizePatternsDim;
begin
  SetLength(FPatterns, FPatternCount, FInputNeuronCount);
end;

procedure TNeuralNetHopf.SetInput(Index: integer; Value: double);
begin
  try
    Layers[1].Neurons[Index].Output := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNetHopf.SetInputNeuronCount(Value: integer);
begin
  if Value > DefaultNeuronCount then
    FInputNeuronCount := Value
  else
    FInputNeuronCount := DefaultNeuronCount;
  ResizePatternsDim;
  Init;
```

```
end;

procedure TNeuralNetHopf.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
end;

procedure TNeuralNetHopf.SetPatterns(InputIndex: integer; PatternIndex: integer;
Value: integer);
begin
  FPatterns[InputIndex, PatternIndex] := Value;
end;

{ TNeuralNetBP }

constructor TNeuralNetBP.Create(AOwner: TComponent);
var
  i: integer;
begin
  inherited;
  FNeuronsInLayer := TStringList.Create;
  for i := 0 to DefaultLayerCount do
    AddLayer(DefaultNeuronCount);
  TStringList(FNeuronsInLayer).OnChange := NeuronsInLayerChange;
  AutoInit := True;
  StopTeach := False;
  TeachStopped := False;
  NeuronsInLayerChange(Self);
  SetDefaultProperties;
end;

destructor TNeuralNetBP.Destroy;
begin
  FNeuronsInLayer.Free;
  SetLength(FRandomOrder, 0);
  FRandomOrder := nil;
  SetLength(FDesiredOut, 0);
  FDesiredOut := nil;
  SetLength(FTestSetPatterns, 0, 0);
  FTestSetPatterns := nil;
  SetLength(FTestSetPatternsOut, 0, 0);
  FTestSetPatternsOut := nil;
  FOnAfterInit := nil;
  FOnAfterTeach := nil;
  FOnBeforeInit := nil;
  FOnBeforeTeach := nil;
  FOnEpochPassed := nil;
  inherited;
end;

function TNeuralNetBP.GetLayersBP(Index: integer): TLayerBP;
begin
  Result := FLayers[Index] as TLayerBP;
end;

function TNeuralNetBP.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNetBP.GetDesiredOut(Index: integer): double;
begin
  Result := FDesiredOut[Index];
end;

function TNeuralNetBP.GetOutput(Index: integer): double;
```

```
begin
  try
    Result := LayersBP[LayerCount - 1].NeuronsBP[Index].Output;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
  end;
end;

function TNeuralNet.GetPatternsOutput(PatternIndex: integer; OutputIndex:
integer): double;
begin
  Result := FPatternsOutput[PatternIndex, OutputIndex];
end;

function TNeuralNetBP.QuadError: double;
var
  i: integer;
begin
  // розраховує середньоквадратичну помилку
  Result := 0;
  for i := 0 to OutputNeuronCount - 1 do
    Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
DesiredOut[i]);
  Result := Result/2;
end;

function TNeuralNetBP.Activation(Value: double): double;
begin
  // Активаційна функція - сигмоїд
  Result := 1/( 1 + exp(-FAlpha * Value) )
end;

function TNeuralNetBP.Activation(Value: double): double;
begin
  // Похідна сигмоїди
  Result := FAlpha * Value * (1 - Value)
end;

function TNeuralNetBP.GetTestSetPatterns(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetBP.GetTestSetPatternsOut(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatternsOut[InputIndex, PatternIndex];
end;

procedure TNeuralNetBP.AddLayer(ANeurons: integer);
begin
  if ANeurons < DefaultNeuronCount then
    NeuronCountError
  else
    NeuronsInLayer.Add(IntToStr(ANeurons));
end;

procedure TNeuralNetBP.AdjustWeights;
var
  i, j, k: integer;
  xCurrentUpdate: double;
begin
  // Підстроювання ваг починаючи з першого шару
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount  do
```

```
      with LayersBP[i].NeuronsBP[j] do
      begin
        // коректує вагу з'єднуючого j-нейрона шару i
        //  з k-нейроном шару i-1:  добутком дельта j-нейрона
        //  на вихід k-нейрона шару i-1
        if k = LayersBP[ i-1].NeuronCount then
           // якщо це нейрон, що задає зсув
           xCurrentUpdate := -TeachRate * Delta + Momentum * PrevUpdate[k]
        else
           xCurrentUpdate := -TeachRate * Delta *
           LayersBP[ i-1].NeuronsBP[k].Output + Momentum * PrevUpdate[k];
        Weights[k]:= Weights[k] + xCurrentUpdate;
        PrevUpdate[k] := xCurrentUpdate;
      end;
    end
end;

procedure TNeuralNetBP.CalcLocalError;
var
  i, j, k: integer;
begin
  // Дельта-правило з останнього шару до першого
  for i := LayerCount - 1 downto 1 do
    // для останнього шару
    if i = LayerCount - 1 then
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Delta := (LayersBP[i].NeuronsBP[j].Output-
DesiredOut[j])
        * Activation(LayersBP[i].NeuronsBP[j].Output)
    else
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        with LayersBP[i].NeuronsBP[j] do
        begin
          Delta := 0;
          // Підсумує добуток локальної помилки k-нейрона шару i+1
          //  на вагу з'єднуючий k-нейрон шару i+1 з j-нейроном шару i
          for k := 0 to LayersBP[i+1].NeuronCount - 1 do
            Delta := Delta + LayersBP[i+1].NeuronsBP[k].Delta *
                             LayersBP[i+1].NeuronsBP[k].Weights[j];
          Delta := Delta * Activation(Output)
        end;
end;

procedure TNeuralNetBP.CheckTestSet;
var
  i, j: integer;
  xArray: TVectorFloat;
  xFirstTestSample: boolean;
  xQuadError: double;
  // функція розраховує середньоквадратичну помилку
  function QuadError(APatternCount: integer): double;
  var
    i: integer;
  begin
    Result := 0;
    for i := 0 to OutputNeuronCount - 1 do
      Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
TestSetPatternsOut[APatternCount, i]);
    Result := Result/2;
  end;
begin
  SetLength(xArray, InputNeuronCount);
  xFirstTestSample := True;
  FRecognizedTestCount := 0;
  FMidTestResidual := 0;
  FMaxTestResidual := 0;
  for i := 0 to TestSetPatternCount - 1 do
  begin
    for j := 0 to InputNeuronCount - 1 do
```

```
      xArray[j] := TestSetPatterns[i, j];
    Compute(xArray);
    xQuadError := QuadError(i);
    // перевірка - чи розпізнаний приклад з тестової множини
    if xQuadError < IdentError then
      Inc(FRecognizedTestCount);
    FMidTestResidual := FMidTestResidual + xQuadError;
    // максимальна помилка на тестовій множині
    if xFirstTestSample then
    begin
      FMaxTestResidual := xQuadError;
      xFirstTestSample := False;
    end
    else
      if FMaxTestResidual < xQuadError then
        FMaxTestResidual := xQuadError;
  end;
  // середня помилка на тестовій множині
  FMidTestResidual := FMidTestResidual/TestSetPatternCount;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetBP.Compute(AVector: TVectorFloat);
var
  i: integer;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  AVector[i];
  Propagate;
end;

procedure TNeuralNetBP.DoOnAfterInit;
begin
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TNeuralNetBP.DoOnAfterNeuronCreated(ALayerIndex, ANeuronIndex:
integer);
var
  i: integer;
begin
  with LayersBP[ALayerIndex].NeuronsBP[ANeuronIndex] do
    for i := 0 to PrevUpdateCount - 1 do
      PrevUpdate[i] := 0;
  if Assigned(FOnAfterNeuronCreated) then
    FOnAfterNeuronCreated(Self);
end;

procedure TNeuralNetBP.DoOnAfterTeach;
begin
  if Assigned(FOnAfterTeach) then
    FOnAfterTeach(Self);
end;

procedure TNeuralNetBP.DoOnBeforeInit;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
end;

procedure TNeuralNetBP.DoOnBeforeTeach;
begin
  if Assigned(FOnBeforeTeach) then
    FOnBeforeTeach(Self);
end;
```

```
procedure TNeuralNetBP.DoOnEpochPassed;
begin
  if Assigned(FOnEpochPassed) then
    FOnEpochPassed(Self);
end;

procedure TNeuralNetBP.DeleteLayer(Index: integer);
var
  i: integer;
begin
  try
    NeuronsInLayer.Delete(Index);
    for i := Index to LayerCount - 2 do
      LayersBP[i].Assign(LayersBP[i + 1]);
    FLayers[LayerCount - 1].Free;
    LayerCount := LayerCount - 1;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;

procedure TNeuralNetBP.Init;
var
  i, j: integer;
begin
  DoOnBeforeInit;
  if NeuronsInLayer.Count > 0 then
  begin
  LayerCount := NeuronsInLayer.Count;
    // FLayers[0] нульовий шар, використовується тільки поле Output
    FLayers[0] := TLayerBP.Create(0, StrToInt(NeuronsInLayer.Strings[0]));
    // для нульового шару не потрібні вагові коефіцієнти
    for i := 1 to LayerCount - 1 do
    begin
      FLayers[i] := TLayerBP.Create(i, StrToInt(NeuronsInLayer.Strings[i]));
      for j := 0 to StrToInt(NeuronsInLayer.Strings[i]) - 1 do
      with LayersBP[i].NeuronsBP[j] do
      begin
        // задає кількість елементів у векторі ваг + зсув
        WeightCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        // задає кількість  у векторі утримуючих попередню
        //  корекцію елементів  + зсув
        PrevUpdateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        PrevDerivativeCount := LayersBP[ i-1].NeuronCount + BiasNeuron; // для
швидких алгоритмів
        LearningRateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;   // для
швидких алгоритмів
        OnActivation := Activation;
        OnActivation := Activation;
        Randomize;
        DoOnAfterNeuronCreated(i, j);
      end
    end;
    // установлює розмірність масиву виходів
    //  число нейронів в останньому шарі = числу виходів
    SetLength(FDesiredOut, OutputNeuronCount);
  end;
  DoOnAfterInit;
end;

procedure TNeuralNetBP.InitWeights;
var
  i,j: integer;
begin
  Randomize;
  // Ініціалізація ваг
  for i := 1 to LayerCount - 1 do
```

```
    for j := 0 to LayersBP[i].NeuronCount - 1 do
       LayersBP[i].NeuronsBP[j].InitWeights;
end;

procedure TNeuralNetBP.LoadPatternsInput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to InputNeuronCount - 1 do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  PatternsInput[APatternIndex,
i];
end;

procedure TNeuralNetBP.LoadPatternsOutput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to OutputNeuronCount - 1 do
    DesiredOut[i] := PatternsOutput[APatternIndex, i];
end;

procedure TNeuralNetBP.NeuronsInLayerChange(Sender: TObject);
begin
  if AutoInit then
    Init;
end;

procedure TNeuralNetBP.NeuronCountError;
begin
  raise ENeuronCountError.Create(SNeuronCount)
end;

procedure TNeuralNetBP.Propagate;
var
  i, j, xIndex: integer;
  xArray: TVectorFloat;
begin
   // Поширення сигналу в прямому напрямку з першого шару
  for i := 1 to LayerCount - 1 do
  begin
    // формування масиву входів з виходів попереднього шару
    SetLength(xArray, LayersBP[ i-1].NeuronCount);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := LayersBP[ i-1].NeuronsBP[xIndex].Output;
    // обчислення виходу нейрона
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      with LayersBP[i].NeuronsBP[j] do
        ComputeOut(xArray);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := 0;
  end;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetBP.ResetLayers;
begin
  Clear;
  FNeuronsInLayer.Clear;
end;

procedure TNeuralNetBP.SetDesiredOut(Index: integer; Value: double);
begin
  FDesiredOut[Index] := Value;
end;

procedure TNeuralNetBP.SetLayersBP(Index: integer; Value: TLayerBP);
begin
  FLayers[Index] := Value as TLayerBP;
```

```
end;

procedure TNeuralNetBP.SetAlpha(Value: double);
begin
  if (Value > 10) or (Value < 0.01) then
    FAlpha := DefaultAlpha
  else
    FAlpha := Value;
end;

procedure TNeuralNetBP.SetTeachRate(Value: double);
begin
  if (Value > 1) or (Value <= 0) then
    FTeachRate := DefaultTeachRate
  else
    FTeachRate := Value;
end;

procedure TNeuralNetBP.SetTestSetPatterns(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatterns[InputIndex, PatternIndex] := Value;
end;

procedure TNeuralNetBP.SetTestSetPatternsOut(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatternsOut[InputIndex, PatternIndex] := Value;
end;

procedure TNeuralNetBP.SetTestSetPatternCount(const Value: integer);
begin
  FTestSetPatternCount := Value;
  SetLength(FTestSetPatterns, FTestSetPatternCount, InputNeuronCount);
  SetLength(FTestSetPatternsOut, FTestSetPatternCount, OutputNeuronCount);
end;

procedure TNeuralNetBP.SetMomentum(Value: double);
begin
  if (Value > 1) or (Value < 0) then
    FMomentum := DefaultMomentum
  else
    FMomentum := Value;
end;

procedure TNeuralNetBP.SetEpochCount(Value: integer);
begin
  if Value < 1 then
    FEpochCount := 1
  else
    FEpochCount := Value;
end;

procedure TNeuralNetBP.ShakeUp;
var
  i, j, k: integer;
begin
  Randomize;
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      for k := 0 to LayersBP[ i-1].NeuronCount  do
        with LayersBP[i].NeuronsBP[j] do
          Weights[k]:= Weights[k] + Random*0.1-0.05;
end;

procedure TNeuralNetBP.Shuffle;
var
  i, j, xNewInd, xLast: integer;
  xIsUnique : boolean;
```

```
begin
  xNewInd := 0;
  FRandomOrder[0] := Round(Random(FPatternCount));
  xLast := 0;
  for i := 1 to PatternCount - 1 do
  begin
    xIsUnique := False;
    while not xIsUnique do
    begin
      xNewInd := Round((Random(FPatternCount)));
      xIsUnique := True;
      for j := 0 to xLast do
        if xNewInd = FRandomOrder[j] then
          xIsUnique := False;
    end;
    FRandomOrder[i] := xNewInd;
    xLast := xLast +1;
  end;
end;

procedure TNeuralNetBP.TeachOffLine;
var
  j: integer;
  xQuadError: double;
  xNewEpoch: boolean;
begin
  DoOnBeforeTeach;
  if not ContinueTeach then
  begin
    // ваги ініціалізуються, якщо мережа навчається з "нуля"
    InitWeights;
    FEpochCurrent := 1;
  end;
  Randomize;
  SetLength(FRandomOrder, FPatternCount);
  TeachStopped := False;
  while (FEpochCurrent <= EpochCount) do
  begin
    FTeachError := 0;
    FMaxTeachResidual := 0;
    FRecognizedTeachCount := 0;
    xNewEpoch := True;
    Shuffle;
    for j := 0 to PatternCount - 1 do
    begin
      LoadPatternsInput(FRandomOrder[j]);
      LoadPatternsOutput(FRandomOrder[j]);
      Propagate;
      xQuadError := QuadError;
      // перевірка - чи розпізнаний приклад з навчальної множини
      if xQuadError < IdentError then
        Inc(FRecognizedTeachCount);
      FTeachError := FTeachError + xQuadError;
      // максимальна помилка на навчальній множині
      if xNewEpoch then
      begin
        FMaxTeachResidual := xQuadError;
        xNewEpoch := False;
      end
      else
        if MaxTeachResidual < xQuadError then
          FMaxTeachResidual := xQuadError;
      CalcLocalError;
      AdjustWeights;
    end;
    // середня помилка на навчальній множині
    FMidTeachResidual := TeachError/PatternCount;
    // перевірка мережі для системи Network Function Virtualization для
операторів зв’язку на узагальнення
```

```
    if TestSetPatternCount > 0 then
      CheckTestSet;
    DoOnEpochPassed;
    if StopTeach then
    begin
      TeachStopped := True;
      Exit;
    end;
    Inc(FEpochCurrent);
  end;
  DoOnAfterTeach;
end;

procedure TNeuralNetBP.SetPatternCount(const Value: integer);
begin
  FPatternCount := Value;
  inherited;
end;

procedure TNeuralNetBP.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Alpha := DefaultAlpha;
  ContinueTeach := False;
  Epoch := True;
  EpochCount := DefaultEpochCount;
  Momentum := DefaultMomentum;
  TeachRate := DefaultTeachRate;
  ResizeInputDim;
  ResizeOutputDim;
end;

{ TNeuralNetExtended }

constructor TNeuralNetExtended.Create(AOwner: TComponent);
begin
  inherited;
  SetDefaultProperties;
end;

destructor TNeuralNetExtended.Destroy;
var
  i: integer;
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  FNeuroDataSource.Free;
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i].Free;
  inherited;
end;

function TNeuralNetExtended.GetFields(Index: integer): TNeuroField;
begin
  Result := FFields[Index];
end;

function TNeuralNetExtended.GetInputFieldCount: integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdInput then
      Inc(Result);
end;

function TNeuralNetExtended.GetOutputFieldCount: integer;
var
```

```
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdOutput then
      Inc(Result);
end;

function TNeuralNetExtended.GetOutput(Index: integer): double;
var
  xTmp: double;
begin
  with Fields[RealOutputIndex[Index]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := -ln(1/LayersBP[LayerCount - 1].NeuronsBP[Index].Output -
1);
                 LayersBP[LayerCount - 1].NeuronsBP[Index].Output := xTmp *
Dispersion + ValueMid;
               end;
      nrmLinear:  Result := (LayersBP[LayerCount - 1].NeuronsBP[Index].Output +
1)*(ValueMax - ValueMin)/2 + ValueMin;
      nrmLinearOut: Result := LayersBP[LayerCount -
1].NeuronsBP[Index].Output*(ValueMax - ValueMin) + ValueMin;
      nrmSigmoid:  Result := - Ln(1/LayersBP[LayerCount -
1].NeuronsBP[Index].Output - 1)/Alpha;
    end;
end;

function TNeuralNetExtended.GetRealInputIndex(Index: integer): integer;
begin
  Result := FRealInputIndex[Index];
end;

function TNeuralNetExtended.GetRealOutputIndex(Index: integer): integer;
begin
  Result := FRealOutputIndex[Index];
end;

procedure TNeuralNetExtended.ComputeUnPrepData(AVector: TVectorFloat);
var
  i: integer;
  xTmp: double;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
  with FFields[RealInputIndex[i]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := (LayersBP[SensorLayer].NeuronsBP[i].Output -
ValueMid)/Dispersion;
                 LayersBP[SensorLayer].NeuronsBP[i].Output := 1/(1 + exp(-
xTmp));
               end;
      nrmLinear: LayersBP[SensorLayer].NeuronsBP[i].Output := 2*(AVector[i] -
ValueMin)/(ValueMax - ValueMin) - 1;
      nrmLinearOut: LayersBP[SensorLayer].NeuronsBP[i].Output := (AVector[i] -
ValueMin)/(ValueMax - ValueMin);
      nrmSigmoid: LayersBP[SensorLayer].NeuronsBP[i].Output := 1/( 1 + exp(-
Alpha * AVector[i]));
    end;
  Propagate;
end;

procedure TNeuralNetExtended.DoOnBeforeTeach;
begin
  if InputNeuronCount <> InputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SInFieldCount);
```

```
  if OutputNeuronCount <> OutputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SOutFieldCount);
  if InputNeuronCount < 0 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  if OutputNeuronCount < 0 then
    raise EInOutDimensionError.Create(SOutNeuronCount);
  if (not FMaxTeachError) and (not FMaxTestError) and
     (not FMidTeachError) and (not FMidTestError) and (not FEpoch) then
    raise EBPStopCondition.Create(SBPStopCondition);
  inherited DoOnBeforeTeach;
end;

procedure TNeuralNetExtended.DoOnEpochPassed;
begin
  if MaxTeachError and (MaxTeachResidual < MaxTeachErrorValue) then
    StopTeach := True;
  if MidTeachError and (MidTeachResidual < MidTeachErrorValue) then
    StopTeach := True;
  if MaxTestError and (MaxTestResidual < MaxTestErrorValue) then
    StopTeach := True;
  if MidTestError and (MidTestResidual < MidTestErrorValue) then
    StopTeach := True;
  if TeachIdent and (Round((FRecognizedTeachCount * 100)/PatternCount) <
TeachIdentCount) then
    StopTeach := True;
  if TestIdent and (Round((FRecognizedTestCount * 100)/TestSetPatternCount) <
TestIdentCount) then
    StopTeach := True;
  inherited DoOnEpochPassed;
end;

procedure TNeuralNetExtended.LoadDataFrom;
var
  xTempStream: TFileStream;
  i, j: integer;
  xFieldCount: integer;
  xArray: TVectorFloat;
  xPatternsList: TStringList;
begin
  // створюється потік
  xTempStream := TFileStream.Create(FSourceFileName, fmOpenRead);
  // створюється список
  xPatternsList := TStringList.Create;
  xPatternsList.LoadFromStream(xTempStream);
  try
    if SettingsLoaded then
    begin
      xFieldCount := FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
      if AvailableFieldsCount <> xFieldCount then
       if MessageDlg('Кількість полів у файлі даних не відповідає значенню
AvailableFieldsCount'+ #13 + 'Установити нове значення AvailableFieldsCount = '+
IntToStr(xFieldCount),
                    mtConfirmation, [mbYes, mbNo], 0) = mrYes then
         AvailableFieldsCount := xFieldCount;
    end
    else
      AvailableFieldsCount :=
FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
    FNeuroDataSource.ExtractHeaders(FFields, xPatternsList.Strings[0]);
    // установлюється розмірність часового масиву
    SetLength(xArray, FAvailableFieldsCount);
    // установлюється розмірність масиву даних
    if FUseForTeach = 100 then
      PatternCount := xPatternsList.Count - 1
    else
    begin
      PatternCount := Round((xPatternsList.Count - 1) * FUseForTeach / 100);
      TestSetPatternCount := xPatternsList.Count - PatternCount - 1;
    end;
```

```
    for i := 0 to FAvailableFieldsCount - 1 do
      FFields[i].DataInCount := xPatternsList.Count - 1;
    for j := 0 to xPatternsList.Count - 2 do
    begin
      FNeuroDataSource.ExtractValues(xArray, xPatternsList.Strings[j + 1]);
      for i := 0 to FAvailableFieldsCount - 1 do
        FFields[i].DataIn[j] := xArray[i]
    end;
  finally
    xTempStream.Free;
    xPatternsList.Free;
    SetLength(xArray, 0);
    xArray := nil;
  end;
end;

procedure TNeuralNetExtended.LoadPhase1;
begin
  FSourceFileName := FNnwFile.ReadString('Phase1', 'LearnSampleFileName', '');
  FNeuroDataSource.Name := FSourceFileName;
end;

procedure TNeuralNetExtended.LoadPhase2;
var
  i: integer;
begin
  AvailableFieldsCount := FNnwFile.ReadInteger('Phase2', 'AvailableFieldsCount',
1);
  for i := 0 to AvailableFieldsCount - 1 do
  with FFields[i] do
  begin
    Name := FNnwFile.ReadString('Phase2', 'FieldName_'+IntToStr(i), '');
    Kind := FNnwFile.ReadInteger('Phase2', 'FieldType_'+IntToStr(i), 0);
    NormType := FNnwFile.ReadInteger('Phase2', 'NormType_'+IntToStr(i), 0);
    ValueMax := FNnwFile.ReadFloat('Phase2', 'Max_'+IntToStr(i), 0);
    ValueMin := FNnwFile.ReadFloat('Phase2', 'Min_'+IntToStr(i), 0);
    ValueMid := FNnwFile.ReadFloat('Phase2', 'Mid_'+IntToStr(i), 0);
    Dispersion := FNnwFile.ReadFloat('Phase2', 'Disp_'+IntToStr(i), 0);
    Alpha := FNnwFile.ReadFloat('Phase2', 'Alpha_'+IntToStr(i), 0);
    Ind := FNnwFile.ReadBool('Phase2', 'Ind_'+IntToStr(i), False);
  end;
  SettingsLoaded := True;
end;

procedure TNeuralNetExtended.LoadPhase4;
begin
  UseForTeach := FNnwFile.ReadInteger('Phase4', 'UseForTeach',
DefaultUseForTeach);
  IdentError:= FNnwFile.ReadFloat('Phase4', 'IdentErr', DefaultErrorValue);
  TestAsValid := FNnwFile.ReadBool('Phase4', 'TestAsValid', False);
  Epoch:= FNnwFile.ReadBool('Phase4', 'Epoch', False);
  EpochCount:= FNnwFile.ReadInteger('Phase4', 'Epoch', DefaultEpochCount);
  MaxTeachError:= FNnwFile.ReadBool('Phase4', 'MaxTeachErr', False);
  MaxTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTeachErr',
DefaultErrorValue);
  MidTeachError:= FNnwFile.ReadBool('Phase4', 'MidTeachErr', False);
  MidTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTeachErr',
DefaultErrorValue);
  TeachIdent:= FNnwFile.ReadBool('Phase4', 'TeachIdent', False);
  TeachIdentCount:= FNnwFile.ReadInteger('Phase4', 'TeachIdent',
DefaultTeachIdentCount);
  MaxTestError:= FNnwFile.ReadBool('Phase4', 'MaxTestErr', False);
  MaxTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTestErr',
DefaultErrorValue);
  MidTestError:= FNnwFile.ReadBool('Phase4', 'MidTestErr', False);
  MidTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTestErr',
DefaultErrorValue);
  TestIdent:= FNnwFile.ReadBool('Phase4', 'TestIdent', False);
```

```
  TestIdentCount:= FNnwFile.ReadInteger('Phase4', 'TestIdent',
DefaultTestIdentCount);
end;

procedure TNeuralNetExtended.LoadNetwork;
var
  i,j,k: integer;
  xLayerCount: integer;
begin
  // знищується поточна конфігурація нейромережі
  ResetLayers;
  Alpha := FNnwFile.ReadFloat('Network', 'Alpha', DefaultAlpha);
  Momentum := FNnwFile.ReadFloat('Network', 'Miu', DefaultMomentum);
  TeachRate := FNnwFile.ReadFloat('Network', 'TeachSpeed', DefaultTeachRate);
  EpochCount := FNnwFile.ReadInteger('Network', 'Epoch', DefaultEpochCount);
  xLayerCount := FNnwFile.ReadInteger('Network','CountLayers',
DefaultLayerCount);
  // задається кількість нейронів у шарах
  AutoInit := False;
  for i := 0 to xLayerCount - 1 do
    AddLayer(FNnwFile.ReadInteger('Network','Layer_'+IntToStr(i),
DefaultNeuronCount));
  AutoInit := True;
  // ініціалізація нової конфігурації нейромережі
  Init;
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Weights[k] := FNnwFile.ReadFloat('Network',
                                'W_'+IntToStr( i-
1)+'_'+IntToStr(k)+'_'+IntToStr(j), 0);
      LayersBP[i].NeuronsBP[j].Weights[LayersBP[ i-1].NeuronCount] :=
FNnwFile.ReadFloat('Network',
       'WT_'+IntToStr( i-1)+'_'+IntToStr(j), 0);
    end;
end;

procedure TNeuralNetExtended.SavePhase1;
begin
  FNnwFile.WriteString('Phase1', 'LearnSampleFileName', FNeuroDataSource.Name);
end;

procedure TNeuralNetExtended.SavePhase2;
var
  i: integer;
begin
  FNnwFile.WriteInteger('Phase2', 'AvailableFieldsCount',
FAvailableFieldsCount);
  for i := 0 to AvailableFieldsCount - 1 do
  with Fields[i] do
  begin
    FNnwFile.WriteString('Phase2', 'FieldName_'+IntToStr(i), Name);
    FNnwFile.WriteInteger('Phase2', 'FieldType_'+IntToStr(i), Kind);
    FNnwFile.WriteInteger('Phase2', 'NormType_'+IntToStr(i), NormType);
    FNnwFile.WriteFloat('Phase2', 'Max_'+IntToStr(i), ValueMax);
    FNnwFile.WriteFloat('Phase2', 'Min_'+IntToStr(i), ValueMin);
    FNnwFile.WriteFloat('Phase2', 'Mid_'+IntToStr(i), ValueMid);
    FNnwFile.WriteFloat('Phase2', 'Disp_'+IntToStr(i), Dispersion);
    FNnwFile.WriteFloat('Phase2', 'Alpha_'+IntToStr(i), Alpha);
    FNnwFile.WriteBool('Phase2', 'Ind_'+IntToStr(i), Ind);
  end;
end;

procedure TNeuralNetExtended.SavePhase4;
begin
  FNnwFile.WriteBool('Phase4', 'Epoch', Epoch);
  FNnwFile.WriteInteger('Phase4', 'Epoch', EpochCount);
```

```
  FNnwFile.WriteFloat('Phase4', 'IdentErr', IdentError);
  FNnwFile.WriteBool('Phase4', 'MaxTeachErr', MaxTeachError);
  FNnwFile.WriteFloat('Phase4', 'MaxTeachErr', MaxTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MaxTestErr', MaxTestError);
  FNnwFile.WriteFloat('Phase4', 'MaxTestErr', MaxTestErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTeachErr', MidTeachError);
  FNnwFile.WriteFloat('Phase4', 'MidTeachErr', MidTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTestErr', MidTestError);
  FNnwFile.WriteFloat('Phase4', 'MidTestErr', MidTestErrorValue);
  FNnwFile.WriteFloat('Phase4', 'Miu', Momentum);
  FNnwFile.WriteBool('Phase4', 'TeachIdent', TeachIdent);
  FNnwFile.WriteFloat('Phase4', 'TeachSpeed', TeachRate);
  FNnwFile.WriteInteger('Phase4', 'TeachIdent', TeachIdentCount);
  FNnwFile.WriteBool('Phase4', 'TestAsValid', TestAsValid);
  FNnwFile.WriteBool('Phase4', 'TestIdent', TestIdent);
  FNnwFile.WriteInteger('Phase4', 'TestIdent', TestIdentCount);
  FNnwFile.WriteInteger('Phase4', 'UseForTeach', UseForTeach);
end;

procedure TNeuralNetExtended.SaveNetwork;
var
  i, j, k: integer;
begin
  FNnwFile.WriteFloat('Network', 'TeachSpeed', TeachRate);
  FNnwFile.WriteFloat('Network', 'Miu', Momentum);
  FNnwFile.WriteFloat('Network', 'Alpha', Alpha);
  FNnwFile.WriteInteger('Network', 'Epoch', EpochCount);
  FNnwFile.WriteInteger('Network','CountLayers', LayerCount);
  // задається кількість нейронів у шарах
  for i := 0 to LayerCount - 1 do
    FNnwFile.WriteInteger('Network','Layer_'+IntToStr(i),
StrToInt(NeuronsInLayer[i]));
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to StrToInt(NeuronsInLayer[i]) - 1 do
    begin
      for k := 0 to StrToInt(NeuronsInLayer[ i-1]) do
        FNnwFile.WriteFloat('Network','W_'+IntToStr( i-1)+'_'+IntToStr(k)+
                            '_'+IntToStr(j),
LayersBP[i].NeuronsBP[j].Weights[k]);
      FNnwFile.WriteFloat('Network','WT_'+IntToStr( i-1)+'_'+IntToStr(j),
         LayersBP[i].NeuronsBP[j].Weights[StrToInt(NeuronsInLayer[j])]);
   end;
end;

procedure TNeuralNetExtended.NormalizeData;
var
  i: integer;
begin
  // нормалізація вхідних і вихідних значень
  for i := 0 to FAvailableFieldsCount - 1 do
  begin
    FFields[i].FindMinMax;
    FFields[i].Normalize;
  end;
end;

procedure TNeuralNetExtended.Train;
var
  i, j, k: integer;
begin
  if FUseForTeach = 100 then
  begin
    PatternCount := FFields[0].DataInCount;
    TestSetPatternCount := 0;
  end
  else
  begin
    PatternCount := Round((FFields[0].DataInCount - 1) * FUseForTeach / 100);
```

```
    TestSetPatternCount := FFields[0].DataInCount - PatternCount;
  end;
  if not TeachStopped then
    NormalizeData;
  // формування вхідних значень навчальної множини
  RealOutputIndexCount := OutputFieldCount;
  RealInputIndexCount := InputFieldCount;
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := 0 to PatternCount - 1 do
        FPatternsInput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
      RealInputIndex[k] := i;
      Inc(k);
    end;
  // формування вихідних значень навчальної множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
      for j := 0 to PatternCount - 1 do
        FPatternsOutput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
      RealOutputIndex[k] := i;
      Inc(k);
    end;
  // формування вхідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatterns[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // формування вихідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatternsOut[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // навчання або донавчання мережі для системи Network Function Virtualization
для операторів зв’язку
  TeachOffLine;
end;

procedure TNeuralNetExtended.SetAvailableFieldsCount(Value : integer);
var
  i: integer;
begin
  FAvailableFieldsCount := Value;
  // встановлюється кількість полів
  SetLength(FFields, Value);
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i] := TNeuroField.Create;
end;

procedure TNeuralNetExtended.SetFields(Index: integer; Value: TNeuroField);
begin
  try
    FFields[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SFieldIndexRange, [Index])
```

```
  end;
end;

procedure TNeuralNetExtended.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Epoch := False;
  IdentError:= DefaultErrorValue;
  MaxTeachError := False;
  MaxTeachErrorValue := DefaultErrorValue;
  MaxTestError:= False;
  MaxTestErrorValue:= DefaultErrorValue;
  MidTestError:= False;
  MidTestErrorValue:= DefaultErrorValue;
  MidTeachError := False;
  MidTeachErrorValue :=  DefaultErrorValue;
  SettingsLoaded := False;
  TeachIdent := False;
  TeachIdentCount:= DefaultTeachIdentCount;
  TestAsValid := False;
  TestIdent:= False;
  TestIdentCount:= DefaultTestIdentCount;
  UseForTeach := DefaultUseForTeach;
end;

procedure TNeuralNetExtended.SetFileName(Value: TFilename);
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  try
    FNnwFile := TIniFile.Create(Value);
    FFileName := Value;
  except
    on E: EInOutError do
      raise E.CreateFmt(SWrongFileName, [Value]);
  end;
  FNeuroDataSource := TNeuroDataSource.Create;
  LoadPhase1;
  LoadPhase2;
  LoadPhase4;
  LoadNetwork;
  LoadDataFrom;
end;

procedure TNeuralNetExtended.SetTeachIdentCount(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FTeachIdentCount := DefaultTeachIdentCount
  else
    FTeachIdentCount := Value;
end;

procedure TNeuralNetExtended.SetUseForTeach(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FUseForTeach := DefaultUseForTeach
  else
    FUseForTeach := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndex(Index: integer; const Value:
integer);
begin
  FRealOutputIndex[Index] := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndexCount(const Value: integer);
begin
  SetLength(FRealOutputIndex, Value)
```

```
end;
procedure TNeuralNetExtended.SetRealInputIndex(Index: integer; const Value:
integer);
begin
  FRealInputIndex[Index] := Value;
end;
procedure TNeuralNetExtended.SetRealInputIndexCount(const Value: integer);
begin
  SetLength(FRealInputIndex, Value)
end;
procedure Register;
begin
  RegisterComponents('Neural_network_', [TNeuralNetHopf, TNeuralNetBP,
TNeuralNetExtended]);
end;
end.
```

**Pumpdata.pas – формування бази знань**

```
unit PumpData;

interface

uses
  SysUtils, IniFiles, Classes, Neural_network_Types;

type

  EFieldNormError = class(Exception);
  EFieldKindError = class(Exception);

  TNeuroField = class;
  TNeuroFields = array of TNeuroField;

  TNeuroField = class(TObject)
  private
    FAlpha: double;
    FDataIn: TVectorFloat;
    FDispersion: double;
    FInd: boolean;
    FKind: byte;
    FName: string;
    FNormType: byte;
    FValueMax: double;
    FValueMid: double;
    FValueMin: double;
    function GetDataIn(Index: integer): double;
    function GetKindName: TNeuroFieldType;
    function GetNormTypeName: TNormalize;
    function GetDataInCount: integer;
    procedure SetDataIn(Index: integer; Value: double);
    procedure SetKind(Value: byte);
    procedure SetNormType(Value: byte);
    procedure SetDataInCount(Value: integer);
  public
    procedure FindMinMax;
    procedure CalcMid;
    procedure CalcDispersion;
    procedure Normalize;
    procedure DeNormalize;
    property Alpha: double read FAlpha write FAlpha;
    property DataIn[Index: integer]: double read GetDataIn write SetDataIn;
    property DataInCount: integer read GetDataInCount write SetDataInCount;
    property Dispersion: double read FDispersion write FDispersion;
    property Ind: boolean read FInd write FInd;
    property Kind: byte read FKind write SetKind;
    property KindName: TNeuroFieldType read GetKindName;
    property Name: string read FName write FName;
    property NormType: byte read FNormType write SetNormType;
    property NormTypeName: TNormalize read GetNormTypeName;
    property ValueMax: double read FValueMax write FValueMax;
    property ValueMin: double read FValueMin write FValueMin;
    property ValueMid: double read FValueMid write FValueMid;
  end;

  TNeuroDataSource = class(TObject)
  private
    FName: TFileName;
    function IsHeaderChar(AValue: char): boolean;
  public
    function FieldCount(AHeader: string): integer;
    procedure ExtractHeaders(const AFields: TNeuroFields; AHeader: string);
    procedure ExtractValues(const AVector:TVectorFloat; AHeader: string);
    property Name: TFileName read FName write FName;
  end;
```

```
implementation

{ Клас TNeuroField }

function TNeuroField.GetDataIn(Index: integer): double;
begin
  Result := FDataIn[Index];
end;

function TNeuroField.GetDataInCount: integer;
begin
  Result := High(FDataIn) + 1;
end;

function TNeuroField.GetKindName: TNeuroFieldType;
begin
  case FKind of
    0 : Result := fdInput;
    1 : Result := fdOutput;
    2 : Result := fdNone;
  end;
end;

function TNeuroField.GetNormTypeName: TNormalize;
begin
  case FNormType of
    0 : if KindName = fdInput then
          Result := nrmLinear
        else if KindName = fdOutput then
          Result := nrmLinearOut;
    1 : Result := nrmSigmoid;
    2 : Result := nrmAuto;
    3 : Result := nrmNone;
  end;
end;

procedure TNeuroField.CalcMid;
var
  i: integer;
begin
  FValueMid := 0;
  for i := Low(FDataIn) to High(FDataIn) do
    FValueMid := FValueMid + FDataIn[i];
  FValueMid := FValueMid/(High(FDataIn) + 1);
end;

procedure TNeuroField.CalcDispersion;
var
  i: integer;
begin
  if High(FDataIn) > 1 then
  begin
    FDispersion := 0;
    for i := Low(FDataIn) to High(FDataIn) do
      FDispersion := FDispersion + sqr(FDataIn[i] - ValueMid);
    FDispersion := sqrt(FDispersion/High(FDataIn));
  end
  else
    FDispersion := 0;
end;

(*procedure TNeuroField.DeNormalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
```

```
                FDataIn[i] := (FDataIn[i] + 1)*(FValueMax - FValueMin)/2 +
FValueMin;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := FDataIn[i]*(FValueMax - FValueMin) + FValueMin;
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := - Ln(1/FDataIn[i] - 1)/Alpha;
  end;
end;*)

procedure TNeuroField.FindMinMax;
var
  i: integer;
begin
  FValueMax:= FDataIn[0];
  FValueMin:= FDataIn[0];
  for i := 1 to High(FDataIn) do
  begin
    if FValueMin > FDataIn[i] then
      FValueMin := FDataIn[i];
    if FValueMax < FDataIn[i] then
      FValueMax := FDataIn[i]
  end;
end;

procedure TNeuroField.Normalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmAuto:  begin
                CalcMid;
                CalcDispersion;
                for i := Low(FDataIn) to High(FDataIn) do
                begin
                  xTmp := (FDataIn[i] - FValueMid)/FDispersion;
                  FDataIn[i] := 1/(1 + exp(-xTmp));
                end;
              end;
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := 2*(FDataIn[i] - FValueMin)/(FValueMax - FValueMin)-
1;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := (FDataIn[i] - FValueMin)/(FValueMax - FValueMin);
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := 1/( 1 + exp(-Alpha * FDataIn[i]));
  end;
end;

procedure TNeuroField.SetNormType(Value: byte);
begin
  if (Value < 0) or (Value > 3) then
    raise EFieldNormError.CreateFmt(SFieldNorm, [Value])
  else
    FNormType := Value;
end;

procedure TNeuroField.SetKind(Value: byte);
begin
  if (Value < 0) or (Value > 2) then
    raise Exception.CreateFmt(SFieldKind, [Value])
  else
    FKind := Value;
end;

procedure TNeuroField.SetDataIn(Index: integer; Value: double);
begin
  FDataIn[Index] := Value;
end;
```

```
procedure TNeuroField.SetDataInCount(Value: integer);
begin
  SetLength(FDataIn, Value)
end;

{ Клас TNeuroDataSource }

function TNeuroDataSource.IsHeaderChar(AValue: char): boolean;
begin
  if (AValue in Letters) or (AValue in Capitals) or (AValue in DigitChars) then
    Result := True
  else
    Result := False;
end;

procedure TNeuroDataSource.ExtractValues(const AVector:TVectorFloat; AHeader:
string);
var
  s: string;
  i, xCurPos: integer;
begin
  i := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  try
    while xCurPos > 0 do
    begin
      s := Copy(AHeader, 1, xCurPos - 1);
      AVector[i] := StrToFloat(s);
      Inc(i);
      Delete(AHeader, 1, xCurPos - 1);
      AHeader := Trim(AHeader);
      xCurPos := Pos(SpaceChar, AHeader);
    end;
    s := AHeader;
    AVector[i] := StrToFloat(s);
  except
    on EConvertError do
      EConvertError.CreateFmt(SCannotBeNumber, [s])
  end;
end;

procedure TNeuroDataSource.ExtractHeaders(const AFields: TNeuroFields; AHeader:
string);
var
  s: string;
  xFieldCount, j, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    s := Copy(AHeader, 1, xCurPos - 1);
    AFields[xFieldCount].FName := '';
    for j := 1 to Length(s) do
      if isHeaderChar(s[j]) then
        AFields[xFieldCount].FName := AFields[xFieldCount].FName + s[j];
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  AFields[xFieldCount].FName := '';
  for j := 1 to Length(AHeader) do
    if isHeaderChar(AHeader[j]) then
      AFields[xFieldCount].FName := AFields[xFieldCount].FName + AHeader[j];
```

```
  { повертає кількість полів }
end;

function TNeuroDataSource.FieldCount(AHeader: string): integer;
var
  xFieldCount, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  { повертає кількість полів }
  Result := xFieldCount + 1;
end;


end.
```

**NeuralNetExtend.pas – навчання мережі для системи Network Function Virtualization для операторів зв'язку**

```
unit NeuralNetExtend;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, Neural_network_Comp, ExtCtrls, StdCtrls, Spin, Grids,
Neural_network_Types,
  IniFiles;

const
  FormCaption = 'Навчання мережі для системи Network Function Virtualization для
операторів зв'язку';

type
  TfrmNeuralNetExtend = class(TForm)
    PageControl: TPageControl;
    pnlNavigation: TPanel;
    Tab1: TTabSheet;
    btnBack: TButton;
    rgrFileType: TRadioGroup;
    btnNext: TButton;
    btnCancel: TButton;
    Tab2: TTabSheet;
    lblFileName: TLabel;
    btnOpenFile: TButton;
    edtFileName: TEdit;
    OpenDialog: TOpenDialog;
    Tab3: TTabSheet;
    ltbFieldName: TListBox;
    Label2: TLabel;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    GroupBox1: TGroupBox;
    Label3: TLabel;
    edtMin: TEdit;
    Label4: TLabel;
    edtMax: TEdit;
    Label5: TLabel;
    edt: TEdit;
    Tab4: TTabSheet;
    speLayers: TSpinEdit;
    Label6: TLabel;
    stgNeuronsInLayer: TStringGrid;
    Label7: TLabel;
    Tab5: TTabSheet;
    Label8: TLabel;
    tbrAlpha: TTrackBar;
    sttAlpha: TStaticText;
    Label9: TLabel;
    edtMomentum: TEdit;
    Label10: TLabel;
    edtTeachRate: TEdit;
    Tab6: TTabSheet;
    Label11: TLabel;
    Label12: TLabel;
    btnContinueTeach: TButton;
    sttMaxTeachError: TStaticText;
    sttEpochCount: TStaticText;
    GroupBox2: TGroupBox;
    Label13: TLabel;
    edtIdentError: TEdit;
    speEpochCount: TSpinEdit;
    cbxEpoch: TCheckBox;
```

```
cbxMaxTeachError: TCheckBox;
edtMaxTeachErrorValue: TEdit;
cbxMidTeachError: TCheckBox;
edtMidTeachErrorValue: TEdit;
cbxTeachIdent: TCheckBox;
speTeachIdentValue: TSpinEdit;
btnBeginTeach: TButton;
cbxMaxTestError: TCheckBox;
cbxMidTestError: TCheckBox;
cbxTestIdent: TCheckBox;
edtMaxTestErrorValue: TEdit;
edtMidTestErrorValue: TEdit;
speTestIdentValue: TSpinEdit;
Tab7: TTabSheet;
Label14: TLabel;
stgInput: TStringGrid;
Label15: TLabel;
stgOutput: TStringGrid;
btnCompute: TButton;
Memo1: TMemo;
Label16: TLabel;
Label17: TLabel;
Memo2: TMemo;
Bevel1: TBevel;
Memo3: TMemo;
Label1: TLabel;
sttMaxTestError: TStaticText;
Label18: TLabel;
sttMidTeachError: TStaticText;
Label19: TLabel;
sttMidTestError: TStaticText;
SaveDialog: TSaveDialog;
edtUseForTeach: TEdit;
Label20: TLabel;
btnSave: TButton;
procedure btnCancelClick(Sender: TObject);
procedure btnNextClick(Sender: TObject);
procedure btnBackClick(Sender: TObject);
procedure btnOpenFileClick(Sender: TObject);
procedure FormActivate(Sender: TObject);
procedure ltbFieldNameClick(Sender: TObject);
procedure rdgFieldTypeClick(Sender: TObject);
procedure rdgNormTypeClick(Sender: TObject);
procedure edtAChange(Sender: TObject);
procedure tbrAlphaChange(Sender: TObject);
procedure edtMomentumChange(Sender: TObject);
procedure edtTeachRateChange(Sender: TObject);
procedure NeuralNetExtendedEpochPassed(Sender: TObject);
procedure btnContinueTeachClick(Sender: TObject);
procedure edtIdentErrorChange(Sender: TObject);
procedure cbxEpochClick(Sender: TObject);
procedure speEpochCountChange(Sender: TObject);
procedure cbxMaxTeachErrorClick(Sender: TObject);
procedure edtMaxTeachErrorValueChange(Sender: TObject);
procedure cbxMidTeachErrorClick(Sender: TObject);
procedure edtMidTeachErrorValueChange(Sender: TObject);
procedure cbxTeachIdentClick(Sender: TObject);
procedure speTeachIdentValueChange(Sender: TObject);
procedure cbxMaxTestErrorClick(Sender: TObject);
procedure edtMaxTestErrorValueChange(Sender: TObject);
procedure cbxMidTestErrorClick(Sender: TObject);
procedure edtMidTestErrorValueChange(Sender: TObject);
procedure cbxTestIdentClick(Sender: TObject);
procedure speTestIdentValueChange(Sender: TObject);
procedure NeuralNetExtendedAfterTeach(Sender: TObject);
procedure btnBeginTeachClick(Sender: TObject);
procedure btnComputeClick(Sender: TObject);
procedure speLayersChange(Sender: TObject);
procedure btnSaveClick(Sender: TObject);
```

```
    procedure edtUseForTeachChange(Sender: TObject);
  private
    { Private declarations }
    Teach: boolean;
    NotSaved: boolean;
    procedure ChangePage;
    procedure OpenFile;
    procedure LoadFile;
    procedure Tune;
    procedure CreateNet;
    procedure RunTeach;
    procedure TestNet;
  public
    { Public declarations }
  end;

var
  frmNeuralNetExtend: TfrmNeuralNetExtend;
implementation

{$R *.DFM}

// Запуск програми
procedure TfrmNeuralNetExtend.FormActivate(Sender: TObject);
begin
  Caption := FormCaption;
  PageControl.ActivePage := PageControl.Pages[0];
  Teach := false;
  NotSaved := false;
end;

// Вихід із програми
procedure TfrmNeuralNetExtend.btnCancelClick(Sender: TObject);
begin
  if NotSaved then
    if MessageDlg('Є незбережені дані. Зберегти?',mtConfirmation, [mbYes, mbNo],
0) = mrYes then
      btnSave.Click;
  Close;
end;

// Натискання кнопки "Вперед"
procedure TfrmNeuralNetExtend.btnNextClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, true, false);
    ChangePage;
    if ActivePage.PageIndex = PageCount - 1 then
      btnNext.Enabled := false
    else
      btnNext.Enabled := true;
    btnBack.Enabled := true;
  end;
end;

// Натискання кнопки "Назад"
procedure TfrmNeuralNetExtend.btnBackClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, false, false);
    if ActivePage.PageIndex = 0 then
      btnBack.Enabled := false
    else
      btnBack.Enabled := true;
    btnNext.Enabled := true;
  end;
end;
```

```
// Дія на зміну сторінки
procedure TfrmNeuralNetExtend.ChangePage;
begin
  case PageControl.ActivePage.PageIndex of
    1: OpenFile;
    2: LoadFile;
    3: Tune;
    4: CreateNet;
    6: TestNet;
  end;
end;

// Вибрати файл - джерело даних
procedure TfrmNeuralNetExtend.OpenFile;
begin
  if rgrFileType.ItemIndex = 0 then
  begin
    OpenDialog.Filter := 'NNW files (*.nnw)|*.nnw';
    lblFileName.Caption := 'Виберіть nnw-файл';
  end
  else
  begin
    OpenDialog.Filter := 'Text files (*.txt)|*.txt';
    lblFileName.Caption := 'Виберіть txt-файл';
  end;
end;

// Вибір файлу в діалоговому вікні
procedure TfrmNeuralNetExtend.btnOpenFileClick(Sender: TObject);
begin
  OpenDialog.Execute;
  Caption := FormCaption + ' - ' + ExtractFileName(OpenDialog.FileName);
  edtFileName.Text := OpenDialog.FileName;
end;

// Завантажити в компонент обраний файл
procedure TfrmNeuralNetExtend.LoadFile;
var
  i: integer;
begin
  try
    if rgrFileType.ItemIndex = 0 then
      NeuralNetExtended.FileName := edtFileName.Text   // nnw-файл
    else
    begin
      NeuralNetExtended.SourceFileName := edtFileName.Text; // текстовий файл
      NeuralNetExtended.LoadDataFrom; // завантажує дані з текстового файлу
      // конфігурація нейронної мережі для системи Network Function
Virtualization для операторів зв’язку за замовчуванням
      NeuralNetExtended.AddLayer(2);
      NeuralNetExtended.AddLayer(3);
      NeuralNetExtended.AddLayer(1);
    end;
  except
    raise Exception.Create('Помилка при відкритті файлу');
  end;
  NeuralNetExtended.Init;  // Ініціалізація мережі для системи Network Function
Virtualization для операторів зв’язку
  // Формування списку полів для StringList-A
  ltbFieldName.Clear;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    ltbFieldName.Items.Add(NeuralNetExtended.Fields[i].Name);
  ltbFieldName.ItemIndex := 0;
  ltbFieldNameClick(Self);
end;

// Настроювання параметрів мережі для системи Network Function Virtualization
для операторів зв’язку
```

```
procedure TfrmNeuralNetExtend.Tune;
var
  i: integer;
begin
  speLayers.Value := NeuralNetExtended.LayerCount - 2;
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
  stgNeuronsInLayer.Cells[0, 0] := '№ шаруючи';
  stgNeuronsInLayer.Cells[1, 0] := 'Нейронів';
  for i := 0 to speLayers.Value - 1 do
  begin
    stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
    stgNeuronsInLayer.Cells[1, i + 1] := IntToStr(NeuralNetExtended.Layers[i +
1].NeuronCount);
  end;

  tbrAlpha.Position := trunc(NeuralNetExtended.Alpha * 100);
  edtMomentum.Text := FloatToStr(NeuralNetExtended.Momentum);
  edtTeachRate.Text := FloatToStr(NeuralNetExtended.TeachRate);
  edtIdentError.Text := FloatToStr(NeuralNetExtended.IdentError);
  edtUseForTeach.Text := FloatToStr(NeuralNetExtended.UseForTeach);

  cbxEpoch.Checked := NeuralNetExtended.Epoch;
  speEpochCount.Text := IntToStr(NeuralNetExtended.EpochCount);

  cbxMaxTeachError.Checked := NeuralNetExtended.MaxTeachError;
  edtMaxTeachErrorValue.Text :=
FloatToStr(NeuralNetExtended.MaxTeachErrorValue);

  cbxMidTeachError.Checked := NeuralNetExtended.MidTeachError;
  edtMidTeachErrorValue.Text :=
FloatToStr(NeuralNetExtended.MidTeachErrorValue);

  cbxTeachIdent.Checked := NeuralNetExtended.TeachIdent;
  speTeachIdentValue.Value := NeuralNetExtended.TeachIdentCount;

  cbxMaxTestError.Checked := NeuralNetExtended.MaxTestError;
  edtMaxTestErrorValue.Text := FloatToStr(NeuralNetExtended.MaxTestErrorValue);

  cbxMidTestError.Checked := NeuralNetExtended.MidTestError;
  edtMidTestErrorValue.Text := FloatToStr(NeuralNetExtended.MidTestErrorValue);

  cbxTestIdent.Checked := NeuralNetExtended.TestIdent;
  speTestIdentValue.Value := NeuralNetExtended.TeachIdentCount;
end;

// Відображення інформації про обране поле (тип поля, нормалізація та інше)
procedure TfrmNeuralNetExtend.ltbFieldNameClick(Sender: TObject);
begin
  // Тип поля - вхідне, вихідне, не використовувати
  rdgFieldType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  // Тип нормалізації
  rdgNormType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  // Параметр нормалізації
  edt.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  // Мінінум та максимум (для лінійної нормалізації)
  edtMin.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin);
  edtMax.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax);
end;

// Змінити тип поля
procedure TfrmNeuralNetExtend.rdgFieldTypeClick(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
```

```
end;

// Змінити тип нормалізації
procedure TfrmNeuralNetExtend.rdgNormTypeClick(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
end;

// Змінити параметр нормалізації
procedure TfrmNeuralNetExtend.edtAChange(Sender: TObject);
begin
 NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha := StrToFloat(edt.Text);
end;

// Змінити параметр Alpha мережі для системи Network Function Virtualization для
операторів зв’язку
procedure TfrmNeuralNetExtend.tbrAlphaChange(Sender: TObject);
begin
  sttAlpha.Caption := FloatToStr(tbrAlpha.Position / 100);
end;

// Зміна кількості схованих шарів
procedure TfrmNeuralNetExtend.speLayersChange(Sender: TObject);
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
end;

// Створення мережі для системи Network Function Virtualization для операторів
зв’язку обраної топології
procedure TfrmNeuralNetExtend.CreateNet;
var
  i: integer;
  xInput, xOutput: integer;
begin
  // Змінюється тільки кількість нейронів у схованих шарах,
  // Кількість нейронів у вхідному й вихідному шарі залежить від
  // типів полів
  with NeuralNetExtended do
  begin
    xInput := InputFieldCount;
    xOutput := OutputFieldCount;
    ResetLayers;
    AddLayer(xInput);
    for i := 0 to speLayers.Value - 1 do
      AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i + 1]));
    AddLayer(xOutput);
  end;
end;

// Змінити момент мережі для системи Network Function Virtualization для
операторів зв’язку
procedure TfrmNeuralNetExtend.edtMomentumChange(Sender: TObject);
begin
  NeuralNetExtended.Momentum := StrToFloat(edtMomentum.Text);
end;

// Змінити швидкість навчання мережі для системи Network Function Virtualization
для операторів зв’язку
procedure TfrmNeuralNetExtend.edtTeachRateChange(Sender: TObject);
begin
  NeuralNetExtended.TeachRate := StrToFloat(edtTeachRate.Text);
end;

// Дія по проходженню однієї епохи навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedEpochPassed(Sender: TObject);
begin
  sttMaxTestError.Caption := '';
  sttMidTestError.Caption := '';
```

```
  with NeuralNetExtended do
  begin
    sttMaxTeachError.Caption := FloatToStr(MaxTeachResidual); // Показати макс.
помилку на навчальній множині
    sttMidTeachError.Caption := FloatToStr(MidTeachResidual); // Показати серед.
помилку на навчальній множині
    if NeuralNetExtended.UseForTeach <> 100 then
    begin
      sttMaxTestError.Caption := FloatToStr(MaxTestResidual); // Показати макс.
помилку на тестовій множині
      sttMidTestError.Caption := FloatToStr(MidTestResidual); // Показати сред.
помилку на тестовій множині
    end;
    sttEpochCount.Caption := FloatToStr(EpochCurrent); // Показати номер
поточної епохи
  end;
  Application.ProcessMessages; // Дати можливість Windows перемалювати форму
end;

// Натискання кнопки "Продовжити навчання"
procedure TfrmNeuralNetExtend.btnContinueTeachClick(Sender: TObject);
begin
  NeuralNetExtended.ContinueTeach := true; // Скинути прапор - "Продовжити
навчання"
  RunTeach;  // Запуск
end;

// Натискання кнопки "Навчити"
procedure TfrmNeuralNetExtend.btnBeginTeachClick(Sender: TObject);
begin
  NeuralNetExtended.ContinueTeach := false; // Скинути прапор - "Почати навчання
знову"
  RunTeach;
end;

procedure TfrmNeuralNetExtend.edtIdentErrorChange(Sender: TObject);
begin
  NeuralNetExtended.IdentError := StrToFloat(edtIdentError.Text);
end;

procedure TfrmNeuralNetExtend.edtUseForTeachChange(Sender: TObject);
begin
  NeuralNetExtended.UseForTeach := StrToInt(edtUseForTeach.Text);
end;

procedure TfrmNeuralNetExtend.cbxEpochClick(Sender: TObject);
begin
  NeuralNetExtended.Epoch := cbxEpoch.Checked;
end;

procedure TfrmNeuralNetExtend.speEpochCountChange(Sender: TObject);
begin
  NeuralNetExtended.EpochCount := StrToInt(speEpochCount.Text);
end;

procedure TfrmNeuralNetExtend.cbxMaxTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTeachError := cbxMaxTeachError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMaxTeachErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MaxTeachErrorValue :=
StrToFloat(edtMaxTeachErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxMidTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTeachError := cbxMidTeachError.Checked;
```

```
end;

procedure TfrmNeuralNetExtend.edtMidTeachErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MidTeachErrorValue :=
StrToFloat(edtMidTeachErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxTeachIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TeachIdent := cbxTeachIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTeachIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTeachIdentValue.Value;
end;

procedure TfrmNeuralNetExtend.cbxMaxTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTestError := cbxMaxTestError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMaxTestErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MaxTestErrorValue := StrToFloat(edtMaxTestErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxMidTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTestError := cbxMidTestError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMidTestErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MidTestErrorValue := StrToFloat(edtMidTestErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxTestIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TestIdent := cbxTestIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTestIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTestIdentValue.Value;
end;

// Зупинка навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedAfterTeach(Sender: TObject);
begin
  btnBack.Enabled := true;
  btnNext.Enabled := true;
  btnCancel.Enabled := true;
  btnContinueTeach.Caption := 'Навчити';
  NeuralNetExtended.StopTeach := true;
end;

procedure TfrmNeuralNetExtend.RunTeach;
begin
  Teach := not Teach;  // Перемикач стану "вчимося/не вчимося"
  if Teach then
  begin
    btnBeginTeach.Enabled := false;
    btnBack.Enabled := false;
    btnNext.Enabled := false;
    btnCancel.Enabled := false;
    NeuralNetExtended.StopTeach := false;
    btnContinueTeach.Caption := 'Зупинити навчання';
```

```
    NotSaved := true;
    NeuralNetExtended.Train;  // Запуск нейромережі на навчання
    btnCompute.Enabled := true;
    btnSave.Enabled := true;
  end
  else
  begin
    btnBeginTeach.Enabled := true;
    btnBack.Enabled := true;
    btnNext.Enabled := true;
    btnCancel.Enabled := true;
    btnContinueTeach.Caption := 'Продовжити навчання';
    NeuralNetExtended.StopTeach := true; // Зупинити навчання
  end;
end;

// Відкриття сторінки - тестування навченої нейромережі
procedure TfrmNeuralNetExtend.TestNet;
var
  i, j: integer;
begin
  stgInput.RowCount := NeuralNetExtended.InputFieldCount + 1;
  stgInput.Cells[0, 0] := 'Поле';
  stgInput.Cells[1, 0] := 'Значення';

  // Проставити імена вхідних полів
  j := 0;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    if (NeuralNetExtended.Fields[i].KindName = fdInput) then // Ознака того, що
поле вхідне
    begin
      Inc(j);
      stgInput.Cells[0, j] := NeuralNetExtended.Fields[i].Name;
    end;
  stgOutput.RowCount := NeuralNetExtended.OutputFieldCount + 1;
  stgOutput.Cells[0, 0] := 'Поле';
  stgOutput.Cells[1, 0] := 'Значення';

  // Проставити імена вихідних полів
  j := 0;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    if (NeuralNetExtended.Fields[i].KindName = fdOutput) then  // Ознака того,
що поле вихідне
    begin
      Inc(j);
      stgOutput.Cells[0, j] := NeuralNetExtended.Fields[i].Name;
    end;
end;

// Натискання кнопки "Обчислити"
procedure TfrmNeuralNetExtend.btnComputeClick(Sender: TObject);
var
  xVectorFloat: TVectorFloat;
  i: integer;
begin
  // Створити вектор, що будемо подавати на вхід
  // довжиною, рівною кількості нейронів на вхідному шарі
  SetLength(xVectorFloat, NeuralNetExtended.InputFieldCount);
  // Заповнити значення елементів вектора
  for i := 0 to NeuralNetExtended.InputFieldCount - 1 do
    xVectorFloat[i] := StrToFloat(stgInput.Cells[1, i + 1]);
  // Подати на вхід нейромережі. Результати будуть у вихідному шарі нейромережі
  NeuralNetExtended.ComputeUnPrepData(xVectorFloat);
  // Відобразити отримані результати
  for i := 0 to NeuralNetExtended.OutputFieldCount - 1 do
    stgOutput.Cells[1, i + 1] := FloatToStr(NeuralNetExtended.Output[i]);
  // Знищити вектор
  SetLength(xVectorFloat, 0);
  xVectorFloat := nil;
```

```
end;

// Зберегти навчену нейромережу
procedure TfrmNeuralNetExtend.btnSaveClick(Sender: TObject);
begin
  SaveDialog.InitialDir := ExtractFilePath(NeuralNetExtended.FileName);
  SaveDialog.FileName := ExtractFileName(NeuralNetExtended.FileName);
  if SaveDialog.Execute then
  begin
    NeuralNetExtended.NnwFile := TIniFile.Create(SaveDialog.FileName);
    NeuralNetExtended.SavePhase1;
    NeuralNetExtended.SavePhase2;
    NeuralNetExtended.SavePhase4;
    NeuralNetExtended.SaveNetwork;
    NotSaved := false;
  end;
end;

end.
```