ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

УДК 004.414

О. В. Багацький

ПРОГРАМНА АРХІТЕКТУРА СИСТЕМИ ДЛЯ ЗБОРУ І ОБРОБКИ ПАРАМЕТРІВ КОМУНАЛЬНИХ ПОСЛУГ

Інститут кібернетики ім. В.М.Глушкова НАН України, м. Київ

Анотація. Запропонована автором архітектура програмної системи, яка була розроблена на основі патернів проектування та описана за допомогою UML-діаграм, дозволяє легко реалізовувати систему для визначення параметрів комунальних послуг, яка складається з однотипних елементів, та зв'язків між ними. Використання запропонованої архітектури дозволить також уніфікувати і вимоги до апаратної складової системи на відповідних ієрархічних рівнях. Як приклад автором була розроблена система, що дозволяє приймати дані з приладу для контролю параметрів електроенергії у споживача в побутовій мережі 220В, оцінювати якість наданої послуги та зберігати отримані результати.

Ключові слова: архітектура програмної системи, комунальні послуги, патерни проектування

Аннотация. Предложенная автором архитектура программной системы, которая была спроектирована на основе паттернов проектирования и описана при помощи UML-диаграмм, позволяет легко реализовывать систему для расчёта параметров коммунальных услуг, которая состоит из однотипных элементов соединенных между собой. Использование предложенной архитектуры позволит также унифицировать и требования к аппаратной составляющей системы на соответствующих иерархических уровнях. В качестве примера автором была разработана система, позволяющая принимать данные с прибора для контроля параметров электроэнергии у потребителя в бытовой сети 220В, оценивать качество предоставляемой услуги и сохранять полученные результаты.

Ключевые слова: архитектура программной системы, коммунальные услуги, паттерны проектирования

Abstract. Proposed by the author a software system architecture, which was designed based on design patterns and described using UML-diagrams, makes it easy to implement a system for calculating parameters of public utilities, which consists of the same elements connected together. Using the proposed architecture will also standardize the requirements for the hardware component of the system according to the appropriate hierarchical levels. To prove that, the author has developed a system that allows getting the data from the device to control the quality of electricity network 220 V, to assess the quality of service provided and to save the recived results.

Key words: architecture of a software system, public utilites, design patterns

Вступ

На сьогодні не існує системи обліку комунальних послуг, яка б могла оцінювати кількість і якість всіх комунальних послуг уніфіковано. Існують окремі системи, які дозволяють вести контроль за параметрами однієї послуги, однак системи які контролюють кількість ї якість всіх послуг немає. Оскільки системи надання комунальних послуг є ієрархічними системами то і системи для контролю комунальних послуг повинні мати таку ж саму ієрархічну структуру. Такі системи інфомаційно-вимірювального типу характеризуються наявністю у структурі вимірювальних пристроїв та приладів, а також цифрової частини, які з'єднуються між собою у певній послідовності [1]. Система повинна розраховувати дані, які є специфічними для кожної комунальної послуги, зберігати отримані результати, видавати їх при запитах та реагувати на аварійні ситуації. Важливо, щоб система контролю комунальних послуг мала можливість масштабування, тобто як додавання одного або декількох елементів на одному з рівнів ієрархії так і додавання ще одного рівня ієрархії. Необхідно передбачити механізми зв'язку, за якими буде відбуватися передача даних між рівнями ієрархії. Для системи бажано також передбачити механізм для налаштування за допомогою оператора.

Актуальність

В наш час в Україні здійснюється переведення абонентських та лічильникових систем контролю параметрів комунальних послуг в централізовані ієрархічні комп'ютерні системи, тому актуальною є задача створення уніфікованого програмного забезпечення для таких систем.

Мета

Метою роботи є розробка архітектури системи, на основі якої можливе створення модульної системи різних рівнів ієрархії, яка дозволяє одночасно проводити як збір, так і обробку даних для всіх комунальних послуг.

Задачі

1. Аналіз архітектури існуючих систем обліку комунальних послуг.
2. Опис пропонованої системи з використанням UML-діаграм.

3. Створення частини системи для одного з рівнів ієрархії, з використанням патернів проектування та урахуванням сформульованих вимог до системи обліку комунальних послуг.

Структура систем, які забезпечують контроль за параметрами комунальної послуги

Розглянемо систему, структура якої наведена на рис.1. Інформація про параметри комунальної послуги надходять від декількох лічильників (приладів), які можуть передавати виміряні параметри комунальної послуги по каналам зв'язку до концентратору, який знаходиться у квартирі. Після обробки даних концентратором, він надсилає ці дані до мультиконцентратору, що обробляє дані з усього будинку, який, в свою чергу, передає оброблені дані на вищий рівень ієрархії.

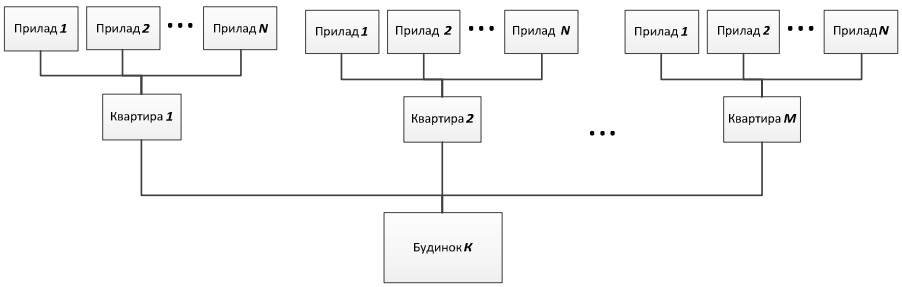

Рисунок 1 – Приклад ієрархічної системи, яка складається з трьох рівнів.

На кожному рівні ієрархії відбувається обмін інформацією – об'єкт нижчого рівня передає отримані дані до об'єкта вищого рівня, де вони обробляються та передаються далі. Бажано створити програму, яку можливо адаптувати для будь-якого рівня ієрархії. Отже, програмна частина системи для контролю комунальних послуг буде складатися з архітектурно однакових частин, з'єднаних ієрархічними зв'язками. Вони будуть відрізнятися на різних рівнях ієрархій об'єктами, від яких отримуються та передаються дані, алгоритмами обробки даних, можливостями збереження даних та взаємодії цих частин з оператором. Такий модульний підхід значно скорочує час розробки всієї програмної частини системи та полегшує подальше обслуговування та модернізацію.

Архітектурні складові програмної частини можуть бути описані за допомогою патернів (patterns) проектування [7]. Патерн проектування – це сукупність класів та зв'язків між ними (за виключенням патерну Singleton, який завжди реалізується одним єдиним класом), які акцентують увагу на особливостях певної реалізації програмного коду. Перевагами патерну є полегшення супроводження програми та можливість повторного використання частин програм при створенні нової програми.

Використання патернів проектування дозволяє абстрагуватись від мови програмування та використовувати програмні конструкції, які вже перевірені на практиці в різних мовах програмування. Для опису конструкцій, за необхідності, можна використовувати UML-діаграми.

Таким чином, достатньо сконструювати архітектуру за допомогою патернів лише для однієї ланки ієрархії, а потім розширити цю архітектуру на всю систему, незалежно від кількості рівнів ієрархії.

Опис архітектури запропонованої системи

На рис.2 зображений рівні ієрархії «Прилад» та «Квартира». Необхідно, щоб на рівні «Квартира» у концентраторі функціонувала програма, яка реалізує функції отримання даних з рівня «Прилад». Оскільки кожен з приладів, які оцінюють параметри комунальної послуги, працюючи у режимі реального часу та надсилає дані асинхронно, доцільно використовувати для отримання результатів мультипоточність.

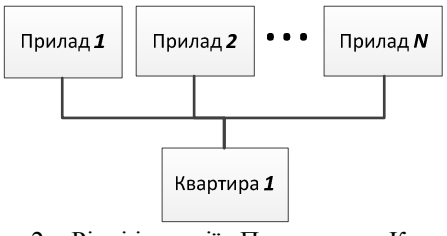

Рисунок 2 – Рівні ієрархії «Прилад» та «Квартира».

Крім того, в програмі необхідно передбачити:

1) кількість даних, що передаються;
2) алгоритми шифрації/дешифрації, корекції/контролю помилок та ін.;
3) буфер даних, в який, при неможливості миттєвої обробки, запам'ятовувались передані дані;
4) унікальний ідентифікаційний код, за допомогою якого можливо визначити конкретний прилад.

Для подальшої передачі даних або довготривалого зберігання їх у концентраторі необхідно передбачити механізм збереження та/або передачі даних на наступний рівень ієрархії.

Також необхідно передбачити наявність програмного модулю який обробляє дані про стан приладів та режим роботи концентратора. В програмі концентратора необхідно передбачити реакцію на вибір режимів роботи з органів керування.

Програмне архітектурне рішення для уніфікованої системи збору та обробки даних комунальних послуг.

Слід зауважити, що у цій статті пропонується лише один спрощений конкретний варіант реалізації архітектури. Можливі і інші архітектурні реалізації, які будуть базуватися на використанні інших патернів проектування.

У запропонованій архітектурі умовно виділяється сім незалежних частин, кожна з яких виконує певні завдання. Такі частини наведені нижче

1. Довідка – дані, які потрібні для інтерпретації даних, отриманих з приладу. Використовується патерн Singleton, оскільки потрібно гарантувати унікальність цієї частини системи та можливість звернення до цієї частини будь-якого іншого елементу.
2. Арбітер – сутність, яка контролює послідовність виконання алгоритмів та доступу до інших частин. Використовується патерн Mediator, який призначений для зменшення зв'язків між класами.
3. Алгоритми – алгоритми, які можуть бути задіяні для всіх об'єктів. Необхідне використання Патерн Strategy використовується для створення сімейства алгоритмів, які не залежать від контексту їх використання.
4. Прилад – під приладом у цій програмній моделі розуміється джерело інформації, яке сумісне з концентратором хоча б на п'яти рівнях (від фізичного до сеансового) моделі ISO/OSI [8]. Для створення такого джерела інформації фізичний прилад повинен бути під'єднаний до входу концентратора. У квартирі можливо використовувати тільки один прилад, який вимірює параметри певної комунальної послуги. Прилади відрізняються один від одного лише кількістю переданих параметрів. Використовуються патерни FactoryMethod, який дозволяє легко контролювати місце появи даних з приладу у коді програми та Critical Section, для використання мультипоточності.
5. Записувач – сутність, яка надає інтерфейси для збереження даних. Використовується патерн FactoryMethod , оскільки він дозволяє легко додавати нові методи збереження даних.
6. Інтерфейс користувача (GUI) – сутність, яка дозволяє приладу для збору даних виводити свої дані для інформування користувача. Використовується патерн Builder, оскільки реалізація інтерфейсу користувача є складною та може змінюватися на різних рівнях ієрархії.
7. Уніфікований інтерфейс передачі даних від користувача до концентратору, що дозволяє налаштування певних елементів системи. Використовується патерн Command, як стандартний патерн для операцій цього типу.

Організація фізичних міжієрархічних зв'язків залежить від апаратної конфігурації системи і не розглядається в даній статті.

На рис. 3 представлена UML-діаграма класів на мові програмування C++, яка спрощено описує систему для збору та обробки даних комунальних послуг. Показані лише декілька варіацій класів-нащадків SomeAlgoritms_SrategyPattern та DeviceAlgoritms_SrategyPattern (програмна реалізація частини «Алгоритми»), Creator_GUI та GUI_BuilderPattern (програмна реалізація інтерфейсу користувача); клас Device_MediatorPattern має тільки одну реалізацію через класи SomeDevice та Device_FactoryMethodPattern; клас Writer представлений тільки одною реалізацією – Writer_to_XML.

Найбільш складним при створенні класів, які показані на рис.3, є реалізація патерну Mediator для класу Arbiter_MediatorPattern. Цей клас є ключовим для взаємодії між собою інших частин програми, тому що саме цей клас реалізує передачу даний на вищий рівень ієрархії.

Клас Devices_MediatorPattern, який реалізує взаємодію між різними класами, створює нові потоки (threads) даних, що, згідно [9], ускладнює внутрішню структуру цієї частини програми.

Клас Init_Knowledge_SingletonPattern містить у собі константи, необхідні для початкової ініціалізації програми на концентраторі.

Клас Init_Device_SingletonPattern містить у собі константи, які потрібні при визначені відомих алгоритмів передачі даних під'єднаних приладів.

При ініціалізації приладу існує два варіанта поведінки «прилад – концентратор»:

1. Активний. Прилад має свій унікальний ідентифікаційний номер, який передає до концентратору, за допомогою потоколу передачі даних. В такому випадку концентратор реєструє наявний унікальний ідентифікатор у масиві ідентифікаторів, формує чергу, за якою опитує прилад та приймає і передає дані до конкретних класів-нащадків GUI_BuilderPattern та Saver. Під’єднаний прилад вважається концентратором «під’єднаним» та «справним».

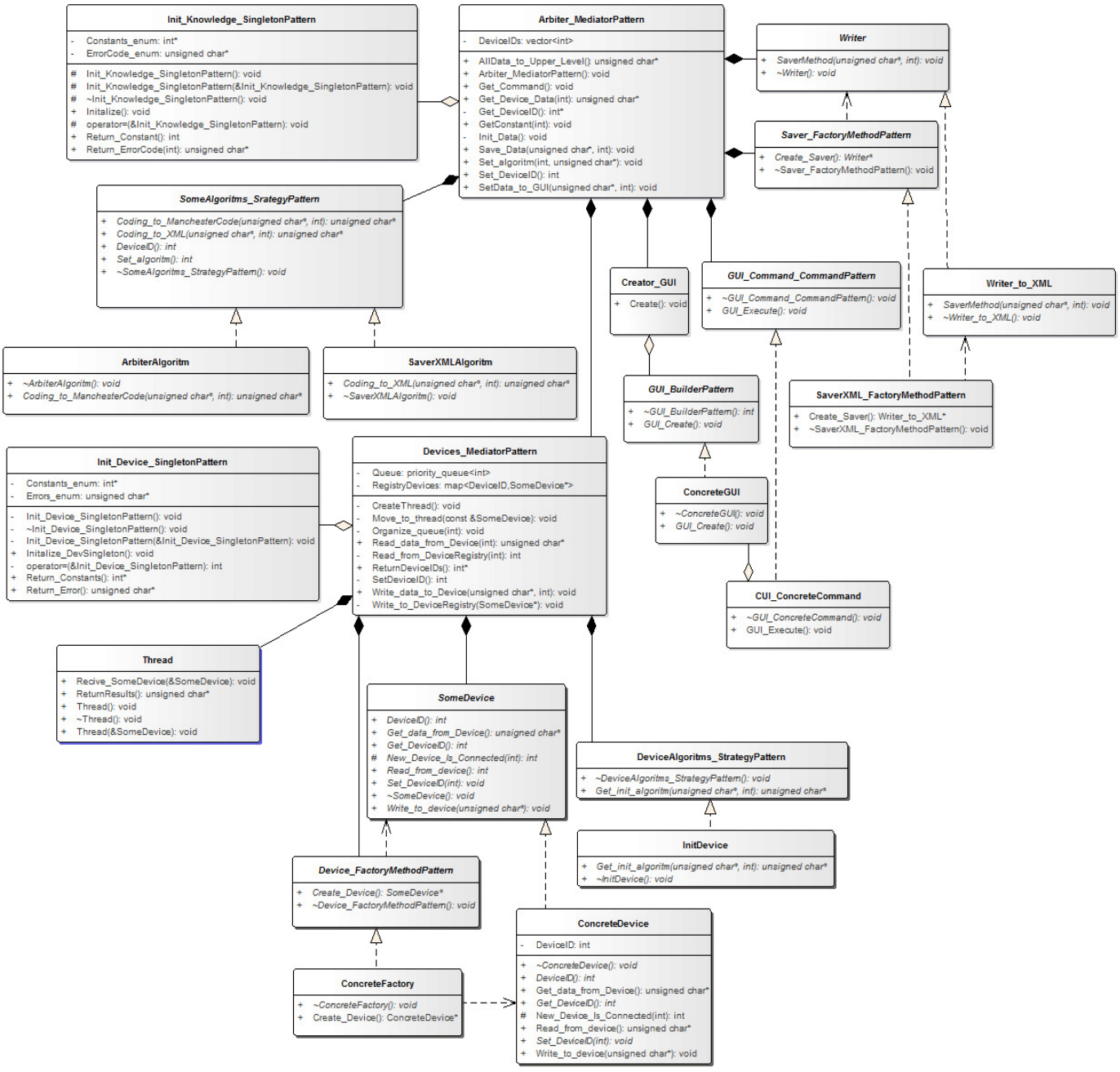

Рисунок 3 – Спрощена UML-діаграма класів системи для збору та обробки даних комунальних послуг.

2. Пасивний. При під’єднанні приладу створюються нові нащадки класу SomeDevice, які за допомогою аналізу даних визначають тип приладу. Якщо аналіз дозволяє ідентифікувати тип приладу та кодування – прилад вважається «під’єднаним». Клас Devices_MediatorPattern створює унікальний ідентифікаційний номер для класу-нащадку SomeDevice, який може базуватися на алгоритмах – нащадках класу DevicesAlgoritms_SrategyPattern. Так само як і у «активному» варіанті поведінки, концентратор, після передачі ідентифікатору до конкретного «під’єднаного» приладу Devices_MediatorPattern, реєструє цей прилад у масиві ідентифікаторів RegistryDevices map<DeviceID, SomeDevice*>, формує чергу Queue

prirority_queue<int>, за якою буде опитуватися прилад та виконувати операції «запису-читання» приладу. Після реєстрації даних «під'єднаний» прилад вважається «під'єднаним» та «справним».

Бажано використовувати «пасивний» варіант поведінки, оскільки такий варіант легко масштабується на наступні рівні ієрархії.

При запуску програми створення класів відбувається у такій послідовності: створення класу Init_Knowledge_SingletonPattern; створення класу Arbiter_MediatorPattern; створення та ініціалізація Algoritms_SrategyPattern та його нащадків; створення класів, що реалізують операції збереження даних (Saver та Saver_FactoryMethod і їх нащадки); створення класів для виводу інформації (GUI_BuilderPattern та його нащадки) і взаємодії з користувачем (GUI_Command_CommandPattern та його нащадки); створення класу Devices_MediatorPattern, після чого створюються класи Init_Device_SingletonPattern, Thread, SomeDevice та Device_FactoryMethodPattern і його нащадки.

Клас Thread реалізує потік (thread) та інкапсулює в собі програмну та апаратну реалізації цього потоку. Використання мультипоточності класом Devices_MediatorPattern переслідує наступні цілі:

1. Отримання даних у режимі реального часу. Дані з під'єднаних приладів обробляються нащадком класу SomeDevice, який переміщується у свій потік після створення та ініціалізації.
2. Обрахування великої кількості даних. На більш високих рівнях архітектури необхідно збирати та обробляти дані з тисяч елементів, для чого можуть бути використані складні багатопроцесорні системи.

Для реалізації мультипоточності класом Thread доцільно використовувати «протопотоки» («protothreads») [10], як компромісний варіант між створенням нестандартної реалізації мультипоточності та аналогом класу «thread» у мові програмування C++ редакції 2011 року [11] або «thread» з бібліотеки boost [12]. Використання стандартних бібліотек значно полегшує подальше обслуговування та модернізацію такої системи. Обслуговування потоків відбувається динамічним способом визначення пріоритетів (наприклад [13], LRU, FIFO та ін.) за допомогою алгоритмів класів-нащадків з DevicesAlgoritms_SrategyPattern. Оскільки в основі цих алгоритмів знаходиться черга та маніпуляції з її індексами, необхідно, щоб такий алгоритм був потокобезпечним [8].

Класи Saver та Saver_FactoryMethod і їх нащадки відповідають за збереження даних приладів та передачу цієї інформації на вищий рівень ієрархії. Збереження даних може відбуватися як у ОЗП так і енергонезалежну пам'ять, тому необхідно передбачити можливі механізми запису та доступу для різних сценаріїв збереження даних (наприклад, кешування [14]).

Класи Creator_GUI і GUI_BuilderPattern та його нащадки використовуються для забезпечення індикації певних даних, які обробляє концентратор для інформування користувача.

Клас GUI_Command_CommandPattern та його нащадки використовуються для налаштування користувачем концентратору.

На рис.3 класи GUI_BuilderPattern та GUI_Command_CommandPattern представлені схематично, оскільки існує велика кількість варіантів фізичної реалізації GUI та елементів керування, від якої буде залежати і програмна реалізація цих функцій.

Практична реалізація мультипоточності на прикладі програми для пристрою «Якість-Е1»

Для написання програми, яка приймає результати вимірювання пристрою «Якість-Е1» було обрано мову програмування Qt. Ця мова створювалася на базі мови C++ саме для створення програм з графічним інтерфейсом для користувача (GUI), що дозволяє використовувати синтаксис C++. Крім цього, у Qt є можливість використовувати потоки (Qthreads), які є платформонезалежними. Такий вибір дозволяє створити мультиплатформну (Windows / *nix) програму, яка через протокол USB буде приймати дані та обробляти їх. Крім того, передбачена можливість прийняття даних через інші інтерфейси зв'язку (COM-порт, Ethernet та ін.) та їх збереження у форматі XML.

Головним мотивом використання мультипоточного програмування в цій програмі є не пришвидшення розрахунків, а можливість якнайшвидше приймати дані, які передає прилад. У програмі можна виділити щонайменше дві асинхронні задачі: отримання даних від приладу та відображення (або збереження) їх на приладі, який є USB-хостом.

Слід зауважити, що і динамічна бібліотека, і частина програми вводу-виводу написані на C++ та фактично є головною її частиною. Щоб зробити цю програму мультплатформною, необхідно компілювати її як під Windows, так і під Linux, використовуючи для цього компілятор C++/C mingw.

Висновки

Використання запропонованої архітектури дозволить зменшити час на розробку програмного забезпечення для концентраторів на всіх рівнях ієрархії систем для збору і обробки даних про параметри комунальних послуг та зменшети час для розробки всієї системи.

Крім того, архітектура легко адаптується для певної апаратної модифікації, що призведе до уніфікації вже існуючого апаратного забезпечення.

Список літератури

1. М.П. Цапенко Измерительные информациионные системы: структуры и алгоритмы, системотехническое проектирование, 2-е изд., М., Энергоатомиздат, 1985 г, – 439 с.

2. http://www.metsys.hu/villamosenergia/files/Felhaszn%E1l%F3i%20k%E9zik%F6nyv (DGC-300%20Ver.%202.4).pdf

3. www.techrentals.com.au/uploads/LEM_UNIGOR355.pdf

4. http://www.techemenergy.ru/

5. http://www.samgas.com.ua/ua/4clients/products/montagniye-komplekti/rs-g16-g4.html

6. Сетевые информационно-измерительные системы. Департамент управления ресурсами фирмы Schlumberger, 2009

7. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2001.–368с

8. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы (4-е издание) – СПб.: Питер, 2010, – 958 с.

9. Александреску А. Современное проектирование на C++ (серия C++ in Depth). – М.: Издательский дом Вильямс, 2008. – 336с

10.http://dunkels.com/adam/pt/

11.http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372

12.Энтони Уильямс Параллельное программирование на C++ в действии. Практика разработки многопоточных программ – М.: ДМК Пресс, 2012, – 672 с.

13.Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. – СПб.: Питер, 2015, – 1120 с.

14.Бьерн Страуструп Язык программирования C++. Специальное издание – М.: Бином, 2011, –1155 с.

Стаття надійшла: 11.03.2016.

Відомості про авторів

Багацький Олексій Валентинович – к.т.н., науковий співробітник Інституту кібернетики ім..В.М.Глушкова НАН України, м. Київ, пр. Глушкова, 40, (044) 526-32-04.