НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"_____2015 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування
(код та назва спеціальності)

на тему: Керування потоками даних

Виконав : студент 4 курсу, групи ДА-11
(шифр групи)

Бабенко Володимир Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доц. Корначевський Я.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант основна частина к.т.н., доц. Харченко К.В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Консультант охорона праці доц. Гусєв А.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент д.т.н., проф. Бідюк П.І. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст.. викладач Британов О.А. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2015 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2015 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Бабенку Володимиру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Керування потоками даних
керівник проекту (роботи) Корначевський Ярослав Ілліч, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02»квітня 2015 р. № 30/1-ст

2. Строк подання студентом проекту (роботи) 08.06.2015

3. Вихідні дані до проекту (роботи)
Dataflow парадигма
Основні концепції FBP та SOA
Pythonect
LabVIEW
NoFlo

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

1. Дослідити dataflow та FBP з візуальною частиною з SOA.
2. Виділити їх основні переваги та недоліки у порівнянні з класичними методами.
3. Дослідити потоко-орієнтовані реалізації Pythonect.

4. Дослідити потоко-орієнтовані реалізації LabVIEW.
5. Дослідити потоко-орієнтовані реалізації VHDL.
6. Дослідити потоко-орієнтовані реалізації NoFlo.
7. Зробити загальні висновки по характеристикам, перевагам та недолікам.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

Висновки роботи у вигляді таблиці – плакат

Приклад парадигм керуванням потоками даних та інструкцій – плакат

Приклад реалізацій ч.1 – плакат

Приклад реалізацій ч.2 – плакат

Презентація роботи – презентація MS PP

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	Харченко К.В. доцент		
Охорона праці	Гусєв А.М. доцент		

7. Дата видачі завдання 01.02.2015

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2015	
2	Збір інформації	15.04.2015	
3	Дослідження dataflow та FBP	28.04.2015	
4	Дослідження застосування SOA	01.05.2015	
5	Робота з Python	15.05.2015	
6	Робота з LabVIEW	20.05.2015	
7	Робота з VHDL	25.05.2015	
8	Робота з NoFlo	30.05.2015	
9	Виведення основної характеристики	08.06.2015	
10	Отримання допуску до захисту та подача роботи в ДЕК	16.06.2015	

Студент _____ В.В.Бабенко
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Я.І.Корначевський
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

АНОТАЦІЯ

до бакалаврської дипломної роботи Бабенка Володимира Володимировича

на тему: «Керування потоками даних»

Дипломна робота присвячена дослідженню парадигми керування потоками даних, потоко-орієнтованого програмування, графічного представлення за допомогою сервіс-орієнтованої архітектури, вивчення особливостей та характеристика переваг та недоліків цих методів. Також було досліджено системи та засоби програмування з використанням парадигми керування потоками даних та виділено основні переваги та недоліки таких засобів. Були досліджені наступні с системи та засоби програмування: Pythonect, LabVIEW, VHDL, NoFlo.

Загальний об’єм роботи 72 сторінок, 16 рисунків, 4 таблиці, 14 посилань.

Ключові слова: парадигма керування потоками даних, потоко-орієнтоване програмування, сервіс-орієнтована архітектура, Pythonect, LabVIEW, VHDL, NoFlo.

АННОТАЦИЯ

к бакалаврской дипломной работе Бабенка Владимира Владимировича

на тему: «Управление потоками данных»

Дипломная работа посвящена исследованию парадигмы управления потоками данных, потоко-ориентированного программирования, графического представления с помощью сервис-ориентированной архитектуры, изучение особенностей и характеристика преимуществ и недостатков этих методов. Также било исследовано системы и средства программирования с использованием парадигмы управления потоками данных и выделено основные преимущества и недостатки таких средств. Было исследовано следующие системы: Pythonect, LabVIEW, VHDL, NoFlo.

Общий объём работы 72 страниц, 16 рисунков, 4 таблицы, 14 ссылок.

Ключевые слова: парадигма управления потоками данных, потоко-ориентированное программирование, сервис-ориентированная архитектура, Pythonect, LabVIEW, VHDL, NoFlo.

ABSTRACT

Of a bachelor’s degree work, which made by Babenko Volodumur Volodumurovich

On theme: “Managing of data streams”

This work is devoted to research of dataflow paradigm, flow-based programming, graphical representation by using service-oriented architecture, study of the features and characterize advantages and disadvantages of these methods. Also programming systems and tools with using of dataflow paradigm were researched, main advantages and disadvantages of these tools were identified. It was research of following systems and tools: Pythonect, LabVIEW, VHDL, NoFlo.

The total amount of work is 72 pages, 16 images, 4 tables, 14 references.

Keywords: dataflow paradigm, flow-based programming, service-oriented architecture, Pythonect, LabVIEW, VHDL, NoFlo.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SOA – Service Oriented Architecture

FBP – Flow-based Programming

ІП – Інформаційний пакет

IIP – Initial Information Packets

DSL – Domain Specific Language

VHDL – VHSIC (Very high-speed integrated circuits) Hardware Description Language

ВСТУП

Дві основні парадигми обчислень конкурують у світі комп'ютерних систем – системи обчислень з керуванням потоком інструкцій (instructions control flow) та системи обчислень з керуванням потоком даних (data flow). Кожна з них має свої переваги і широко застосовувалась протягом останніх десятиліть. Проте з розвитком різноманітних підходів до інформаційних систем та широким використанням SOA актуальним стає використання парадигми обчислень з керуванням потоками даних. Такий підхід органічно відображає взаємодію компонентів SOA та полегшує процес розробки архітектури складних інформаційних систем, дозволяє спеціалістам в предметній галузі брати більш широку участь у розробці програмних систем. На базі парадигми керування потоків даних розроблялись і використовуються потужні комплекси програмування

Системи програмування з використанням парадигми керування потоками даних набирають актуальність для сервіс-орієнтованої архітектури, а особливу увагу заслуговують методи графічного представлення для наочного відображення взаємодії компонентів складних інформаційних комплексів.

Мета цієї роботи – дослідити парадигму керування потоками даних та графічне представлення в SOA, а також проаналізувати системи та засоби програмування з використанням парадигми керування потоками.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- дослідити особливості парадигми керування потоками даних та графічне представлення в SOA;
- дослідити та проаналізувати системи та засоби програмування з використанням парадигми керування потоками даних;

1 ОСОБЛИВОСТІ ПАРАДИГМИ КЕРУВАННЯ ПОТОКАМИ ДАНИХ ТА ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ В SOA

1.1 Flow-based Programming

1.1.1 Історія виникнення

FBP було винайдено J. Paul Morrison на початку 1970-х років. Назва FBP зазнала ряд змін протягом років. На ранніх стадіях реалізації цієї технології на апаратах IBM(AMPS) її використовували в Банку Монреаля(1975р.), де була представлена онлайн банківська система. Одна з програм побудована з використанням AMPS використовується і в наш час. Більшість основних концепцій було описано в Technical Disclosure Bulletin у 1971, використовуючи досить загальні поняття. Стаття, що описує поняття методу та досвід використання його була опублікована в 1978 в IBM Research. IBM Systems Journal під назвою DSLM. Друга реалізація була зроблена в рамках спільного проекту IBM Canada і IBM Japan, під назвою “Data Flow Development Manager”(DFDM).

Загалом, поняття будо поширене в IBM як “Data Flow”, але цей термін вважався занадто загальним та згодом була прийнята назва “Data Flow Programming” і в 1994 році була випущена книга з такою назвою. У 2009 році кілька компаній презентували засоби, що базуються на принципі керування потоками даних, серед них: InforSense, Accelrys, Kettle and Knime.[2]

1.1.2 Введення та основні визначення

Flow-based programming (FBP) представляє собою, з одного боку, парадигму керування потоками даних з унікальним набором характеристик та, з іншого боку, вид компонентно-орієнтованого підходу програмної інженерії.

FBP описує додаток(програму) використовуючи метафору “фабрика даних”. Воно розглядає додаток не як єдиний, послідовний процес, який починається в даний момент часу, а потім виконує одну дію за один раз поки він не закінчиться, а як мережа асинхронних процесів, що зв’язуються за допомогою потоків структурованих сегментів даних, що називаються «Інформаційні пакети»(ІП). З цієї точки зору акцент робиться на даних програми і перетворення, що застосовуються до них для отримання бажаних результатів. Ззовні процесів мережа визначається як список зв’язків, які інтерпретуються частиною програмного забезпечення, що називається. «scheduler».

Процеси сполучаються за допомогою з'єднань з фіксованою потужністю. З'єднання прикріплене до процесу за допомогою порту, що має назву погоджену між кодом процесу та визначенням мережі. Більш ніж один процес може бути виконаний однією й тією ж частиною коду. У будь-який момент часу, даний ІП може належати одному процесу, або передаватися між двома процесами. Порти можуть бути як простими, так і складними (поєднання портів з асинхронними процесами, що дозволяє підтримувати багато тривалих та примітивних функцій обробки даних, наприклад, сортування, злиття, додавання тощо, у вигляді «чорних скриньок».

Процес є екземпляром компоненти, що працює паралельно з іншими процесами, у тому числі потенційно з іншими екземплярами цієї компоненти. Зазвичай цей метод зображується у вигляді орієнтованого графу, що складається з процесів(як вузлів) і з’єднань(як ребра). Для розробників, компоненти зазвичай будуть у вигляді «чорних скриньок». У вигляді «білих скриньок» буде в тому разі, коли потрібно написати нову компоненту, або якщо ця частина реалізована

як підграф(у такому випадку користувач може рухатися між різними рівнями графу)

Через те, що FBP процеси можуть тривати так довго, доки вони мають дані для роботи над ними та місце для виведення результатів, FBP додатки працюють менше часу, ніж звичайні програми і роблять оптимальне використання процесорів на машині без спеціального програмування, що потрібно для досягнення цього.

Визначення мережі, як правило, схематичне і конвертується в список з'єднань на деякій мові нижчого рівня або індексацією. Таким чином, на цьому рівні FBP представляє собою візуальну мову програмування. Більш складні описи мережі мають ієрархічну структуру, будучи побудовані з підмереж із «sticky» з'єднаннями.

FBP має багато спільного з мовою Linda в тому, що це, за термінологією Gelernter та Carriero, є «координаційна мова», це, по суті, незалежність від мови. Дійсно, враховуючи, що scheduler написаний на мові досить низького рівня, компоненти, написані на різних мовах, можуть бути поєднані в єдину мережу. Таким чином, FBP подає себе в концепції предметно-орієнтованої мови програмування або «міні-мови».

FBP експонує «Зв'язність даних» (міра в якій модуль програми залежить від кожного іншого модуля) як вільний тип зв'язку між компонентами. Ця концепція показує, що FBP задовольняє ряд критеріїв для сервіс-орієнтованої архітектури, хоча й у більш детальному рівні, ніж у більшості прикладів цієї архітектури.

Отже, загальний підхід FBP це осмислити програму у вигляді ряду(серії) потоків та субпотоків, які течуть(flow) через ряд поєднаних процесів. Паралельність та узгодженість можливі завдяки обмеженню спілкування між процесам для використовування потоків ІП. Проблема розділення пам'яті уникається завдяки вищеописаному правилу, що один ІП може належати одному процесу в даний момент часу.

Візуальне програмування в цьому контексті проявляється в поєднанні текстових компонентів та(або) графів у двомірне представлення, яке використовує як перевагу візуального стилю мислення людини.

FBP сприяє високорівневому, функціональному стилю специфікації, що спрощує розуміння поведінки системи і, як результат, маємо спрощене повторне використання коду, тестування та можливість підтримки.[2]

1.1.3 Класифікація FBP

FBP орієнтовано на абстракцію асинхронних машин, що працюють разом і існує багато способів імітації такої поведінки. Потоки, петля або мережа комп'ютерів та інші варіанти можуть впливати на реалізацію, але граф та дизайн компонентів завжди будуть однакові при виконанні концепції FBP.

- **Classical FBP.** Це основний підхід, де операційна система або віртуальна машина використовує потоки, процеси зчитують та записують одночасно, вибираючи порти для зчитування, запису або відключення. Також реалізовані більшість додаткових функцій та можливостей. Наприклад, JavaFBP.
- **Reactive FBP.** Цей підхід зв'язаний з подіями. Процеси працюють, очікуючи на такі події як підключення, відключення, відправка тощо, та контролюють тільки відправку та відключення. Вони також можуть інші механізми для використання ручного зчитування\прийому, але загальний дизайн будується навколо «слухачів» подій. Наприклад, NoFlo.
- **"Litmus test".** Для того, щоб розрізнити вид FBP, з яким ви маєте справу, чи присутнє вибіркове зчитування. Чи можете виразити вимоги до даних для роботи в якості послідовного зчитування, чи маєте кодувати стан машини яка залежить від подій?

Далі проілюструємо приклад компонента, який зчитує два числа з двох портів і посилає більше через порт GT і менше через LT.

Classical (in pseudo java):

```
... component code and setup
```

```
Packet a = read("A");
Packeg b = read("B");
if( a.data() > b.data() ){
send("GT", a);
send("LT", b);
} else {
send("GT", b);
send("LT", a);
}
```

Reactive (in javascript)

```
... component code and setup
inPorts.a.on('data', function(data, index){
  this.a = data;
  compare();
}
inPorts.b.on('data', function(data, index){
  this.b = data;
  compare();
}
function compare(){
 if( this.a !== null && this.b !== null ){
  if( this.a > this.b){
    outPorts.gt.send(a);
    outPorts.lt.send(b);
  }else{
    outPorts.gt.send(b);
    outPorts.lt.send(a);
  }
```

```
    this.a = null;
    this.b = null;
  }
```

1.1.4 Характеристики та концепція FBP

На приведеній нижче діаграмі показані основні елементи FBP (окрім інформаційних пакетів (IP)). Така діаграма може буди безпосередньо перетворена на список з'єднань, які, у свою чергу, можуть бути виконані деяким програмним або апаратним комплексом.

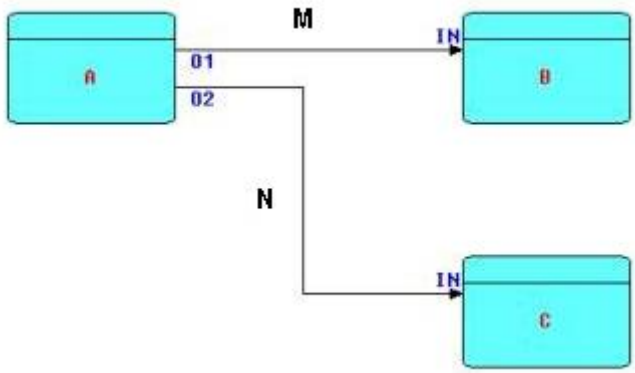

Рис.1 - Приклад FBP мережі[2]

A, B і C є процесами виконання коду компонентів. O1, O2 та два IN порти призначені для з'єднань M та N. Можлива ситуація коли процеси B і C будуть виконувати одну і ту саму ділянку коду, тому кожен з процесів повинен мати свою власну ділянку оперативної пам'яті, власні блоки управління тощо. Але у будь-якому з випадків (код B і C однаковий або ні) ці блоки можуть використовувати вхідні порти з однаковими іменами. M і N інколи називають "обмеженими буферами" оскільки у кожний момент часу вони можуть утримувати заздалегідь задану кількість інформаційних пакетів. Концепція портів дозволяє використовувати будь-який компонент у одній і тій самій мережі

кілька разів. У поєднанні з можливістю параметризації за допомогою “ініціалізуючих(або початкових) інформаційних пакетів” (Initial Informational Packets — IIP) це робить FBP одним з різновидів компонентно-орієнтованої архітектури. Інформаційні пакети визначаються у деякому просторі, що називається “простором інформаційних пакетів” і мають чітко визначений час існування. Після використання інформаційні пакет повинен бути знищеним. Процедура знищення є явною і обов'язок її виконання покладається на процес, що прийняв пакет. Пакети переміщуються за різними зв'язками мережі (конкретніше, саме зв'язки займаються переміщенням пакетів) цілком асинхронно, починаючи свій шлях у момент генерації. Така поведінка реалізує концепцію “ледачих обчислень”. Тобто процес починає свою роботу, як тільки у нього з'являються усі необхідні для обчислень дані. Як правило, інформаційні пакети це структуровані блоки даних, але пакет може і не містити будь-якої інформації і використовуватись просто як сигнал до певної дії. Прикладом такого пакету може бути “дужковий” (“bracket IP”) пакет, тобто пакет, що вказує на початок деякої послідовності (підпослідовності). В такого пакету повинен бути відповідний пакет, що буде закривати послідовність. Можливі вкладені одна в одну послідовності. Такі послідовності можуть переміщуватись системою як один об'єкт. Система зв'язків і процесів, описаних вище, легко масштабується до будь-якого розміру. Під час розробки між реальними блоками можуть бути поміщені блоки моніторингу, для цілей моніторингу і відлагодження. Також можна використовувати блоки-заглушки, що згодом будуть змінені на реальні блоки. Всі ці можливості роблять FBP зручним інструментом для створення прототипів. Насправді FBP є метафорою промислової лінії збірки (конвеєра): інформаційні пакети переміщуються від процесу до процесу подібно до деталей, що подорожують від верстата до верстата. "Машини" (блоки) можуть бути легко підключені, зняті на ремонт, заміну тощо. Досить дивно, але це дуже схоже на обробку інформації до того, як з'явилися комп'ютери.

FBP часто плутають з Dataflow програмуванням, але насправді це розділ Dataflow програмуванням з наступними унікальними властивостями:

- Структурні: графи структуровані; компоненти теж мають структуру(інтерфейс, стан, поведінку).
- Проектування системи розподілене на 2 шари: графічний(як правило, візуальний) та компонентний(зазвичай, текстовий). З точки зору архітектури програмного забезпечення рекомендуються окремі ролі, такі як «графічний дизайнер» та «реалізатор компонент».
- Паралельність: кожен FBP процес у своєму власному потоці.
- Інформаційні пакети мають свій життєвий цикл і явно належать тільки одному процесу одночасно.
- Процес може бути станом.
- Компоненти можуть мати декілька входів або виходів.
- Порти можуть бути простими та складними, де окремі елементи розглядаються як окремі порти, але адресація відбувається за допомогою імені порту та індексу.
- Додаток являє собою граф, а не дерево. Циклічні сполуки(петлі зворотного зв’язку) не допускаються.
- З’єднання можуть бути поєднані в граф, за умови, що пакети з різних ребер прибувають на вхідний порт у порядку FIFO(First In, First Out).
- З’єднання повинні бути явно розділені за допомогою компоненти по причині різних стратегій розділення і явного правила власності ІП
- З’єднання, що реалізовані у вигляді обмежених буферів з FIFO порядком та ємністю від 0 до «А» є обмеженими в реалізації.
- Дані можуть мати дані. Пакет може мати вкладені пакети і компоненти не повинні бути попереджені про прийняті дані. За цим принципом можуть бути створені деревовидні структури та словники.
- Параметри надаються екземпляру компоненти в мережі як сукупність значення даних із певним вхідним портом. Ця концепція називається «Ініціалізація ІП» і її реалізація змінюється між версіями FBP.

Далі більш детально розглянемо основні принципи та концепції, які дадуть нам поняття про роботу FBP у цілому.

Компоненти. Компоненти – будівельні блоки в FBP. Це поняття охоплює елементарні компоненти, які зазвичай представлені класами, функціями або невеликими програмами

- **Процес** – екземпляр компонента, що знаходиться в графі Система повинна обробляти процеси у вигляді співпрограми, потоків або аналогічній формі багатозадачності(паралелізму) Вони повинні мати доступ тільки до свого внутрішнього стану та портів, але не більше, наприклад, про граф та інші процеси.

- **Зв'язки.** Процес комунікації через сполучення, доступ до яких процес отримує за допомогою портів. Зв’язки зазвичай реалізуються через обмежений буфер або черги FIFO. Розмір буфера або максимальне можливе число пакетів у черзі визначається як ємність з’єднання. Деякі реалізації FBP дозволяють реалізовувати зв'язок з ємністю 0, що означає, що дані ІП одразу передається між процесом-отримувачем та процесом-відправником.

- **Порти** – точки взаємодії між процесами та з’єднаннями. Вони мають назву та можуть бути проіндексовані у випадку з складним портом(Array Port). Процес може відправляти та отримувати дані з будь-якого порту. Досить зручно, що є можливість «бачити» чи з’єднання вхідний порту має пакети або з’єднання вихідного порту може заповнитись. Прецедентом для цього може бути баланс завантаженості або уникнути складного пересилання та маршрутизації пакетів, щоб створити реактивну складову. Прикладом цього може бути процес, який конвертує валюту та переглядає новий курс перед кожною ітерацією.

- **Вхідні порти** забезпечують зчитування або отримання повідомлень з черги і вимагають індекс у випадку складних портів. Інші функції, такі як визначення завантаженості буфера або на які індекси є надходження дуже спрощують реалізацію компонента. У «класичному» FBP, зв'язок поєднує один і більше вихідних портів з одним вхідним, враховуючи кожен індекс складного порту як простий порт. Коли з’єднання стає порожнім, то процес призупиняє

підтримку цього зв’язку до тих пір, поки пакет не прибуває. Це справедливо для «класичного» FBP, у той час як інші реалізації чекають пакети, щоб активувати подію або мають колекцію буферів, до яких процес може отримати доступ. У «класичному» FBP, коли більш ніж один вихідний порт підключений до вхідних портів, пакети прибувають в порт введення за принципом "first come, first served "

- **Вихідні порти** забезпечують функцію відправки пакету у чергу порту, що підключений до процесу. Зазвичай, для відправки потрібен пакет та ім’я порту, або посилання це метод, а вихідний порт є відомим об’єктом. Якщо зв'язок переповнений, то процес призупиняється, доки буфер не звільниться.
- **Внутрішній стан**. Процес, будучи екземпляром компонента, може утримувати свій внутрішній стан поки він активний. Важливою частиною FBP є утримування стану приватним і взаємодіяти з іншими процесами тільки через дій на портах.
- **Обробка даних** орієнтована на обробку потоків пакетів то вбудованих підпотоків. Проекти повинні бути орієнтовані на перетворення та фільтрацію даних. Поки дані між процесами знаходяться в буфері, асинхронність операції досягається, тим самим звільняючи розробника від обробки додаткової логіки.

Інформаційні пакети. Щодо визначення цього поняття ведуться досить багато дискусій, але врешті-решт воно залежить від області застосування. Загальна думка полягає в тому, що пакет повинен нести інформацію, яка впорядкована і пасивна по своїй суті. Відправка екземпляру відеопрогравача в пакеті є прикладом того, що не повинно робитись, у той час, коли відправка окремих кадрів буде правильним підходом. Пакет повинен містити багаторазовий тип даних і, у залежності від реалізації, бути доступним для функцій таких як додавання словника входів з інших пакетів, приєднання пакетів як рідної форми деревовидної структури, право на зупинку процесу від зміни пакету, схема валідації пакетних даних тощо.

- **Початкові ІП.** У графі кожен процес може мати декілька початкових інформаційних пакетів, які виконують функцію конфігурації. Вони не подаються до порту доки процес не визначає отримання або зчитування на цьому порті. Про них можно думати як про пасивні пакети.
- **Активація.** Процес активується коли є надходження пакетів на вхідні порти, винятком цього буде початковий компонент в деяких реалізаціях, що призначений для початку активації графу та подальше прикріплення пакетів. ІП не повинні активовувати граф, хоча це зроблено в деяких реалізаціях.

Зв'язки(з'єднання). З'єднання обмежують буфери між портами та їх розмір, виражений числом пакетів. Коли вони заповнені, то процес процес-відправник виконує блок «відправити», коли ж вони порожні, то приймаючий процес виконує блок «зчитування».

- **Об'єднання.** Коли два чи більше портів повинні підключитись до простого одинарного порту, у дію вступають деякі форми об'єднання. Або у вигляді компонента об'єднання з декількома входами та одним виходом, через який упорядковано виходять дані, або прибуття в порядку читання складного порту. Автоматичне об'єднання також може забезпечуватися бібліотекою.
- **Розщеплення.** Коли один вихідний порт має підключатись до декількох вхідних портів, то пакет повинен роз'єднуватися наче у водному потоці, саме для досягнення цієї мети потрібен компонент, що виконує розщеплення. Він повинен робити копії пакету та розсилати на кожний підключений вихід. Виконання цих дій може забезпечуватися і без явного «Split»-компонента, хоча Морісон навпаки заохочує його використання

Графи. FBP використовує орієнтовані графи для відображення структури програми. Процеси представлені вузлами графа, а з'єднання між портами – ребрами. Виходи можуть бути з'єднані тільки з входами. Граф будується як статичне представлення програми, що виконується за допомогою бібліотек. Деякі засоби розроблюються для роботи з графами і вони можуть бути побудовані за допомогою як візуальних засобів розробки таких як DrawFBP або

NoFlo, так і текстових DSL. Граф може бути вбудованим у інший граф як процес, якщо він розкриває свої зовнішні порти для підключення до внутрішніх процесів іншого.

- **Зовнішні порти.** При створенні графа, що може бути вбудований в інший граф, дизайнер графів відкриє зовнішні порти, які можуть стати портами вводу чи виводу для забезпечення підключень до внутрішніх процесів дизайнеру графів. При цьому заохочується створення нових компонентів із уже існуючих.
- **Підпрограми.** Ми можемо розглянути випадок, коли процес надсилає дані та одразу ж розпочинає зчитування з порту, як еквівалент виклику підпрограми. В ідеалі, «відправник» та «адресат» не матимуть спільного стану через надсилання інформаційного пакету. Таким чином, граф може виродитись у синхронну програму.
- **Петлі зворотного зв'язку.** FBP дозволяє топології петель у мережах. Це може бути корисно в ситуаціях, коли результати будь-якого процесу виводяться, щоб стати вхідними даними у подальшому, наприклад, при використанні інтерактивних додатків, а також для додатків, таких як бланк обробки матеріалів, де компоненти послідовно розбиваються на підкомпоненти до тих пір, поки це можливо, а тоді результати розбиття вилучаються із петлі. Коли процес використовується у т. зв. режимі підпрограми, також може бути використана петля. В такому випадку «відправник» виконує прийом/передачу, а «адресат», відповідно, передачу/прийом. Це може бути корисним, оскільки не перешкоджає використанню «відправника» у потоковому режимі в інших додатках, або ж в іншому місці цього самого додатку. Нажаль, ця топологія призводить до додаткової складності: за критерієм останови FBP, процес має зупинятися при закритті всіх процесів, що передають йому дані. Але, у петлевій топології для будь-якого процесу всі інші є такими, а отже певний процес має проконтролювати відключення мережі. Стандартна техніка для цього – закрити всі порти вводу та завершити свою роботу, що призведе до зупинки всіх інших процесів у петлі.

У FBP в межах одного процесу не дозволяється підключення порту виводу до порту вводу, оскільки це призводить до зациклення і помилки.

- **Час виконання процесу.** Процес, що виконується протягом довгого часу, пропускає дані через свої вхідні порти кілька разів впродовж свого виконання, що може призводити до зупинки. З іншого боку, процеси, що виконуються протягом короткого часу, використовуються як підпрограми і деактивуються одразу ж після свого завершення. Дуже важливо розрізняти, компонент якого роду створюється або реалізується, щоб уникати використання надто складних, або ресурсоємних процесів. Наприклад, у залежності від роду, компонент, що здійснює перевірку наявності нових повідомлень у поштовому обліковому записі, може або постійно підтримувати активне з'єднання та перевіряти пошту щохвилини, або ж встановлювати з'єднання, виконувати перевірку та одразу ж розривати з'єднання при виклику.[3]

1.1.5 Додаткові можливості FBP

Додаткові можливості допомагають спрощувати часто виконувані задачі, або розв'язувати певні специфічні проблеми. Такі можливості не є обов'язковими у кожній реалізації FBP саме через те, що вони є необхідними в невеликій кількості ситуацій, або через те, що їх можна замінити за допомогою інших засобів.

Можливості, пов'язані з даними:

- **Підпотоки.** Основний механізм контролю потоків працює із з'єднаннями та роз'єднаннями, але пакети можуть бути згруповаными в потоці для виконання різноманітних завдань. Наприклад, у відповідь на запит на групування відбувається передача списків пакетів, побудова деревовидних структур у потоці, тощо. Група виділяється відкриваючим та закриваючим інформаційними пакетами, що називаються керівними. Це звичайні інформаційні пакети, але вони позначаються дужками. Наприклад, якщо компонент робить запити на пересилання йому URL, а потім надсилає відповіді

на ці запити, ці відповіді можна згрупувати таким чином: відкриваючий пакет, який містить URL, вміст відповідей та закриваючий пакет із URL. Для того, щоб відмітити це, використовуються символи > та <. За необхідності групи можна робити вкладеними.

Можливості, пов'язані з портами:

- **Складні порти.** Це порти, що мають індекси, причому, кожен індекс може бути поставлений у відповідність певному порту. Користувач також може зчитувати й записувати незалежні індекси. Для цього використовуються ті ж правила, що й для нормальних портів при блокуванні.
- **Автоматичні порти.** Не для всіх портів компоненти, до яких вони підключені, мають бути відомими: іноді бажано обумовити з'єднання, не відомі для портів у мережі. Це допомагає впроваджувати часові обмеження в додаток без необхідності додавати логіку до включених компонентів або справлятися із різними помилковими умовами. Один із типів таких портів називають автоматичним через те, що компоненти не контролюють їх роботу. Існують два типи автоматичних портів: автоматичні порти вводу та автоматичні порти виводу. Перший з них відіграє роль затримки: якщо такий порт підключено, активація процесу затримується до тих пір, поки на цей порт не надійде ІП, або поки цей порт не буде закрито. Інший порт при зупинці процесу надсилає сигнал, що може бути використаним для того, щоб запустити певні дії далі за потоком.

Можливості, пов'язані із потоками:

- **Заблоковані відправлення та зворотній тиск.** У випадку, коли з'єднання працює на повну потужність, послідовні відправлення блокуються доти, доки не з'явиться місце для нових пакетів, гальмуючи таким чином виконання процесу відправлення. Це не дозволяє процесу поглинути всі системні ресурси, що призвело б до повної зупинки. Таким чином, процеси, що виконуються вверх за потоком, регулюють кількість даних, які обробляють процеси, що діють вниз за потоком.

- **Примусова відміна блокування відправлень.** Для операцій, які потребують безперервного надходження потоку нових даних, наприклад, зчитування показників датчиків, видалення старих пакетів або уникнення блокування процесів, що діють вниз за потоком, примусова відміна блокування є дуже корисною.
- **Виявлення перешкод.** Деякі додатки містять вбудовані можливості для виявлення та запобігання перешкод, які одні процеси можуть викликати для виконання інших.
- **Рекурсивні графи.** Рекурсивні графи можуть посилатися та включати в себе один одного. Це дозволяє візуалізувати рекурсивну обробку даних.

Динамічна структура. Додатки FBP є статичними: від моменту старту додатку графу додавання або видалення вузлів неможливе. Деякі реалізації дозволяють змінювати структуру додатку під час його виконання, роблячи його з одного боку більш гнучким, а з іншого, менш надійним.[3]

1.2 Сервісно-орієнтована архітектура та зв'язок з FBP

Сервісно-орієнтована архітектура (SOA) — архітектурний шаблон програмного забезпечення, модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабко пов'язаних замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколам.

1.2.1 Основні поняття

SOA — це не технологія, а спосіб проектування і організації інформаційної архітектури та бізнес-функціональності.

SOA: це парадигма, призначена для проектування, розробки та управління дискретних одиниць логіки (сервісів) в обчислювальному середовищі. Застосування цього підходу вимагає від розробників проектування застосунків як набору сервісів, навіть якщо переваги такого рішення відразу неочевидні. Розробники повинні вийти за межі своїх застосунків і подумати, як скористатися вже наявними сервісами, або вивчити, як їх сервіси можуть бути використані їх колегами.

Архітектура не прив'язана до якоїсь певної технології. Вона може бути реалізована з використанням широкого спектру технологій, включаючи такі технології як REST, RPC, DCOM, CORBA або веб-сервіси. SOA може бути реалізована, використовуючи один з цих протоколів і, наприклад, може використовувати додатково механізм файлової системи для обміну даними.

Головне, що відрізняє SOA, це використання незалежних сервісів з чітко визначеними інтерфейсами, які для виконання своїх завдань можуть бути викликані якимось стандартним способом, за умови, що сервіси заздалегідь нічого не знають про додатку, який їх викличе, а додаток не знає, яким чином сервіси виконують своє завдання.

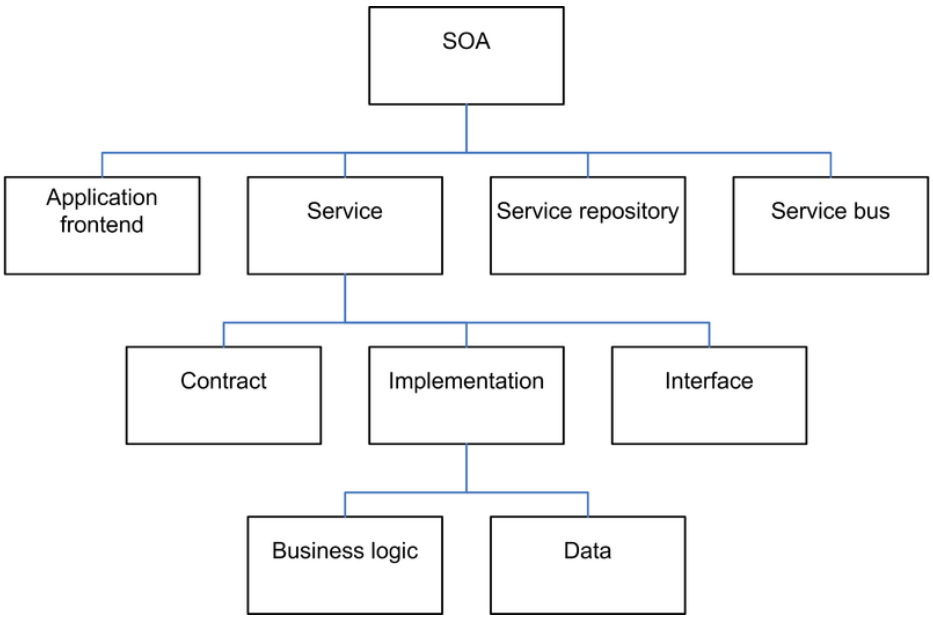

Рис.2 - Елементи сервіс-орієнтованої архітектури[4]

SOA також може розглядатися як стиль архітектури інформаційних систем, який дозволяє створювати додатки, побудовані шляхом комбінації слабо-зв'язаних і взаємодіючих сервісів. Ці сервіси взаємодіють на основі якого-небудь строго певного платформо-незалежного та мовно-незалежної інтерфейсу (наприклад, WSDL). Визначення інтерфейсу приховує мовно-залежну реалізацію сервісу.

Таким чином, системи, засновані на SOA, можуть бути незалежні від технологій розробки і платформ (таких як Java, . NET і т. д.). Наприклад, сервіси, написані на C #, що працюють на платформах. Net і сервіси на Java, що працюють на платформах Java EE, можуть бути з однаковим успіхом викликані загальним складовим додатком. Додатки, що працюють на одних платформах, можуть викликати сервіси, що працюють на інших платформах, що полегшує повторне використання компонентів.

SOA підштовхує до використання альтернативних технологій і підходів (таких як обмін повідомленнями) для побудови додатків за допомогою зв'язування сервісів, а не за допомогою написання нового програмного коду. У цьому випадку, при належному проектуванні, застосування повідомлень дозволяє компаніям своєчасно реагувати на зміну ринкових умов — настроювати процес обміну повідомленнями, а не розробляти нові програми.

Ще до недавніх пір термін «сервіс-орієнтована архітектура» був синонімом «Web-сервіс». SOA — виклик Web-сервісів за допомогою засобів і мов управління бізнес-процесами. SOA — це термін, який з'явився для опису виконуваних компонентів — таких як Web-сервіси — які можуть викликатися іншими програмами, які виступають у якості клієнтів або споживачів цих сервісів. Ці сервіси можуть бути повністю сучасними — або навіть застарілими — прикладними програмами, які можна активізувати як чорний ящик. Від розробника не потрібно знати, як працює програма, необхідно лише розуміти, які вхідні та вихідні дані потрібні, і як викликати ці програми для виконання. У найзагальнішому вигляді SOA припускає наявність трьох основних учасників: постачальника сервісу, споживача сервісу та реєстру сервісів. Взаємодія

учасників виглядає досить просто: постачальник сервісу реєструє свої сервіси в реєстрі, а споживач звертається до реєстру із запитом.

Для використання сервісу необхідно дотримуватися угоди про інтерфейс для звернення до сервісу — інтерфейс повинен не залежати від платформи. SOA реалізує масштабованість сервісів — можливість додавання сервісів, а також їх модернізацію. Постачальник сервісу і його споживач виявляються непов'язаними — вони спілкуються за допомогою повідомлень. Оскільки інтерфейс повинен не залежати від платформи, то і технологія, використовувана для визначення повідомлень, також повинна не залежати від платформи. Тому, як правило, повідомлення є XML-документами, які відповідають XML-схемі.

Дійсно, відкриті стандарти, що описують XML і Web-сервіси, дозволяють застосовувати SOA до всіх технологій і додатків, встановлених в компанії. Як відомо, Web-сервіси базуються на широко поширених і відкритих протоколах: HTTP, XML, UDDI, WSDL і SOAP. Саме ці стандарти реалізують основні вимоги SOA — по-перше, сервіс повинен піддаватися динамічному виявленню і викликом (UDDI, WSDL і SOAP), по-друге, повинен використовуватися незалежний від платформи інтерфейс (XML). Нарешті, HTTP забезпечує функціональну сумісність.[4]

1.2.2 Переваги використання SOA

Перш ніж, перерахувати переваги використання SOA, буде доречним нагадати, що переваги бувають різні: стратегічні і тактичні. SOA має ряд переваг як стратегічних, так і тактичних.

Стратегічна цінність SOA:

- Скорочення часу реалізації проектів, або «часу виходу на ринок».
- Підвищення продуктивності.
- Швидша і дешевша інтеграція застосунків і інтеграція B2B(Business to business).

Тактичні переваги SOA:

- Простіша розробка і впровадження додатків.
- Використання поточних інвестицій.
- Зменшення ризику, пов'язаного з впровадженням проектів в області автоматизації послуг і процесів.
- Можливість безперервного поліпшення наданої послуги.
- Скорочення числа звернень за технічною підтримкою.
- Підвищення показника повернення інвестицій (ROI).
- Перспективи

1.3 Приклад використання SOA та FBP

Для пояснення роботи SOA та FBP з розглянемо наступний приклад – алгоритм вирішення квадратного рівняння у вигляді компонента SOA[1]

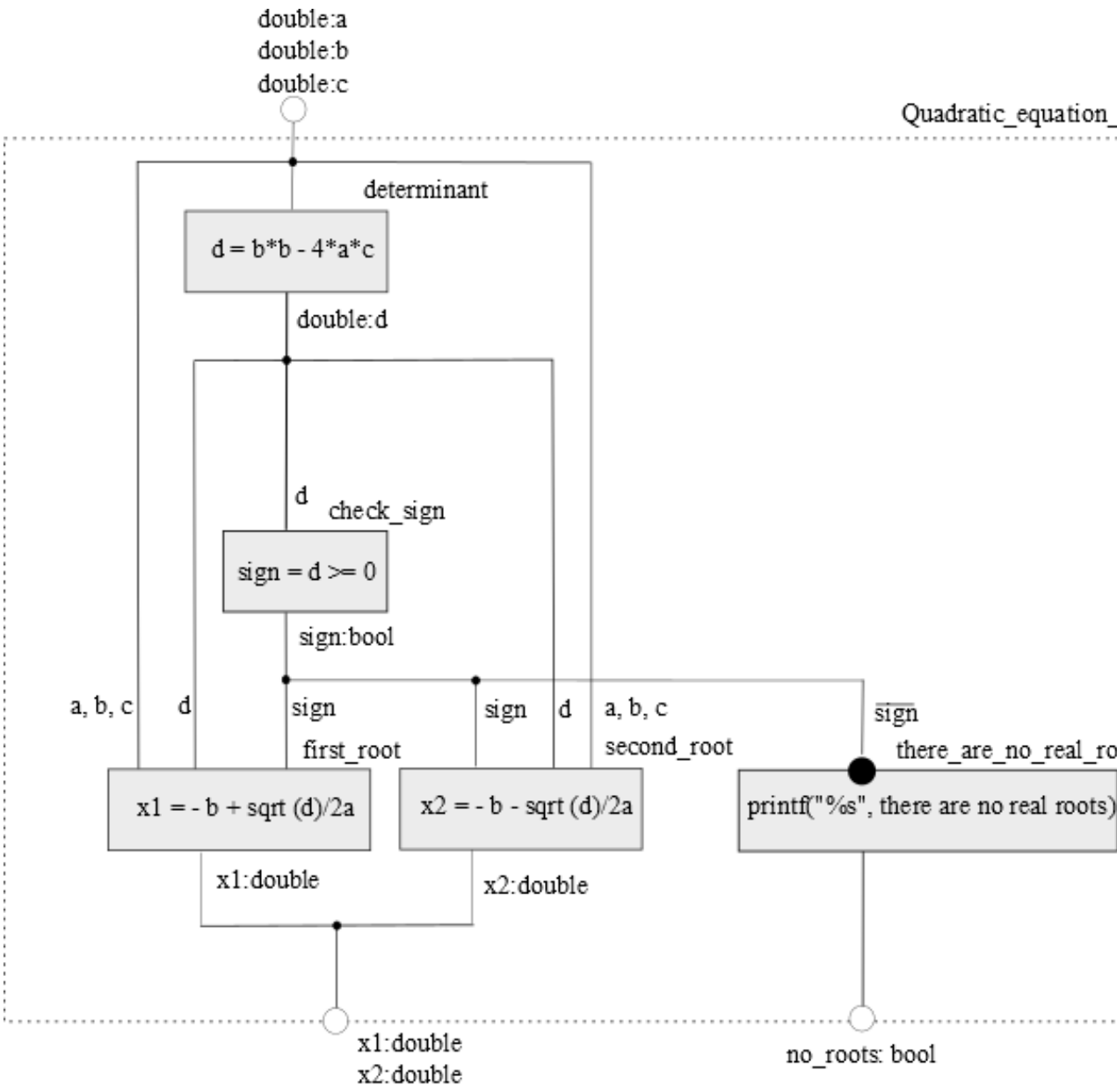

Рис.3. – Діаграма реалізації компоненти SOA для рішення квадратного рівняння[1]

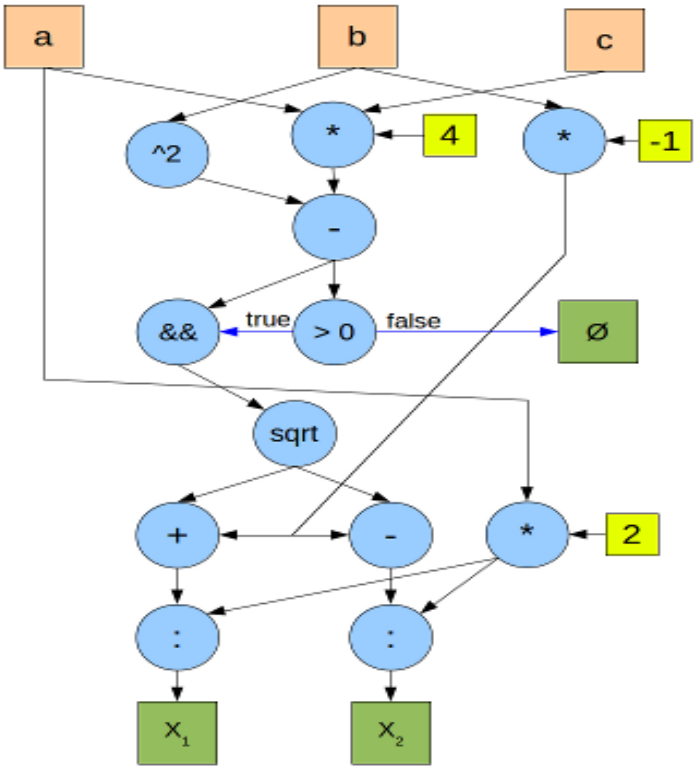

Рис.4 – Відповідний граф, що ілюструє роботу FBP[3]

Сині кола — оператори(процеси), помаранчеві квадрати — вхідні дані, зелені — вихідні, жовті — константи. Чорні стрілки(зв’язки) означають передачу чисельних даних, сині — булевих.

Як готовий компонент в інших діаграмах потоків даних можливо використовувати зображення компонента вирішення квадратного рівняння у вигляді прямокутника з відповідними входами та виходами (рис. 5).

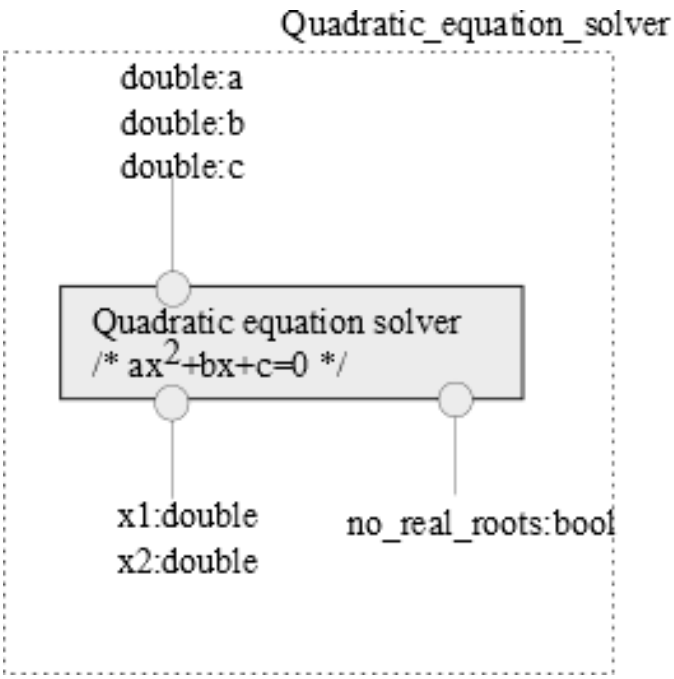

Рис.5. – Відповідне зображення діаграми використання компоненти SOA для рішення квадратного рівняння[1]

1.4 Висновки

Переваги використання запропонованого методу зображення діаграм потоків даних для розробки інформаційних систем, в тому числі з використанням SOA, наступні:

- Мережі потоків даних дуже вдалий вибір для представлення та моделювання процесу
- Передача повідомлень позбавляється проблем, що пов’язані з розділенням пам’яті
- Узгодженість та паралельність в основі. Код може бути розподілений між ядрами(процесами) та навіть між мережами.
- Програми з використанням FBP значно гнучкіші в питанні розширення та «надбудови». Елементи можна легко скомпонувати в складені елементи та композиції «знизу-догори».
- Поєднання процесу створення архітектури інформаційної системи та процесу її реалізації у вигляді програмного коду окремих компонентів

- Ієрархічна побудова архітектури інформаційних систем, можливість вкладеного опису діаграми з керуванням потоками даних
- Можливість використання процесу розробки, що керується тестами (Test Driven Development)
- Просте графічне зображення потоків даних, яке дозволяє відображати інтуїтивно зрозумілу логіку роботи системи, можливість простого відображення у векторних графічних редакторах загального призначення (Visio, Dia, Inkscape і т.д.)
- Швидкість зображення діаграм потоків даних на ескізах за допомогою олівця без спеціальних навиків креслення
- Можливість використанні високопотужних систем і парадигм обчислювань, таких як map-reduce, за допомогою паралельного виконання компонентів.

З недоліків можна виділити:

- Для більшості програмістів, особливо професійних, спосіб мислення через парадигму керування потоками даних досить незвичне та, навіть, незнайоме. Більшість DF засобів та мов програмування складають окрему нішу з порівняно невеликою кількістю зацікавлених.
- Обмеженість схемою FIFO.(LIFO не використовується)
- Відсутність використання розподіленої пам’яті має свою ціну. Повідомлення або мають мати копії, або бути незмінними.
- Використання FBP потребує дотримання цього методу з самого початку. Тобто, перетворення традиційних програм в DF дуже складне через різницю в структурі.

Такі методи розробки досить перспективно використовувати в наступних галузях: інформаційні системи корпоративного рівня з використанням сервіс-орієнтованої архітектури, системи цифрової обробки сигналів, системи обробки зображень, системи керування, аналіз великих даних, робототехніка.

2. СИСТЕМИ ТА ЗАСОБИ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ ПАРАДИГМИ КЕРУВАННЯ ПОТОКАМИ ДАНИХ

2.1 Pythonect

Pythonect це нова експериментальна мова програмування загального призначеня, що має в основі на парадигму керування потоками даних та базується на мові Python. Він підтримує як візуальне програмування, так і текстову скриптову мову. Текстова частина прагне об’єднати швидку і інтуїтивно зрозумілу оболонку сценаріїв з засобами та можливостями Python. Візуальна частина ж базується на основі ідеї діаграми, що складається з наступних складових частин: процесів-скриньок та зв’язків-стрілок.

Інтерпретатор Pythonect безкоштовний з відкритим кодом, що повністю написаний на Python, і доступний згідно з ліцензією BSD 3-пунктом.

Pythonect, будучи мовою dataflow програмування, представляє дані як потік, що прибуває з джерела, проходить через ряд процесів та надходить до кінцевого пункту призначення. І тому є найбільш підходящим варіантом реалізацій додатків, що спрямовані на потоки даних. Найбільш популярним прикладом потоко-орієнтованих додатків є обробка сигналів у реальному часі, наприклад, відеопроцесор, що отримує сигнал на вхід, модифікує його та подає на вихід(дисплей). Як і відео, інші задачі з цієї області (наприклад, обробка зображень, аналіз даних, швидке прототипування тощо) можуть бути виражені як різні компоненти, що з’єднанні багатьма каналами. Паралелізм та можливість масштабувати – одні з переваг використання таких засобів. Різні компоненти в мережі можуть «маневрувати», щоб утворювати унікальні потоки даних без обов’язкового прив’язування та наслідування. Окрім того, дизайн та концепція Pythonect полегшує роботу з розподіленими системами та паралельними процесами.

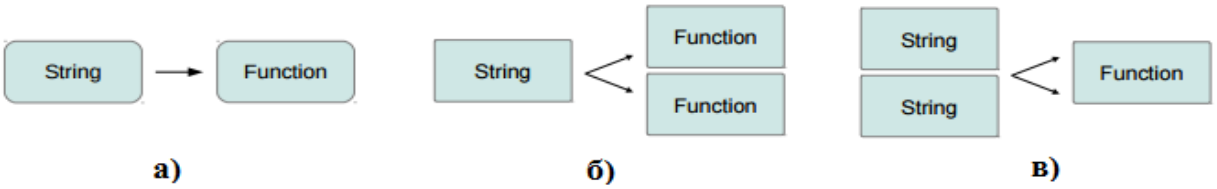

Рис.6 - а) один відправник-один одержувач; б) один відправник-декілька одержувачів; в) декілька відправників-один одержувач

На рис.3 зображені схеми основних функцій:

а) ‘Text’ -> print

б) ‘Text’ -> [print, print]

в) [‘Text’, ‘Text’] -> print

‘Text’ виступає як текстовий рядок, а print як функція. Але це тільки текстова частина.

Графічна частина реалізується за допомогою графічного редактора Dia. Dia — кросплатформенний вільний редактор діаграм, частина GNOME Office, але може бути встановлений незалежно. Може використовуватись для створення різних видів діаграм: блок-схем алгоритмів програм, деревовидних схем, статичних структур UML, баз даних, діаграм сутність-зв’язок, радіоелектронних елементів, потокових діаграм, мережевих діаграм та інших.

Можливості:

- Підтримка діаграм потоків, структурних діаграм і т. д.
- Експорт в PostScript
- Завантаження і збереження у форматі XML
- Можливість опису нових об’єктів
- Установка властивостей за замовчуванням для об'єктів що додаються
- Зміна кольору шрифту і заливки блоків

2.1.1 Основні функції та концепції Pythonect

Залежно від інтерфейсу сценаріїв, напрямок потоку може бути представлена за допомогою простого тексту або взаємопов’язаних ліній.

Текстова мова сценаріїв прагне об'єднати швидкий і інтуїтивно зрозумілий вид оболонки сценаріїв

- «|» як синхронне вперед. Цей оператор проштовхує дані до наступної операції та блокується\очікує до її завершення. Наприклад: [1,2,3,4] | print – буде завжди виводити 1, 2, 3, 4(кожен елемент у власному потоці та в такому порядку)
- -> як асинхронне вперед. Цей оператор проштовхує дані до наступної операції та не блокується\очікує до її завершення. Наприклад: [1,2,3,4] -> print – може вивести 4, 3, 2, 1 або 2, 1, 3, 4 тощо
- Змінна у багатопоточності. Можна визначити змінну і встановити її з безліччю значень, в той же час (кожне значення map ()'ся до різних потоків) наступним чином: [x = 0, x = 1, x = 2] -> x -> print, та графічне представлення(рис.7):

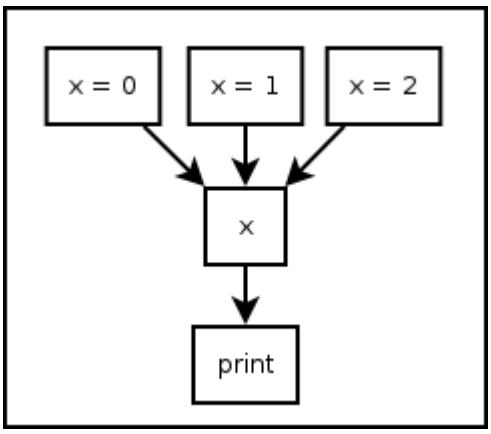

Рис.7 – Графічне представлення «безліч до одного»

Обидві версії будуть друкувати, кожен у своєму власному потоці, і не обов'язково в такому порядку: 0, 1, 2.
Також можна застосовувати будь-які комбінації типів даних Python: [x = 1, x = 1, x = 0.34, x = "Hello, world"] -> x -> print
Це буде друкувати, кожне у своєму власному потоці, і не обов'язково в такому порядку: 1, 1, 0.34, і " Hello, world ".

- Визначені змінні. Pythonect зумовлює дві змінні: «_» і «_!».Перша призначає поточне значення на потік. Друга – список з усіма поточними змінними в програмі.
- Також не має ключового слова if, замість цього булеві показники використовуються для визначення зупинки або продовження потоку. True як сигнал проходження, False як сигнал припинення.[6]

2.2.2 Лямбда функції

Для реалізації деяких функцій в Pythonect ми будем використовувати лямда функції. Наприклад, для множення або ділення чисел та іншої обробки даних.

Python підтримує досить цікавий синтаксис, що дозволяє обробляти та визначати невеликі однорядкові функції «на льоту». Позичені з Lisp, так звані lambda-функції можуть використовуватися будь-де, де вони потрібні

Наведемо приклад:

```
>>> def f(x):
...     return x*2
...
>>> f(3)
6
>>> g = lambda x: x*2    ❶
>>> g(3)
6
>>> (lambda x: x*2)(3)   ❷
6
```

Рис.8 – Приклад використовування лямбда-функцій[5]

1 – Ця функція виконує те ж саме, що і звичайна функція, що описана вище. Слід звернути уваги на скорочений синтаксис. Крім того, функція не має імені, проте може бути викликана за допомогою змінної, до якої вона присвоєна.

2 – Можна використовувати лямбда-функцію навіть без присвоєння змінній. Не найкорисніший приклад, але він ілюструє, що лямбда-функція може бути визначена одразу на місці її використання.[5]

2.2.3 Приклад роботи та реалізації Pythonect

Далі приведені декілька прикладів роботи Pythonect, зокрема реалізація графу для рішення квадратного рівняння та перевірку вхідних даних на наявність паліндрому.

wa– Реалізація графу для рішення квадратного рівняння в Pythonect

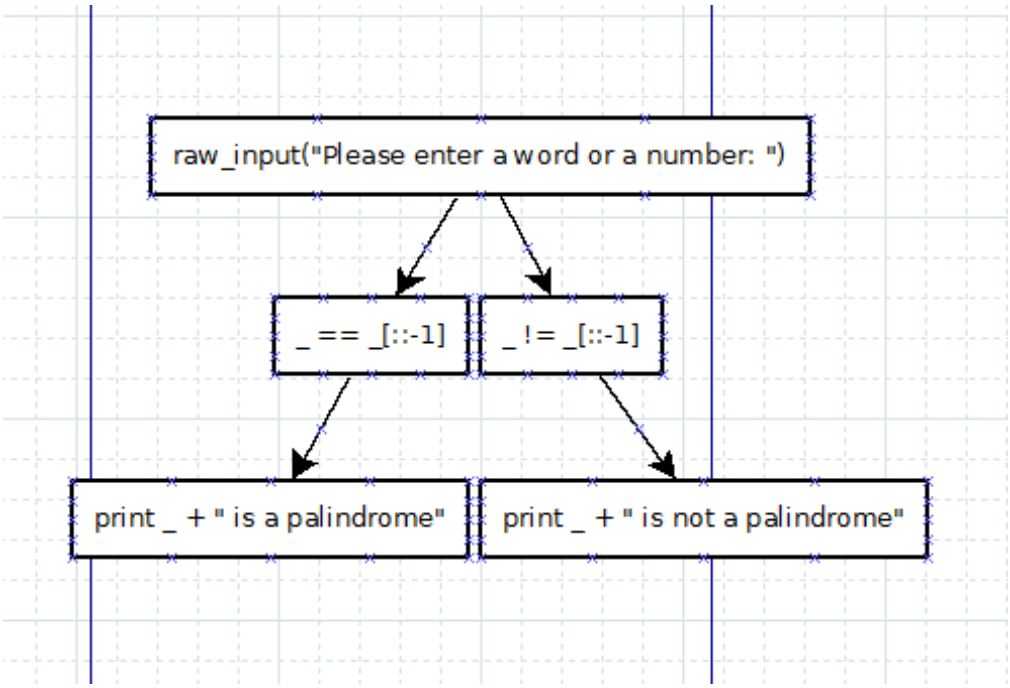

Рис.10 – Реалізація перевірки вхідних даних на наявність паліндрому

2.2.4 Висновки

Переваги:

- Досить потужний потенціал Python
- Dia, що використовується для графічного режиму, в свою чергу, також інтуїтивно зрозуміла та водночас покриває більшість потреб для візуальної розробки
- Pythonect, як і Dia, безкоштовні та знаходяться у вільному доступі.

- Підтримка декількох варіантів для візуальної розробки(наприклад, MS Visio)
- Кросплатформеність (може використовуватися на всіх платформах, що підтримують Python)
- Проста реалізація багатопоточності
- Є перспектива розвитку

Недоліки:

- Відсутність юзер-інтерфейсу
- Відсутність власного засобу для візуальної розробки
- Досить молодий засіб програмування, тому невелика кількість інструкцій та документації

2.2 LabVIEW

2.2.1 Загальний огляд програмного пакету LabVIEW

LabVIEW – це універсальне середовище для розробки систем збору, обробки даних та управління експериментом. Дане середовище має велику бібліотеку функцій, методів аналізу (спектральний та кореляційний аналіз, вейвлетний аналіз, методи фільтрації, статистична обробка та ін.), бібліотеки драйверів пристроїв, що відповідають найпоширенішим стандартам. Основою роботи в середовищі LabVIEW є графічне програмування з використанням блок-діаграм, що складаються з функціональних вузлів та зв’язків між ними). Всі дії зводяться до побудови структурної схеми програми в інтерактивній графічній системі з набором всіх необхідних бібліотечних образів, з яких збираються об’єкти.

Система LabVIEW включає в себе:

- ядро, що забезпечує можливість роботи програмних процесів, розділення апаратних ресурсів між процесами;

- компілятор графічної мови програмування „G”;
- інтегроване графічне середовище розробки, виконання та налаштування програм;
- набір бібліотек елементів програмування в LabVIEW, зокрема, бібліотеки графічних елементів інтерфейсу користувача, бібліотеки функцій та підпрограм, бібліотеки драйверів, бібліотеки програм для організації взаємодії з вимірювально-управляючими апаратними засобами і т.д.;
- добре структуровану довідкову систему;
- об’ємний набір програм-прикладів з можливістю як тематичного, так і алфавітного пошуку.

LabVIEW має вбудовану підтримку всіх програмних інтерфейсів, що використовуються у наш час, зокрема Win32 DLL, COM, NET, DDE, мережевих протоколів на базі IP, Data Socket та ін. До складу LabVIEW входять бібліотеки управління різними апаратними засобами та інтерфейсами, такими як PCI, VME, PLC, VISA, системами технічного зору та ін. Програмні продукти створені з використанням LabVIEW можуть доповнюватися фрагментами, що розроблені на традиційних мовах програмування, наприклад, C++, Pascal, Basic, Fortran. І, навпаки, можна використовувати модулі, що розроблені в LabVIEW у проектах, які створені в інших системах програмування. Таким чином, LabVIEW дозволяє розробляти практично будь-які програми, що взаємодіють з іншими видами апаратних засобів, які підтримуються операційною системою комп'ютера.

Використовуючи технологію віртуальних пристроїв, розробник може перетворити стандартний персональний комп'ютер і набір довільного контрольно-вимірювального обладнання у багатофункціональний вимірювально-обчислювальний комплекс. Однією з переваг LabVIEW є те, що для розробника та користувача є доступними функціонально ідентичні системи програмування для різних операційних систем, таких як Microsoft Windows, Linux, MacOS.

Програмування в LabVIEW максимально наближене до поняття алгоритм. Після того, як розробник продумує алгоритм роботи своєї майбутньої програми,

йому залишається лише нарисувати блок-схему цього алгоритму з використанням графічної мови програмування G. Тобто, автору не доводиться думати про комірки пам’яті, адреси, порти вводу-виведення, переривання та інші атрибути системного програмування. Дані будуть передаватися від блоку до блоку по „провідниках”, оброблятися, відображатися та зберігатися у відповідності з розробленим алгоритмом. Більше того, сам потік даних буде керувати ходом виконання вашої програми. Ядро LabVIEW може автоматично використовувати ефективні сучасні обчислювальні можливості, наприклад, такі як багатозадачність, багатопотоковість та ін.

Процес програмування в LabVIEW схожий на збирання моделі з конструктора. Програміст формує інтерфейс користувача програми – „мишкою” вибирає з наявних палітр-меню потрібні елементи (кнопки, регулятори, графіки та ін.) і розташовує їх на робоче поле програми. Аналогічно „рисується” алгоритм: з палітр-меню обираються необхідні підпрограми, функції, конструкції програмування (цикли, умовні конструкції та ін.). Потім, за допомогою мишки встановлюються зв’язки між елементами – створюються віртуальні дроти, по яких дані будуть проходити від джерела до приймача.[7]

2.2.2 Приклади реалізацій в LabVIEW

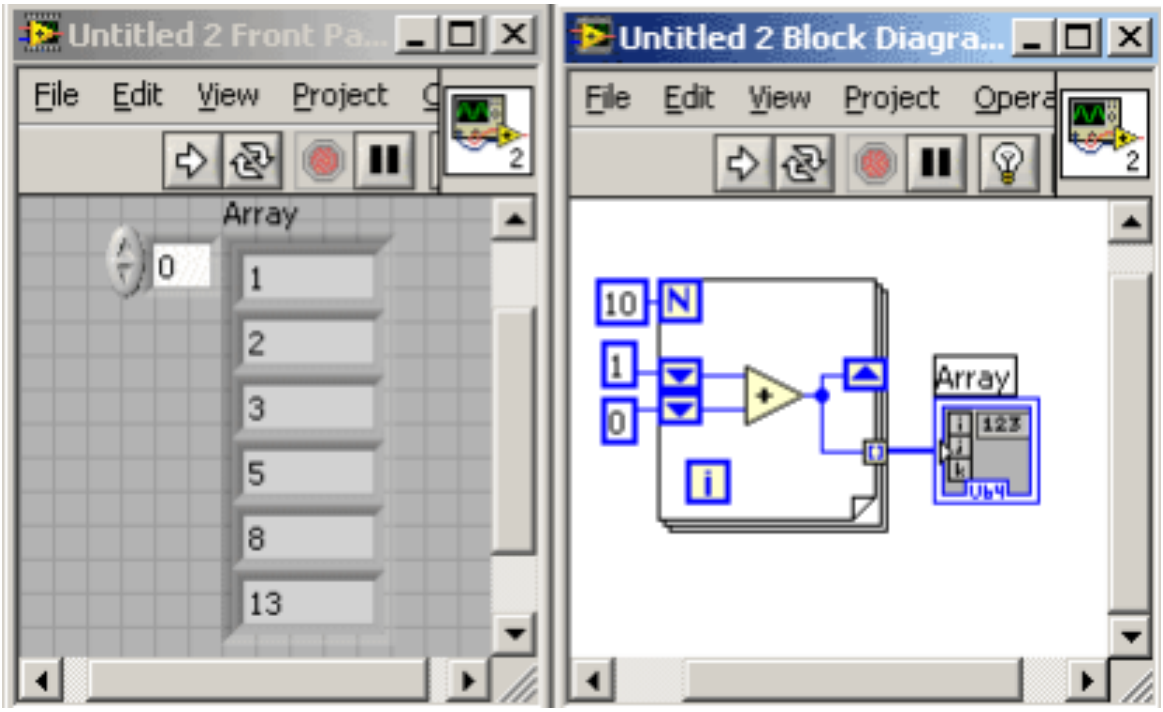

Рис.11 – Приклад реалізації знаходження чисел Фібоначі

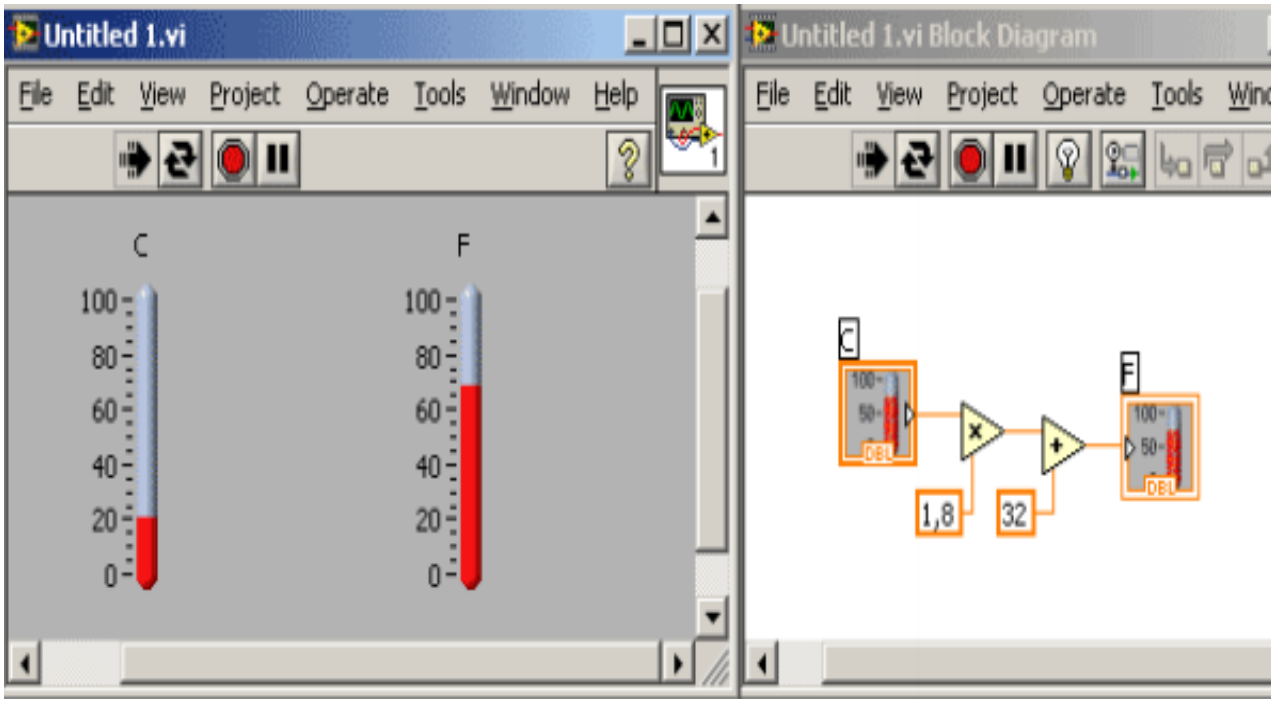

Рис.12 – Приклад реалізації конвертування температури

2.2.3. Висновки

Основні напрямки використання:

- Збір та аналіз сигналів датчиків

- Керування вимірювальними пристроями
- Автоматизація випробувань
- Інтегровані системи моніторингу та управління
- Навчальні цілі

Переваги:

- Першочергово створювалась для інженерів, тож середовище зроблено максимально інтуїтивно зрозумілим та порівняно легким у вивченні
- Завдяки графічному засобу програмування не потрібно витрачати багато часу на пошук помилок та недоліків «правопису»
- Досить великий набір бібліотек, зокрема, для роботи з периферійними пристроями, обробки та аналізу сигналів тощо
- Модульна система, тобто, є можливість розширювати функціональні можливості шляхом встановлення додаткових модулів та пакетів інструментів
- Дозволяє створювати завершені додатки і компілювати у повноцінні exe-додатки
- Це кросплатформене середовище. Також дозволяє писати додатки для Windows Mobile та ARM мікроконтролерів
- Досить прості засоби налагодження та тестування
- Досить легка робота з паралельними потоками(створенн, керування і тд.)
- Дозволяє створювати як невеличкі програмки, так і великі проекти з великим числом датчиків і складним інтерфейсом

Недоліки:

- Не зважаючи на досить велику гнучкість середовища, є деякі обмеження(наприклад, ігрові додатки, текстові процесори та ін. майже неможливо реалізувати)
- Через обмеженість в сфері застосування програє в плані більш загальних задач класичним середовищам програмування

- Також небагато поступається продуктивністю іншим мовам(ціна за легкість та гнучкість)

2.3 VHDL та середовище Active-HDL

Мова VHDL забезпечує високорівневу абстракцію опису апаратних засобів завдяки наявності як великої кількості попередньо визначених типів даних, так і можливості створювати ієрархічні організовані типи даних на основі базових.

Завдяки цим можливостям та досить «дружнього» концепту як для людини, так і для програмних засобів, він може використовуватися на етапах проектування, верифікації, синтезу та тестування апаратури, для передачі проектних даних, модифікації та супроводу проекту. Використовується для схем будь-якої складності – мікросхема, плата, блок, пристрій, комплекс.

2.3.1 Загальний огляд

VHDL є формальним записом, що призначений для опису функції та логічної організації цифрових систем. Функція системи визначається як перетворення значень на входах у значення на виходах, причому час у цьому перетворенні задається явно. Організація системи задається рядом зв’язаних компонентів.

Об’єкт проекту(entity) представляє собою опис компонента проекту, що має чітко вказані входи та виходи і чутко визначену функцію. Об’єкт проекту може представляти всю систему, що проектується, деяку підсистему, пристрій, вузол, стійку, плату, кристал, логічний елемент і тд.

У описі об’єкта можна використовувати можна використовувати компоненти, які, в свою чергу, можуть бути описані як самостійні об’єкти проекту більш низького рівня. Таким чином, кожен компонент об’єкта може бути зв’язаним з об’єктом нижчого рівня. У результаті такої декомпозиції об’єкту проекту користувач будує ієрархію об’єктів проекту, що представляють весь

проект в цілому і складену з декількох рівнів абстракцій. Така сукупність об’єктів проекту називається ієрархією проекту(design_hierarchy).

Кожен об’єкт проекту складається з, як мінімум, двох різних типів опису: опис інтерфейсу та одного чи більше архітектурних тіл.

Інтерфейс описується в оголошенні об’єкта проекту(entity_declaration) і визначає тільки входи та виходи.

Для опису поведінки об’єкта або його структури використовується архітектурне тіло(architecture_body).

Для того, щоб задати які об’єкти проекту використані для створення повного проекту, використовується оголошення конфігурації

У мові VHDL передбачений механізм пакетів для часто використовуваних описів, контактів, типів, сигналів. Цей опис розміщується в оголошенні пакету.

Якщо користувач використовує нестандартні операції або функції, то їх інтерфейси описуються в оголошенні пакету, а тіла містяться в тілі пакету

Таким чином, при описуванні цифрової схеми мовою VHDL, користувач може використовувати п’ять різних типів опису: оголошення об’єкта проекту, архітектурне тіло, оголошення конфігурації, оголошення пакету і тіло пакету. Кожний опис є самостійною конструкцією мови VHDL, може бути незалежно проаналізований аналізатором і тому отримав назву «Модуль проекту» (design_unit). Модулі проекту, у свою чергу, можна розділити на дві категорії: первинні та вторинні. До первинних модулів належать різні види оголошень. До вторинних – окремо аналізуємі тіла первинних модулів. Один або декілька модулів проекту можуть бути поміщені в один файл, що називається файлом проекту (design_file).

Кожний проаналізований модуль проекту переміщується в бібліотеку проекту і стає бібліотечним модулем (library_unit). Така реалізація дозволяє створювати будь-яке число бібліотек проекту. Кожна бібліотека в мові має своє логічне ім’я (ідентифікатор). Фактичне ім’я файлу, що містить цю бібліотеку, може збігатися або ні з логічним ім’ям бібліотеки проекту. Для асоціації

логічного імені бібліотеки з відповідним фактичним передбачений спеціальний механізм встановлення зовнішніх посилань.

По відношенню до сенсу роботи, у VHDL існують два класи бібліотек проекту: робочі бібліотеки та бібліотеки ресурсів.

Робоча бібліотека – це бібліотека, з якої у даному сеансі працює користувач і в яку поміщають бібліотечний модуль, отриманий у результаті аналізу модуля проектів.

Бібліотека ресурсів – це бібліотека, що утримує в собі бібліотечні модулі, посилання на які знаходяться в аналізованому модулі проекту.

У кожний конкретний момент користувач працює з однією робочою бібліотекою та довільним числом бібліотек ресурсів

Можливість створення багатьох бібліотек ресурсів дозволяє користувачеві класифікувати бібліотечні модулі по різним критеріям. Наприклад, в одні бібліотеці зберігати описи мікросхем однієї серії, в іншій – описи мікросхем другої серії і т.д. Або в одній бібліотеці зберігати описи мікросхем з одним типом затримки, а в іншій – опис мікросхем з другим типом затримки і т.д.

Будь-яка машинна мова характеризується визначеною множиною дозволених лексичних елементів. Текст опису мовою VHDL складається з одного або більше файлів проекту. Файл проекту являє собою послідовність лексичних елементів, кожен з яких складений із символів суворо визначеного набору символів. Текст кожного модулю проекту є послідовністю окремих лексичних елементів. Кожен лексичний елемент – це або обмежувач, або ідентифікатор (який може бути службовим словом), або абстрактний літерал, або строковий літерал, або коментар. Також у тексті VHDL описів дозволено використовувати тільки графічні символи і символи управління формату ASCII.[8]

2.3.2. Приклад використання мови та середовища

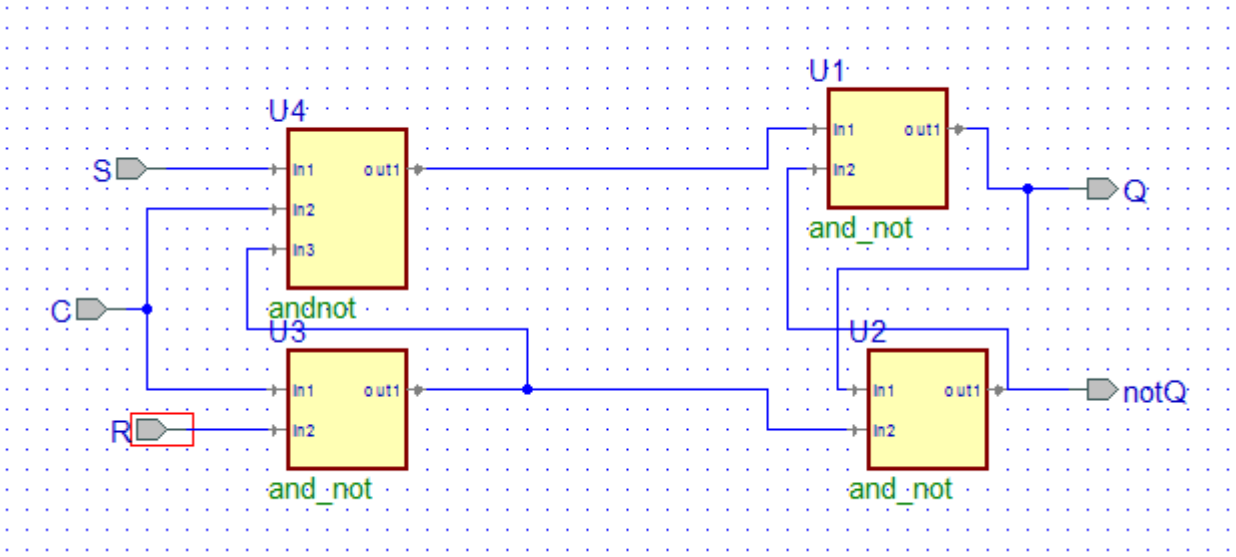

Рис.13 – Приклад графічної реалізації тригеру у середовищі ActiveHDL

Та одразу ж наведемо приклад текстової реалізації цього ж тригеру:

```
`ifdef _VCP
`else
`define library(a,b)
`endif
// ---------- Design Unit Header ---------- //
`timescale 1ps / 1ps
module TRIGGA (C,R,S,Q,notQ) ;
// ------------ Port declarations --------- //
input C;
wire C;
input R;
wire R;
input S;
wire S;
output Q;
wire Q;
```

```
output notQ;
wire notQ;
// ----------- Signal declarations -------- //
wire NET109;
wire NET57;
// -------- Component instantiations -------//
// synopsys translate_off
`library("U1","R_TRIGGISHE")
// synopsys translate_on
and_not U1
(       .in1(NET57),
        .in2(notQ),
        .out1(Q)
);
// synopsys translate_off
`library("U2","R_TRIGGISHE")
// synopsys translate_on
and_not U2
(       .in1(Q),
        .in2(NET109),
        .out1(notQ)
);
// synopsys translate_off
`library("U3","R_TRIGGISHE")
// synopsys translate_on
and_not U3
(
        .in1(C),
        .in2(R),
        .out1(NET109)
```

```
);
// synopsys translate_off
`library("U4","R_TRIGGISHE")
// synopsys translate_on
andnot U4
(
        .in1(S),
        .in2(C),
        .in3(NET109),
        .out1(NET57)
);
endmodule
```

2.3.3. Висновки

Основні переваги:

- Мова паралельного програмування
- Ієрархічність
- Зручна для сприймання людиною та опрацюванню на ЕОМ.
- За допомогою VHDL можна спростити введення та перевірку великого проекту
- VHDL програми надійні. Синтаксичний аналіз, програмне моделювання і компіляція в логічну схему швидко виявляють помилки проекту
- Універсальність. Розроблені раніше компоненти можна використовувати в багатьох інших проектах.
- Легка переносимість програм. Наприклад між мікросхемами різних технологій
- Компактність представлення досить складних логічних елементів

Недоліки:

- Через обмеженість в сфері застосування програє в плані більш загальних задач класичним середовищам програмування
- Переважно текстова частина досить складна та заплутана
- Конструкції, які мають аналогічні цілі мають дуже різний синтаксис

2.4 NoFlo

2.4.1 Загальний огляд

NoFlo це FBP середовище для JavaScript.

У flow-based програм логіка програмного забезпечення визначається у вигляді графу. Вузли графу – це екземпляри NoFlo компонентів, а ребра визначаються як з’єднання між ними.

NoFlo компонент реагує на вхідні повідомлення (пакети). Коли компонент отримує пакети на свої вхідні порти, він виконує певні прописані операції, та відсилає результат у вигляді пакету на вихідні порти. Не має спільного стану, єдиний спосіб комунікації між компонентами через пересилку пакетів.

NoFlo компоненті побудовані як прості JavaScript або CoffeeScript класи, що визначають вхідні та вихідні порти і реєструють різні обробки подій на них. При виконанні, NoFlo створює граф, або мережу з графів, екземпляри компонентів використовуються в графах та з’єднують їх.

NoFlo графи можуть працювати з різним набором вхідних парадигм. Той же потік може реагувати на вхідні запити HTTP, текстові повідомлення, а також зміни в файловій системі, також може давати вихід на різні цілі, такі як записи у базі даних, відповідати на HTTP запити або оновлювати панель. Це просто питання вибору компонентів, що будуть використовуватися.

Є два шляхи виконання потоко-орієнтованих програм за допомогою NoFlo. Якщо ваш додаток в цілому базується на потоках, то можна просто виконати його

у NoFlo. Потоко-орієнтовані програми, що виконані таким чином називаються незалежними графами.

Інший варіант реалізації полягає в тому, щоб вставляти NoFlo графи в існуючий додаток на JavaScript, використовуючи їх як постійну Node.js бібліотеку. Це досить корисно коли вже існує система і потрібно автоматизувати деякі частини як їх власні потоки або додати нового функціоналу

Коли ви починаєте працювати з NoFlo мережею, то вона нічого не робить самостійно, окрім завантаження компонентів графу та встановлює зв'язки між ними. Тоді компоненти починають відсилати повідомлення на вихідні порти, або реагувати на повідомлення, що надходять на вхідні порти.

Так як більшість компонентів вимагає деякі вхідні дані до того, як вони починають діяти, то звичайний спосіб виконання графу це посилання деяких початкових інформаційних пакетів (IIPs). Прикладом цього є відправка номерів порту, які приймають компоненти веб сервісу, або відправка назви файлу для зчитування файлу.

Така модель активації забезпечує багато можливостей:

- Починати потік на основі взаємодії з користувачем (командна оболонка, натиснувши кнопку)
- Починати потік в даний момент часу або даний інтервал (запуск графу першого числа кожного місяця, або через деякий час)
- Починати flow based на основі контексту (при прибутті у деяке фізичне місце, коли користувач переходить на даний сайт)[9]

2.4.2 Приклад

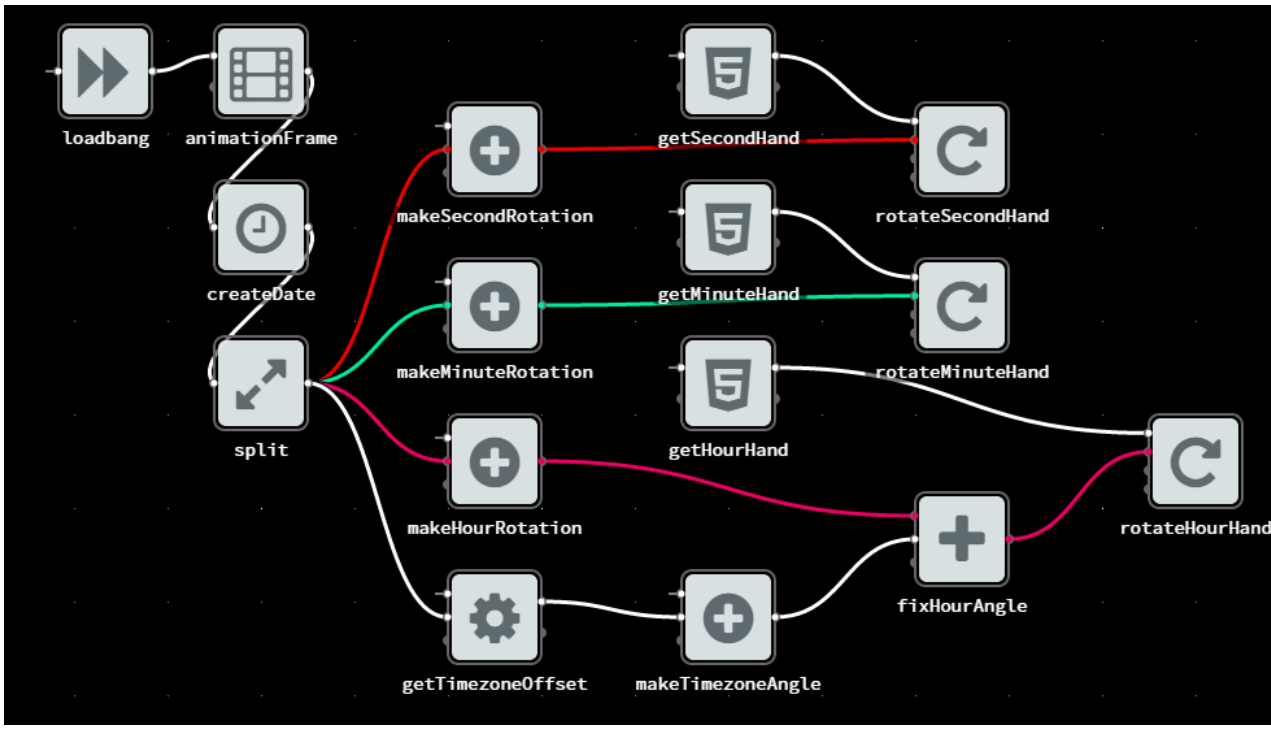

Рис.14 – Приклад реалізації анімованого годинника

Рис.15 – Результат роботи реалізації

Приклад текстової реалізації одного з модулів(makeSecondRotation):

function (exports, require, module){

var CreateDate, noflo,

```
        extend = function(child, parent) { for (var key in parent) { if (hasProp.call(parent, key)) child[key] = parent[key]; } function ctor() { this.constructor = child; } ctor.prototype = parent.prototype; child.prototype = new ctor(); child.__super__ = parent.prototype; return child; },
          hasProp = {}.hasOwnProperty;
        noflo = require("noflo");
        CreateDate = (function(superClass) {
          extend(CreateDate, superClass);
          CreateDate.prototype.description = 'Create a new Date object from string';
          CreateDate.prototype.icon = 'clock-o';
          function CreateDate() {
            this.inPorts = new noflo.InPorts({
              "in": {
                datatype: 'string',
                description: 'A string representation of a date in RFC2822/IETF/ISO8601 format'
              }
            });
            this.outPorts = new noflo.OutPorts({
              out: {
                datatype: 'object',
                description: 'A new Date object'
              }
            });
            this.inPorts["in"].on('begingroup', (function(_this) {
              return function(group) {
                return _this.outPorts.out.beginGroup(group);
              };
            })(this));
            this.inPorts["in"].on("data", (function(_this) {
```

```
      return function(data) {
        var date;
        if (data === "now" || data === null || data === true) {
          date = new Date;
        } else {
          date = new Date(data);
        }
        return _this.outPorts.out.send(date);
      };
    })(this));
    this.inPorts["in"].on('endgroup', (function(_this) {
      return function() {
        return _this.outPorts.out.endGroup();
      };
    })(this));
    this.inPorts["in"].on('disconnect', (function(_this) {
      return function() {
        return _this.outPorts.out.disconnect();
      };
    })(this));
  }
  return CreateDate;
})(noflo.Component);
exports.getComponent = function() {
  return new CreateDate;
};
```

2.4.3 Висновки

Переваги:

- NoFlo графи можуть працювати з різним набором вхідних парадигм
- Можливість працювати зі стандартними додатками
- Орієнтованість на широкий спектр задач, а не тільки на якусь вузьку спеціалізацію
- Не зважаючи на недавню реалізацію має хорошу базу документації та інструкцій по застосуванню, що спрощує роботу з цією системою
- Зручна робота з паралельним програмуванням
- Дружній та зручний інтерфейс, зокрема для візуальної частини

Недоліки:

- Орієнтованість тільки під Java Script або Coffee Script
- Досить «молодий» засіб (ще не реалізовано усіх задуманих особливостей)

2.5 Результати випробувань різноманітних dataflow систем на деяких обчислювальних алгоритмів

У пунктах 2.1-2.4 було досліджено dataflow системи та розглянуті реалізації наступних алгоритмів: у Pythonect – реалізація графу алгоритму вирішення квадратних рівнянь та алгоритму знаходження паліндромів у вхідних даних; LavVIEW – програми для конвертації температури та розрахунку чисел Фібоначі; VHDL – реалізація синхронного RS тригера; NoFlo – реалізація аналогового годинника. За результатами досліджень було зроблено наступні висновки, що наведені у таблиці 1. Як висновок можна зазначити, що для ряду задач, зокрема реалізацій інженерних питань, паралельного програмування, краще використовувати dataflow системи. Також їх використання забезпечує нам набагато кращу наочність при плануванні структури та логіки системи, що надає перевагу у порівнянні з класичними методами.

Таблиця 1. Загальні висновки по результатах випробувань

Розглянута система	Переваги	Недоліки
Pythonect	• Зручний синтаксис Python • Добре пристосована для паралельного програмування • Кросплатформеність	• Невеликий функціонал, що пов’язано з недавньою розробкою • Відсутність власного засобу для візуального програмування
LabVIEW	• Кросплатформеність • Зручна розробка у сфері інженерних питань • Розробку програм можна вести тільки візуальними засоби	• Обмежена сфера застосування
NoFlo	• Графи NoFlo можуть працювати з різним набором вхідних парадигм • Орієнтованість на широкий спектр задач • Можливість повторного використання компонентів	• Орієнтованість тільки на Java Script
VHDL	• Робота з паралельним програмуванням • Ієрархічність • Універсальність. Розроблені раніше компоненти можна використовувати в багатьох інших проектах.	• Обмежена сфера застосування • Досить складна текстова частина

2.6 Висновки

Усі розглянуті системи та засоби програмування з використанням data flow парадигми мають свої переваги та недоліки, але досить багато пунктів співпадають у різних засобах, далі виділимо основні з них.

Спільні переваги. Перш за все, за допомогою розглянутих систем досить зручно та просто працювати з паралельним програмуванням, це зумовлюється використанням dataflow парадигми. Усі використовують візуальні засоби програмування(як свої вбудовані, так і зовнішні), що надзвичайно спрощує розуміння архітектури, загального дизайну та концепцій проекту, що досить важливо при плануванні та обговоренні проекту, спрощує його розробку та тестування, дозволяє скорочувати витрати часу та фінансових засобів. Наступна позитивна сторона це те, що більшість таких систем є кросплатформенними, що спрощує питання портування додатків на різні пристрої з різними ОС. Також не можна оминути увагою те, що раніше написані модулі(компоненти) можна повторно використовувати при потребі в розробці замість того, щоб переписувати їх наново кожен раз, що також дозволяє економити час та інші витрати на розробку додатків.

Основні недоліки. Багато dataflow засобів вузькоспеціалізовані, в основному для інженерних реалізації, створення та опису схем та ін. Хоча більш сучасні засоби, що створюються в даний період орієнтуються на вирішенні більш широкого спектру задач. Також пов’язано з попереднім пунктом те, що FBP метод не використовується в широкій аудиторії користувачів, хоча з подальшим розвитком безумовно буде набирати популярність.

3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Визначення поняття охорони праці дається в ст. 1 Закону України від 14 жовтня 1992 р. «Про охорону праці». Охорона праці — це система правових, соціально-економічних, організаційно-технічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження здоров'я і працездатності людини в процесі праці. У поняття охорони праці входять і всі ті заходи, що спеціально призначені для створення особливих полегшених умов праці для жінок і неповнолітніх, а також працівників зі зниженою працездатністю.

У даній дипломній роботі було досліджено парадигму керування потоками даних. Робота може бути застосована для реалізації програмного забезпечення, тобто використовуватись у громадських та офісних приміщеннях.

Метою цього розділу є визначення відповідності приміщення нормам охорони праці, оскільки людина постійно взаємодіє з оточуючим її виробничим середовищем, а тому необхідно забезпечити їй максимально сприятливі і безпечні умови праці.

Правильна взаємодія людини з навколишнім середовищем насамперед залежить від оцінки шкідливих ті небезпечних виробничих факторів, параметрів мікроклімату та освітлення. Ще одним важливим елементом є аналіз пожежної безпеки. Тобто визначення категорії приміщення, а також правильне обрання вогнегасника та пожежної сигналізації.

3.1 Аналіз умов праці

Приміщення, в якому використовується дана методика розробки, знаходиться на сьомому поверсі 12-поверхового будинку. Вікно розташовано на північну сторону, площа засклення 20%. Кімната має як природне освітлення, так і штучне.

Важливе значення мають характеристики робочого приміщення та робочого місця. Побудуємо робоче приміщення з допомогою спеціального

програмного пакету Microsoft Visio. План-схема робочого приміщення зображена на рис. 16:

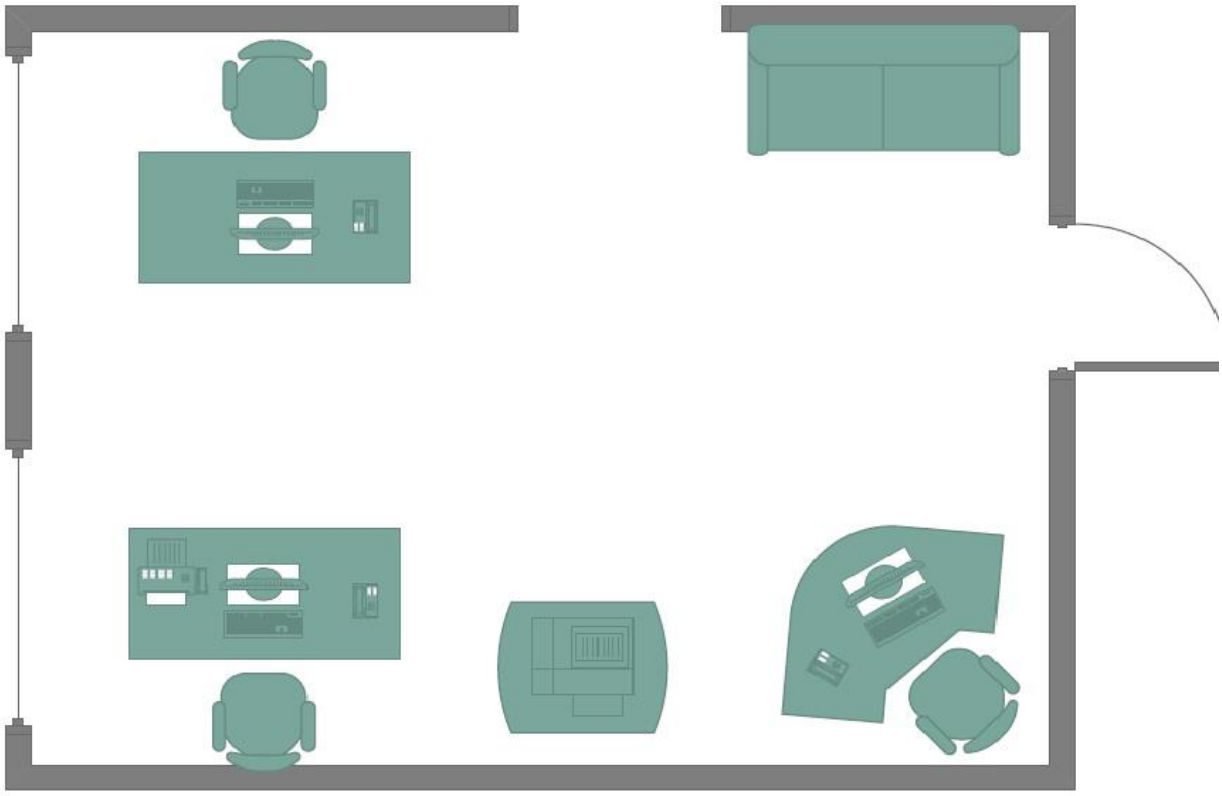

Рис.16 – План-схема робочого приміщення

Довжина приміщення a = 6 м;

Ширина приміщення b = 4.5 м;

Висота стелі в приміщенні h = 3.5 м.

Площа приміщення: $S = a * b = 6 * 4.5 = 27$ кв.м

Об'єм приміщення: $V = a * b * h = 6 * 4.5 * 3.5 = 94.5$ куб.м

В даному приміщенні одночасно працюють 3 особи.

Таким чином на одного працівника приходиться:

$S = \frac{27}{3} = 9$ кв.м.

$V = \frac{94.5}{3} = 31.5$ куб.м

Відомо, що об'ємно-планувальні рішення будівель та приміщень для роботи з ВДТ ЕОМ і ПЕОМ (візуальними дисплейними терміналами електронно-обчислювальних машин і персональних ЕОМ) мають відповідати вимогам:

- площа на одне робоче місце має становити не менше ніж 6,0 кв.м.,
- об'єм - не менше ніж 20,0 куб.м.

Таким чином, можна зробити висновок, що приміщення відповідає нормативам [10].

Розглянемо тепер відповідність характеристик робочого місця нормативним. Для цього зведемо основні вимоги до організації робочого місця і відповідні фактичні значення у таблиці 2.

Таблиця 2 – Характеристики робочого місця

Найменування параметра	Значення	
	Фактичне	Нормативне
Висота робочої поверхні, мм	750	680 – 800
Висота простору для ніг, мм	720	не менше 600
Ширина простору для ніг, мм	600	не менше 500
Глибина простору для ніг, мм	700	не менше 650
Висота поверхні сидіння, мм	440	400 – 500
Ширина сидіння, мм	400	не менше 400
Глибина сидіння, мм	400	не менше 400
Висота опорної поверхні спинки, мм	500	не менше 300
Ширина поверхні спинки, мм	400	не менше 380

Відстань від очей до екрану, мм	800	700 – 800

3.2 Аналіз шкідливих та небезпечних чинників

Небезпечним називають виробничий чинник, дія якого на організм працюючого у відповідних умовах може призводити до травм або іншого раптового, різкого погіршення стану здоров’я.

Шкідливим називається виробничий чинник, дія якого на організм працюючого може призводити у певних умовах до захворювання чи зниження рівня працездатності.

Працівники пов'язані з постійною роботою за комп'ютером, піддаються впливу факторів виробничої небезпеки, основними з яких є: підвищений рівень напруги в електричному ланцюзі, запиленість повітря робочого приміщення, несприятливі мікрокліматичні умови, некоректний рівень освітленості.

3.2.1 Мікроклімат

У даному приміщенні працюють спеціалісти, характер роботи яких переважно аналітичний (так як дана методика мінімізує потреби у переписуванні алгоритмів десятки разів). Тому згідно з ДСН 3.3.6.042-99 цю роботу можна віднести до категорії легка 1а.

У таблиці 3 наведені оптимальні значення параметрів мікроклімату для категорії робіт 1а, а також фактичні значення цих параметрів у досліджуваному приміщенні [11].

Таблиця 3 – Параметри мікроклімату на робочому місці

Температура зовнішнього повітря, °С	Параметри повітряного середовища на робочому місці					
	Оптимальні			Фактичні		
	Температура	Відносна	Швидкість руху повітря, м/с	Температура	Відносна	Швидкість руху

	ратура, °С	вологість, %		ратура, °С	вологість, %	повітря, м/с
Вище +10	23 – 25	40 – 60	0.1	24 – 26	40 – 60	< 0.1
Нижче +10	22 – 24	40 – 60	0.1	22 – 23	40 – 50	< 0.1

Всі показники задовольняють вимогам для робіт категорії легка Іа і є задовільними для здоров’я людини.

3.2.2 Шуми

Шум являється одним із найбільш розширених факторів зовнішнього середовища, який негативно впливає на організм людини. У відповідності до ДСН 3.3.6.037 -99, в приміщеннях, де використовується робота з ЕОМ, рівень шуму не повинен перевищувати 50 дБ.

Можливі джерела шуму в приміщенні:

– внутрішні: персональний комп’ютер (системи охолодження ЕОМ, процесори ЕОМ, дисководи), факсимільна машина, принтер, телефони, системи охолодження кімнати;

– зовнішні: шум вуличного транспорту.

Наведемо таблицю 4 джерел шуму в робочій кімнаті та рівня шуму, який вони створюють.

Таблиця 4 Джерела шуму в кімнаті

Джерело шуму	Рівень шуму La, дБА	Час дії шуму t, год	Кількість джерел шуму N
Зовнішній шум	35	8	1
ПК	37	8	3

Кондиціонер	32	5	1
Струменний принтер	52	3	1
Факсимільна машина	54	0.5	1

Визначимо $L_{екв.}$ еквівалентний рівень шуму за 8 робочих годин:

$$L_{\text{екв}} = 10 * lg\frac{1}{T}\sum t_i * 10^{0.1*La} == 10 \cdot lg\frac{1}{8} \cdot (10^{3.5} + 3 \cdot 10^{3.7} + 10^{3.2} + 10^{5.2} + 10^{5.4}) \approx 47 \text{ дБА}$$

Отже можна зробити висновок, що у приміщенні рівень звукового тиску та рівень звуку на робочих місцях відповідає вимогам, так як в приміщеннях управління та робочих кімнатах допустимий еквівалентний рівень звуку має становити не більше 50дБА [12].

3.2.3 Освітлення

Відповідні умови освітлення в приміщеннях з ЕОМ досягаються використанням як загального, так і місцевого освітлення, розміщення якого регулюються відповідними нормативними актами. В якості джерела світла при штучному освітленні застосовуються люмінесцентні лампи типу ЛБ-40 (низького тиску). Представимо їх технічні характеристики:

• потужність – 40 Вт;

• напруга на лампі – 103 В;

• світловий потік: номінальний – 3120 лм,

мінімальний – 2810 лм;

• довжина лампи: без штирків – 1199.4 мм,

із штирками – 1213.6 мм;

• діаметр – 40 мм.

Система загального освітлення становить собою світильники, розташовані ліворуч від робочих місць, паралельно лінії зору працюючих.

Відблискування на робочих поверхнях обмежується за рахунок правильного вибору типів світильників та розташування робочого місця по відношенню до джерел штучного освітлення. Яскравість відблисків на екрані моніторів не перевищує 40 кд/м2. Покриття підлоги є матовим з коефіцієнтом відбиття 0,3-0,5. Поверхня підлоги рівна, неслизька, з антистатичними властивостями. Освітлення використовується достатньо рівномірно розподілено на робочих поверхнях і в навколишньому просторі, уникаються різкі тіні, освітлення є рівномірним в часі; напрямок випромінюваного освітлювальними приладами світлового потоку оптимальний.

3.2.4 Пожежна безпека

В нашому випадку приміщення не розташоване у підвалі або на цокольному поверсі, що повністю відповідає нормам [10].

За вибухопожежною та пожежною безпекою приміщення належить до категорії В, оскільки в цьому приміщенні знаходяться горючі речовини та матеріали: ізоляційні матеріали (так, наприклад, ПВХ має температуру займання 385^{0}С, стеклотекстоліт - 350^{0}С, папір - 200^{0}С, тканини - 250^{0}С), столи, стільці, крісла, шафа, стелаж, тумбочка – виготовлені переважно з дерева.

Вся споруда належить до категорії В. Виходячи з роду матеріалів, в приміщенні можлива пожежа класу А [13].

Причини займання – коротке замикання, перенавантаження обладнання, великі перехідні опори, зіпсованість електроустаткування, порушення протипожежного режиму (тобто неправомірне використання обігрівачів, електрочайників, кип’ятильників та ін.).

Приміщення оснащене системою автоматичної пожежної сигналізації. Виконано норматив по розміщенню сповіщувачів 1 на 10 кв. м. Крім, того проводяться організаційні міри (навчання та контроль) за експлуатацією,

технічні міри в обладнанні (захист від перевантажень), застосовуються важко горючі поверхні (фторопластові). Для гасіння можливої пожежі в приміщенні розташовано три вогнегасники вуглекислотного типу ВВК-1.4 і безпровідна автоматична система порошкового пожежогасіння і сигналізації "ГАРАНТ-Р" (ПО-2), також присутні інструкції по використанню протипожежних засобів [13]. Пристрої обчислювальної техніки встановлені далеко від опалювальних і нагрівальних приладів (відстань не менше 1 м і в місцях, де ускладнена їх вентиляція і попадання прямих сонячних променів).

Робоче місце захищене від перенавантажень, а також в ньому проводиться експлуатаційна профілактика. Є також евакуаційні сходи. Всі вимоги дотримано, план евакуації відповідає нормативу. Всі працівники пройшли інструктаж з питань пожежної безпеки робітників.

3.2.5 Електробезпека

Приміщення належить до 1 групи – приміщення з ЕОМ та споруди управління. В цьому приміщенні дотримуються технічних та організаційнх мір профілактики електротравматизму. Заземлені конструкції надійно

захищені діелектричними сітками від випадкового дотику. ЕОМ та периферійні пристрої ЕОМ, інше устаткування (світильники), електропроводи та кабелі мають апаратуру захисту від струму короткого замикання та інших аварійних режимів.

Лінія електромережі для живлення ЕОМ та периферійних пристроїв ЕОМ виконується як окрема групова три-провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників.

ЕОМ та периферійні пристрої ЕОМ підключаються до електромережі тільки за допомогою справних штепсельних з’єднань і електророзеток заводського виготовлення. Штепсельні з’єднання та електророзетки крім контактів фазового та нульового робочого провідників мають спеціальні контакти для підключення нульового захисного провідника. Електромережі

штепсельних з’єднань та електророзеток для живлення персональних ЕОМ та периферійних пристроїв ЕОМ виконуються за магістральною схемою, по 6 електророзеток в одному колі.

Для протирання підлоги застосовуються рідини, пара яких не утворює вибухопожежонебезпечних сумішей з повітрям та не викликає корозії контактів електричних з’єднань.

3.3 Рекомендації щодо поліпшення умов праці

У зв’язку зі специфікою робіт з ЕОМ можна порекомендувати виконання комплексів вправ для психічного та психологічного розвантаження.

При інтенсивній роботі з вхідними даними, редагуванні програм, читанні інформації з екрану монітора безперервна тривалість роботи не повинна перевищувати 4-х годин (при 8-годинному робочому дні). Задля зниження напруженості праці необхідно, якщо це можливо, рівномірно розподіляти навантаження і раціонально чергувати характер діяльності.

Варто щогодини робити перерву на 5-10 хвилин, а через 2 години – на 15 хвилин. Один або кілька разів у годину необхідно виконувати серію легких вправ на розтягування, що можуть зменшити напругу, накопичену в м’язах при тривалій роботі за комп’ютером.

3.4 Висновки

У даному розділі були розглянуті умови праці для виробничого приміщення, у якому операторами експлуатуються програмний продукт. Розміри приміщення та параметри робочих місць відповідають нормам чинного законодавства з охорони праці.

Було проведено аналіз шкідливих та небезпечних виробничих чинників, наявних у даному приміщенні. Значення параметрів мікроклімату, виробниче

освітлення та засоби пожежної безпеки приміщення відповідають необхідним нормативам.

ВИСНОВКИ

Дана дипломна робота присвячена дослідженню dataflow парадигми, систем та засобів програмування з її використанням.

У першому розділі було досліджено dataflow парадигми та потоко-орієнтоване програмування, що базується на ній. Також був розглянутий метод графічного представлення за допомогою SOA. Хоча цей метод програмування був розроблений запропонований досить давно, проте він не набув великої популярності у широкого загалу розробників, проте з розвитком різноманітних підходів до інформаційних технологій та широким використанням SOA та візуального програмування, FBP набирає свою заслужену популярність. У висновках до першого розділу виділені основні переваги та недоліки FBP, проте слід виділити декілька найголовніших. По-перше, це можливість повторного використання вже написаних модулів, що дає нам можливість зробити додаток у візуальному режимі написавши мінімум коду(а в деяких випадках взагалі без нього), що в свою чергу набагато спрощує розробку. По-друге, завдяки візуальному представленню архітектури програми(у тому числі, за допомогою SOA) полегшується планування та дизайн системи, планування розробки та тестування, що в свою чергу дозволяє економити час та інші ресурси, зокрема фінансові.

У другому розділі було досліджено та описано основні характеристики, позитивні та негативні сторони систем та засобів програмування з використанням парадигми керування потоками даних. У висновку було виділено основні спільні переваги та недоліки. Одними з найголовніших позитивних ознак є легка робота з паралельним програмуванням, здатність використовувати візуальний спосіб розробки. Серед основних недоліків – переважно вузька спеціалізація розробки.

Перспективою для подальших досліджень є розробка open-source фреймворку для веб-додатків на парадигмі керування потоками даних

ПЕРЕЛІК ПОСИЛАНЬ

1. Харченко К.В. Парадигма керування потоками даних та їх графічне представлення в SOA./ Харченко К.В.// "Східно-Європейський журнал передових технологій", №3/9 (69): Харків. 2014. - С. 22 - 29.
2. Офіційний сайт John Paul Morrison. – Режим доступу : http://www.jpaulmorrison.com/fbp.html. – Дата доступу : 17.05.2015.
3. GitHub Flow-based programming specification. – Режим доступу : https://github.com/flowbased/flowbased.org/wiki – Дата доступу : 18.05.2015.
4. Офіційний сайт Service Oriented Architecture. – Режим доступу : http://serviceorientation.com/whatissoa/service_oriented_architecture. – Дата доступу : 19.05.2015.
5. Офіційний сайт Python. – Режим доступу : https://www.python.org. – Дата доступу : 19.05.2015.
6. Pythonect 0.7.dev0 documentation. – Режим доступу : http://docs.pythonect.org/en/latest/index.html. – Дата доступу : 19.05.2015.
7. Офіційний сайт National Instruments, розділ LabView. – Режим доступу : http://www.ni.com/labview. – Дата доступу : 20.05.2015.
8. Івченко В.Г. Применение языка VHDL при проэктиировании специализированых СБИС; Навчальний посібник. Таганрог, 1999, 80 с.
9. Офіційний сайт NoFlo. – Режим доступу : http://noflojs.org/documentation. – Дата доступу : 21.05.2015.
10. Правила охорони праці під час експлуатації електронно-обчислювальних машин. НПАОП 0.00-1.28-10 [Електронний ресурс]. – Режим доступу: http://document.org.ua/pravila-ohoroni-praci-pid-chas-ekspluataciyi-elektronno-obch-nor17970.html. – Дата доступу : 15.05.2015.
11. Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99. – [Чинний від 2000-01-01]. – К. : МОЗ України, 2000. – 42 с. – (Національні стандарти України).

12. Санітарні норми виробничого шуму, ультразвуку та інфразвуку : ДСН 3.3.6.037-99. – [Чинний від 2000-01-01]. – К. : МОЗ України, 2000. – 37 с. – (Національні стандарти України).

13. Про затвердження типових норм належності вогнегасників: наказ Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 2 квітня 2004 року N 151 [Електронний ресурс] // Офіційний веб-портал Верховна Рада України – Режим доступу : http://zakon2.rada.gov.ua/laws/show/z0554-04 - Дата доступу : 15.05.2015.

14. Природнє і штучне освітлення : ДБН В.2.5-28:2006 – [Чинний від 2006-10-01]. – К. : Міністерство будівництва, архітектури та житлово-комунального господарства України, 2006. – 68 с. – (Національні стандарти України).