УДК 004.432.2, 004.422.83 *doi: 10.30970/vam.2023.31.11964*

ПОБУДОВА ВІКОННИХ ЗАСТОСУНКІВ У СЕРЕДОВИЩІ PHARO ЗА ДОПОМОГОЮ БІБЛІОТЕКИ SPEC

С. Ярошко, О. Ярошко

Львівський національний університет імені Івана Франка,
вул. Університетська 1, Львів, 79000
e-mail: serhiy.yaroshko@lnu.edu.ua

На прикладі побудови застосунку, що моделює роботу вуличного світлофора, продемонстровано використання засобів бібліотеки Spec і середовища Pharo. Стисло описано синтаксис мови Pharo, описано створення класів, що утворюють модель даних задачі. Для побудови графічного інтерфейсу користувача використано відображення графічних компонент, загальних елементів керування, стандартні компоненти-макети і власноруч спроєктоване відображення. Побудований засобами Pharo застосунок порівняно з WPF-застосунком мовою C#, який розв'язує ту саму задачу. Підтверджено переваги Pharo. Автори хотіли б привернути увагу користувача до Pharo – вільно поширюваної об'єктно-орієнтованої системи з відкритим кодом.

Ключові слова: Pharo, Spec, WPF, C#, віконний застосунок, графічний інтерфейс користувача, візуальний компонент, клас, об'єкт.

1. Вступ

Інтерес до програмного забезпечення з відкритим кодом постійно зростає впродовж останніх років [1], так само й до програм, які вільно поширюють. Особливо актуальним є використання вільного та відкритого програмного забезпечення в навчальних закладах, зокрема для навчання програмуванню. Одна з відкритих систем програмування – об'єктно-орієнтована система Pharo [2]. Її тривалий час використовують на факультеті прикладної математики та інформатики ЛНУ ім. Івана Франка, а університет є академічним партнером Pharo Consortium [3].

Pharo (Фарó) – це сучасна об'єктно-орієнтована динамічно типізована мова програмування, дуже близька за синтаксисом до класичної мови Smalltalk. Водночас Pharo – це інтерактивне середовище, оснащене досконалими інструментами для "живого" програмування, яке дає змогу розробникові легко створювати, налагоджувати, поширювати програмний код, ба більше – змінювати його під час виконання. Pharo – багатоплатформна система, яка однаково добре працює в різних операційних системах: OS X, Windows, Linux, Android, iOS та Raspberry Pi. Pharo – універсальна мова. Завдяки наявним бібліотекам можна писати настільні та веб застосунки, взаємодіяти з базами даних, застосовувати числові методи, розв'язувати задачі штучного інтелекту тощо.

На одному навчальному прикладі описано процес створення у Pharo віконного застосунку за допомогою бібліотеки Spec і проведено порівняння з можливостями Windows Presentation Foundation (WPF), мови C# і середовища розробки Visual Studio компанії Microsoft для побудови такого ж застосунку. Продемонстровано переваги та особливості Pharo. Автори сподіваються, що стаття сприятиме популяризації Pharo.

2. Коротко про синтаксис Pharo

Розробники Pharo небезпідставно гордяться простотою синтаксису мови. Кажуть, що весь його опис можна помістити на поштівку, і це справді так. Pharo відрізняється від інших мов простотою і зрозумілістю концепцій:

- усі сутності програми є об'єктами, екземплярами класів;
- так само все в системі є об'єктами, навіть вікна, класи, компілятор;
- об'єкти взаємодіють між собою за допомогою надсилання повідомлень.

Pharo використовує всього кілька синтаксичних конструкцій:

- надсилання повідомлення, яке має вигляд *отримувач селектор аргумент*;
- присвоєння вигляду *змінна := об'єкт*;
- повернення результату виконання методу ^*об'єкт*, за замовчуванням метод повертає отримувача повідомлення;
- зображення літералів, оголошення тимчасових змінних методу.

Мова програмування Pharo (чи Smalltalk) не надто поширена, тому для кращого розуміння опишемо стисло її синтаксис. Докладний його опис можна знайти в [4].

Повідомлення у Pharo поділяють на три категорії: унарні, бінарні, ключові – тут вони перелічені за спаданням пріоритету.

Унарне повідомлення складається з одного селектора (без аргументу). Наприклад, у кожного об'єкта можна запитати, екземпляром якого класу він є: *3.1416 class*. Кожен об'єкт уміє зобразити себе рядком: *myObj printString*. Унарне повідомлення можна надіслати результатові іншого унарного повідомлення. Наприклад, ланцюжок *100 factorial printString size* повідомить, скільки цифр у десятковому записі числа 100!.

Бінарне повідомлення має один аргумент, а його селектор складається з одного-двох символів. Наприклад, арифметичні дії програмують за допомогою бінарних повідомлень. У виразі *7 * 8* об'єкт *7* – отримувач, зірочка – селектор, об'єкт *8* – аргумент. Бінарні повідомлення можна об'єднувати в ланцюжки як унарні.

Селектор *ключового* повідомлення складається з одного або більше ключів (слова з двокрапкою наприкінці) та відповідної кількості аргументів, які записують після двокрапок. Наприклад, щоб задати значення першого елемента деякого масиву, можна використати повідомлення *myArray at: 1 put: 'Hello'*. Можна надіслати низку повідомлень одному отримувачу. Тоді їх відокремлюють крапкою з комою.

Об'єктна модель Pharo. Щоб написати програму мовою Pharo, потрібно оголосити клас (чи класи) і визначити його методи. Pharo підтримує просте наслідування класів. Ієрархію засновано на єдиному кореневому класі *Object*. Кожен клас може мати довільну кількість підкласів. Щоб оголосити новий клас, базовому класові *надсилають повідомлення*. Оголошення пов'язаних класів об'єднують у пакети. Створений клас стає повноправним членом наявної ієрархії класів.

Усі методи об'єкта – відкриті, немає приватних чи захищених. Усі змінні об'єкта (поля даних) – захищені: об'єкт може використовувати власні змінні та успадковані від надкласів. Доступитися до змінних об'єкта можна лише за допомогою повідомлень (оголошених методів доступу). Приватність об'єкта захищена на рівні екземпляра, а не на рівні типу, як в інших мовах. Це означає, що екземпляр класу не має доступу до змінних іншого екземпляра цього ж класу. Навіть клас не має доступу до пам'яті своїх екземплярів. Клас у Pharo – також об'єкт, тому й він може

мати змінні та методи. Щоб створити екземпляр, класові надсилають відповідне повідомлення.

Турбуватися про очищення пам'яті використаних об'єктів не треба, бо в системі працює автоматичне збирання "сміття".

Структури керування реалізовано методами відповідних класів. Наприклад, галуження виконують методи *ifTrue: aBlock*, *ifTrue: trueBlock ifFalse: falseBlock* класу *Boolean*, повторення – метод *to: stopValue by: step do: anUnaryBlock* класу *Number*. За потреби можна оголошувати власні методи для керування потоком виконання. Як видно з прикладів, аргументами таких методів часто є блоки. *Блок* у Pharo – це частина коду, обрамлена квадратними дужками. Блоки найбільше схожі на лямбда-вирази мови C++ чи C#.

3. Формулювання задачі

Для повноцінної демонстрації можливостей Pharo і Spec використаємо просте у формулюванні завдання, для виконання якого потрібно застосувати досить багато різних засобів.

Задача 1. *Створіть віконний застосунок, який моделює роботу вуличного світлофора. Він зображає ліхтарі червоного, жовтого та зеленого кольорів, дає змогу перемикати їх у ручному й автоматичному режимах. Тривалість світіння кожного ліхтаря в автоматичному режимі може налаштовувати користувач. Під час роботи світлофора засоби налаштування приховані.*

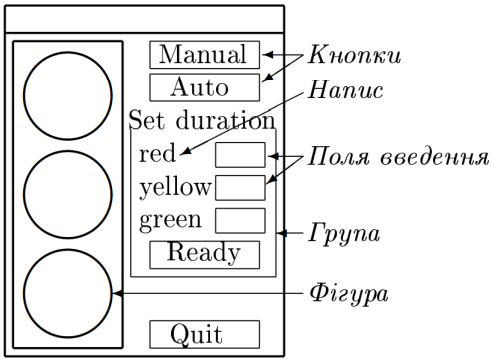

Рис. 1. Схема застосунку

Таке завдання справді охоплює різні аспекти програмування віконних застосунків. Є нагода оголосити класи, відповідальні за модель даних задачі, використати стандартні компоненти для взаємодії з користувачем, графічні примітиви, передбачити різні способи керування програмою: командами користувача та плином часу (в автоматичному режимі).

Перелічимо компоненти, потрібні для побудови застосунку

- Графічні елементи для відображення ліхтарів і корпусу світлофора.
- Кнопки для ручного перемикання світла, для переходу в режим налаштування тривалостей, для завершення роботи тощо.
- Пояснювальні написи і поля для введення.

- Засоби групування візуальних елементів і взаємного розташування у вікні застосунку.
- Таймер для автоматичного перемикання.

Схематично графічний інтерфейс застосунку зображено на рис. 1.

Але розпочнемо роботу не з інтерфейсу користувача, а з проєктування класів, відповідальних за модель. Нагадаємо, що наша мета – порівняти розробку у Pharo з роботою у Visual Studio з використанням мови C#, тому тексти програм далі будуть двома мовами.

4. Модель

Кожен ліхтар має низку властивостей: розміри, координати, колір, тривалість світіння. Він може вмикатися та вимикатися. Усе це свідчить про те, що ми маємо справу з об'єктом реального світу, стан і поведінку якого можна моделювати окремим класом, наприклад, *Lamp*. Очевидно, що розміри та координати належать до властивостей графічного компонента, відповідального за зображення ліхтаря, а екземплярові *Lamp* достатньо зберігати посилання на нього. У середовищі Pharo таким компонентом може бути екземпляр *CircleMorph*, а в програмі мовою C# – екземпляр класу System.Windows.Shapes.Ellipse.

Щоб створити новий клас у Pharo, достатньо виконати в системному оглядачі класів одне ключове повідомлення:

```
Object subclass: #Lamp
   instanceVariableNames: 'color view duration switchedOn'
   classVariableNames: '' package: 'TrafficLightsProject'
```

Клас *Object* звернеться до компілятора з проханням утворити йому підклас *Lamp* (ім'я підкласу задано символом) з чотирма полями, чиї імена записано в рядку, другому параметрі повідомлення, та віднести підклас до пакета *TrafficLightsProject*, який створиться автоматично.

Після виконання такого повідомлення в системі з'явиться (поки що порожній) клас, з яким уже можна працювати: створювати його екземпляри, інспектувати їх тощо. Система Pharo пристосована до поступального стилю програмування. Програміст пише код невеликими частинами, оголошує зазвичай один-два методи й одразу може випробувати їх в дії, написати модульні тести й тільки тоді переходити до наступних частин програми. Через цю особливість ви рідко бачите повне оголошення класу: системний оглядач відображає у своєму вікні редагування тільки один вибраний вами метод. Заради економії місця ми наведемо нижче всі методи класу *Lamp* підряд, хоча на практиці писали і випробовували їх по одному. Щоб зазначити належність методу, в літературі використовують позначення `Class >> method`, де знак `>>` є лише позначенням, а не частиною синтаксису. У подвійних лапках у Pharo записують коментарі.

```
Lamp class >> new: aColor for: time on: aMorph
      "створення екземпляра заданого кольору та тривалості світіння"
   ^ self new
  color: aColor; duration: time; view: aMorph.
Lamp >> switchOn
   switchedOn := true. view color: color.
```

```
Lamp >> switchOff
   switchedOn := false. view color: Color lightGray lighter.
      "Методи доступу до змінних екземпляра"
Lamp >> color: anObject
   color := anObject.
Lamp >> isSwitchedOn
   ^ switchedOn.
Lamp >> duration
   ^ duration.
Lamp >> duration: anNumber
   duration := anNumber.
Lamp >> view: anObject
   view := anObject.
```

Відповідне оголошення класу мовою C# наведене нижче. Згідно з вимогами C# все оголошення записане неперервно, клас зачислимо до певного простору імен. Текст оголошення дещо компактніший за попередній завдяки спрощеному синтаксису властивостей і наявності доступу до змінних екземпляра у конструктора класу.

```
namespace WpfCrossLightModel
{
    class Lamp
    {   private Ellipse view;
        private SolidColorBrush brush;
        private bool switchedOn;
        public int Duration { get; set; }
        public Lamp(Ellipse ellipse, SolidColorBrush color, int duration)
        {
            view = ellipse; brush = color;
            Duration = duration; switchedOn = false;
        }
        public bool IsSwitchedOn => switchedOn;
        public Lamp SwitchOn()
        {
            switchedOn = true; view.Fill = brush;
            return this;
        }
        public Lamp SwitchOff()
        {
            switchedOn = false; view.Fill = Brushes.LightGray;
            return this;
        }
    }
}
```

Легко знайти аналогії в наведених оголошеннях. Друге з них багатьом видасться звичним і зрозумілим завдяки C-подібному синтаксису. Проте важко буде сперечатися з тим фактом, що мова C# використовує значно більше понять і синтаксичних конструкцій.

Роботою ліхтарів керує окремий клас – світлофор. Він має зберігати колекцію ліхтарів, пам'ятати номер увімкнутого, вміти перемикати ліхтарі по одному або автоматично. Створимо клас *TrafficLights* у Pharo за допомогою вже згаданого ключового повідомлення.

```
Object subclass: #TrafficLights
   instanceVariableNames: 'lamps activeLamp view mustRun stopwatch'
   classVariableNames: '' package: 'TrafficLightsProject'
```

Тут оголошено такі змінні екземпляра: масив ліхтарів; номер увімкнутого ліхтаря; посилання на екземпляр *BorderedMorph*, який зображає світлофор на екрані та містить вкладені *CircleMorph* ліхтарів; ознака того, чи увімкнуто автоматичний режим; секундомір.

Певну цікавість викликає алгоритм перемикання ліхтарів: який колір потрібно увімкнути після жовтого – зелений чи червоний? Щоб не шукати відповідь на це запитання, збережемо в масиві *два* посилання на жовтий ліхтар. Тоді зможемо завжди перемикати на наступний ліхтар (у припущенні, що після останнього знову йде перший). Методи *TrafficLights* оголошують і випробовують у системі по одному, але ми наведемо їх тут усі підряд.

```
TrafficLights class >> newWith: aMorph
   " Створення світлофора з заданим графічним зображенням "
   ^ self basicNew view: aMorph; initialize.
TrafficLights >> view: aMorph
view := aMorph.
TrafficLights >> initialize
   " Для створення колекції ліхтарів використано динамічний масив.
     Увімкнуто жовтий колір.
     Морф світлофора містить вкладені морфи ліхтарів. "
   lamps := { (Lamp new: Color red for: 5 on: (view submorphs at: 1)).
              (Lamp new: Color yellow for: 1 on: (view submorphs at: 2)).
              (Lamp new: Color green for: 5 on: (view submorphs at: 3)).
              nil }.
   lamps at: 4 put: (lamps at: 2). " додаткове посилання на жовтий ліхтар "
   activeLamp := 2.
   (lamps at: activeLamp) switchOn.
   mustRun := false.
   stopwatch := Stopwatch new.
TrafficLights >> run
   " Метод відстежує значення секундоміра, щоби вчасно перемкнути ліхтар.
     Аби не блокувати вікно застосунку, виконання запускаємо в окремому потоці.
     Для цього достатньо надіслати блокові коду повідомлення fork."
   mustRun := true.
   [ [ mustRun ] whileTrue: [ " увімкнуто автоматичний режим "
      stopwatch activate.
      [ stopwatch duration seconds < (lamps at: activeLamp) duration ]
      whileTrue: [ "do nothing, just wait" ].
      stopwatch reset.
      self changeLamp ] ] fork.
TrafficLights >> stop
   " Сигналізує про потребу зупинити відлік часу "
   mustRun := false.
TrafficLights >> changeLamp
```

```
   " Перемикання на наступний ліхтар (по колу). "
(lamps at: activeLamp) switchOff.
activeLamp := activeLamp % 4 + 1.
(lamps at: activeLamp) switchOn.
TrafficLights >> setDurations: anArray
   " Зберігає тривалості світіння, задані користувачем. "
   1 to: 3 do: [ :i | (lamps at: i) duration: (anArray at: i) ].
```

Порівняйте з оголошенням такого ж класу мовою C#.

```
namespace WpfCrossLightModel
{
    class TrafficLights
    {
        private const int countOfLamps = 4;
        private Lamp[] lamps;
        private int activeLamp;
        private DispatcherTimer timer;

        public TrafficLights(Ellipse[] views, int activeNo = 1)
        {
            activeLamp = activeNo;
            lamps = new Lamp[countOfLamps]
            {
                new Lamp(views[0], Brushes.Red, 5),
                new Lamp(views[1], Brushes.Yellow, 2),
                new Lamp(views[2], Brushes.Green, 5),
                null
            };
            lamps[3] = lamps[1];
            lamps[activeLamp].SwitchOn();
            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromSeconds(GetDuration());
            timer.Tick += timer_Tick;
        }
        public void ChangeLamp()
        {
            lamps[activeLamp].SwitchOff();
            activeLamp = ++activeLamp % countOfLamps;
            lamps[activeLamp].SwitchOn();
        }
        public int GetDuration() => lamps[activeLamp].Duration;
        public void SetDurations(int[] durations)
        {
            for (int i = 0; i < countOfLamps - 1; ++i)
               lamps[i].Duration = durations[i];
        }
        public void Run() => timer.Start();
        public void Stop() => timer.Stop();
        private void timer_Tick(object sender, EventArgs e)
```

```
        {
            ChangeLamp();
            timer.Interval = TimeSpan.FromSeconds(GetDuration());
        }
    }
}
```

Графічні зображення (фігури) у застосунках WPF мовою C# не можуть утворювати ієрархічні групи, тому конструктор отримує масив кругів (а не одне посилання на вигляд, як у випадку Pharo). Дещо простіше виглядає відстеження часу завдяки використанню *DispatcherTimer*.

5. Графічний інтерфейс користувача

Віконний інтерфейс застосунку мовою Pharo побудуємо за допомогою засобів бібліотеки Spec [5], а мовою C# напишемо WPF-застосунок [6]. Через брак місця ми не зможемо навести повні тексти створених проєктів. Їх можна знайти в [7], [8]. Тут обговоримо головні особливості й відмінності використаних інструментів.

Взаємодію з користувачем у віконному застосунку Pharo будують за допомогою підкласів класу *SpPresener*, класу подання. Бібліотека Spec містить оголошення класів подання для графічних елементів і для багатьох поширених елементів керування: кнопок, рядків уведення, списків, меню тощо. Особливість подання у тому, що без жодних зусиль можна відкрити вікно, яке містить це подання, і випробувати його в дії. Отримати "кнопку на все вікно" можна за допомогою одного унарного повідомлення. Подання добре пристосовані для поєднання в композитні подання. Для керуванням взаємним розташуванням візуальних елементів у композиті використовують стандартні компоненти-макети, які автоматично підлаштовують координати й розміри вкладених компонент до розмірів вікна. За допомогою таких макетів програміст будує розмітку вікна. Саме тому ми обрали для порівняння WPF як найбільш схожу альтернативу.

У своєму проєкті ми використали *SpBoxLayout*, певний аналог *StackPanel* у WPF; *SpGridLayout*, аналог *Grid*, та *SpMorphLayout*, який не має аналога у WPF. Є ще й інші, унікальні компоненти-макети, про які можна довідатися в [4].

Ще одна характерна риса компонента-подання – придатність для повторного використання. Можна сказати, що розробка віконного інтерфейсу у Pharo "компонентно-орієнтована". Будь-яку дещо самостійну частину майбутнього вікна можна спроєктувати за допомогою окремого класу подання без жодних додаткових зусиль. Такий клас можна випробовувати в окремому (своєму) вікні незалежно від інших частин інтерфейсу. Згодом його можна повторно використовувати в інших проєктах. Мовою C# також можна оголошувати візуальні компоненти користувача: і в середовищі Windows Forms, і в Windows Presentation Foundation. Але для цього доведеться створити окремий проєкт, виготовити збірку-бібліотеку та приєднати її до основного проєкту – затратити доволі додаткових зусиль.

На рис. 1 позначено групу елементів керування, які відповідають за введення тривалостей світіння. Тут є написи, рядки введення, кнопка. У своєму проєкті ми виготовили її як окреме подання, клас *InputPanelPresenter*. Згодом використали його екземпляр поруч зі стандартними для побудови цілого вікна класу *TrafficPresenter*.

Для виготовлення подання віконного застосунку потрібно виконати кілька прос-

тих кроків.

- Створити підклас класу *SpPresenter*, оголосити змінними екземпляра майбутні вкладені компоненти.
- Визначити метод *initializePresenters*, у тілі якого створюють вкладені компоненти, налаштовують їхній зовнішній вигляд. Якщо компонент багато, то створення можна розподілити по кількох допоміжних методах, які викликають у тілі *initializePresenters*.
- Визначити метод *defaultLayout*, у тілі якого описують розмітку вікна за допомогою створення екземплярів компонент-макетів і їхніх комбінацій. Вкладені візуальні компоненти розташовують усередині макета.
- Поведінку візуальних компонент задають у методі *connectPresenters*, наприклад, реакцію кнопки на клацання описують тут.
- Додатковий метод *initializeWindow* дає змогу налаштувати початковий розмір подання та заголовок його вікна.

Продемонструємо на прикладі класу *InputPanelPresenter* як це виглядає на практиці, а тоді порівняємо з відповідною XAML-розміткою застосунку WPF.

```
SpPresenter subclass: #InputPanelPresenter
   instanceVariableNames: 'redInput yellowInput greenInput readyButton'
   classVariableNames: ''
   package: 'TrafficLightsProject'.
InputPanelPresenter >> initializePresenters
   redInput := self newNumberInput rangeMinimum: 1 maximum: 10;
                       number: 5; yourself.
   yellowInput := self newNumberInput rangeMinimum: 1 maximum: 10;
                       number: 2; yourself.
   greenInput := self newNumberInput rangeMinimum: 1 maximum: 10;
                       number: 5; yourself.
   readyButton := self newButton label: 'Ready';
                       icon: (self iconNamed: #smallOk); yourself.
InputPanelPresenter >> defaultLayout
   ^ SpGridLayout new
      add: 'Set durations of ligths' at: 1 @ 1 span: 2 @ 1;
      add: 'red' at: 1 @ 2;    add: redInput at: 2 @ 2;
      add: 'yellow' at: 1 @ 3; add: yellowInput at: 2 @ 3;
      add: 'green' at: 1 @ 4;  add: greenInput at: 2 @ 4;
      add: readyButton at: 2 @ 5 span: 2 @ 1;
      yourself.
InputPanelPresenter >> action: aBlock
   " Поведінку кнопки задає користувач класу "
   readyButton action: aBlock! !
InputPanelPresenter >> initializeWindow: aWindowPresenter
   aWindowPresenter
      title: 'Set durations of ligths'; initialExtent: 200 @ 200.
```

Такий код добре структурований, легко пишеться, легко читається. Наприклад, *SpGridLayout* автоматично додає рядки і стовпці у відповідь на повідомлення *add:at:*. Кнопку легко оздобити піктограмою, бо середовище надає багато готових зображень.

Розглянемо фрагмент XAML-розмітки такої ж групи компонент у альтернативному застосунку.

```
<GroupBox Name="DurationBox" Header="Set durations of lights"
 Height="125" Width="140" Margin="0,10,0,0">
   <Canvas>
      <Label Canvas.Top="0" Canvas.Left="5" Content="red"/>
      <Label Canvas.Top="20" Canvas.Left="5" Content="yellow"/>
      <Label Canvas.Top="40" Canvas.Left="5" Content="green"/>
      <TextBox Name="RedTextBox" Canvas.Top="4" Canvas.Left="65"
       Width="55" Text="5" PreviewTextInput="NumberValidationTextBox"/>
      <TextBox Name="YellowTextBox" Canvas.Top="24" Canvas.Left="65"
       Width="55" Text="2" PreviewTextInput="NumberValidationTextBox"/>
      <TextBox Name="GreenTextBox" Canvas.Top="44" Canvas.Left="65"
       Width="55" Text="5" PreviewTextInput="NumberValidationTextBox"/>
      <Button Name="ReadyButton" Canvas.Top="67" Canvas.Left="30"
       Width="80" Height="30" Content="Ready" Click="ReadyButton_Click"/>
   </Canvas>
</GroupBox>
```

Перевага бібліотеки класів WPF для цього прикладу в тому, що компонент *GroupBox* зображає рамку навколо себе (і вкладених компонент). З іншим є певні проблеми. Всередині *GroupBox* не вдалося помістити макет *Grid*, який став би в нагоді для швидкого заповнення компонентами рядків і стовпців уявної таблиці, бо до *Grid* потрібно додавати визначення рядків і стовпців, а *GroupBox* такого "не розуміє". Тому довелося використати *Canvas* і підбирати абсолютні координати та розміри візуальних компонент. Бібліотека WPF не надає спеціалізованого компонента для введення чисел, тому довелося використовувати *TextBox* і доповнювати його поведінку перевіркою на правильність введеного.

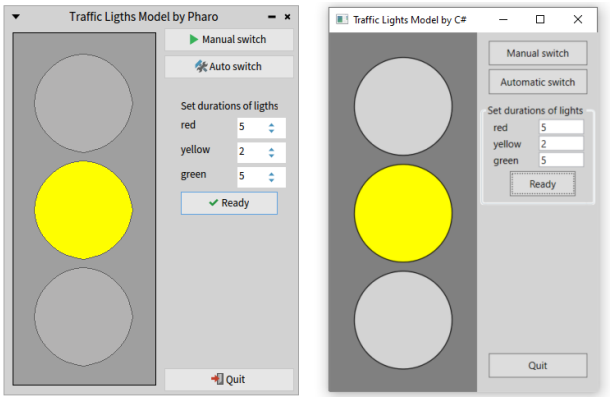

Рис. 2. Вигляд створених застосунків: у Pharo – ліворуч, C# – праворуч

Недоліком можна вважати "монолітність" файлу розмітки. В ньому міститься опис цілого вікна, усіх малих і великих його частин. Поруч в одному рядку розмітки розташовано оголошення типу компонента, імені змінної з посиланням на нього, налаштування значень, які задають його зовнішній вигляд, задання методів,

прив'язування до батьківського компонента – дуже багато всього. Звісно, Visual Studio полегшує процес розробки за допомогою відступів тексту, синтаксичного підсвічування та синхронного відображення у вікні дизайнера розміченого вікна. Проте варто трапитися одній помилці в тексті розмітки, як зображення вікна зникає. Таких недоліків позбавлений модульний код у середовищі Pharo. Кожну найменшу частину подання можна програмувати та перевіряти незалежно.

Нагадаємо, що повний текст програм на Pharo і на C# можна завантажити зі сховищ GitHub, відповідно, [7] і [8]. Зовнішній вигляд готових застосунків зображено на рис. 2.

6. Висновки

Pharo – вільно поширювана система програмування з відкритим кодом, яка може скласти гідну конкуренцію популярним нині мовам і середовищам програмування. Мова Pharo є сучасним втіленням і творчим розвитком мови Smalltalk, яка мала значний вплив на шляхи розвитку та можливості сучасного програмного забезпечення. Синтаксис мови Pharo у рази простіший за синтаксис C++, C# чи Java, а можливості надає такі самі, чи й більші. Адже приватність пам'яті на рівні екземпляра не підтримує жодна з перелічених мов. Середовище Pharo пристосоване до інкрементного процесу побудови програм. Розробник створює код невеликими частинами, які відразу можна тестувати та використовувати, а динамічна типізація та взаємодія виключно через надсилання повідомлень робить можливою зміну програмного коду під час його виконання.

Бібліотека Spec надає багатий арсенал засобів для швидкої побудови розвинутого графічного інтерфейсу віконних застосунків. Програмний код добре структурований, придатний для повторного використання без додаткових зусиль. Здатність кожного подання – стандартного чи написаного розробником – відображатися і функціонувати в окремому вікні тільки пришвидшує процес розробки та налагодження. Динамічна природа Pharo дає змогу легко змінювати зовнішній вигляд програми під час її виконання. Достатньо змінити використаний компонент-макет на інший, додати чи вилучити візуальний компонент через надсилання відповідних повідомлень, і застосунок миттєво пристосується до потреб користувача. Докладніше такі можливості описані в [5].

Список використаної літератури

1. 2023 State of Open Source Report [Електронний ресурс] / OpenLogic // Free Open Source Report, 2023. – Режим доступу: https://www.openlogic.com/resources/2023-state-open-source-report – Назва з екрану.
2. Pharo [Електронний ресурс] / Pharo Consortium // Official Page of Pharo, 2023. – Режим доступу: https://pharo.org/ – Назва з екрану.
3. Pharo consortium portal [Електронний ресурс] / Pharo Consortium // Official Page of Pharo Consortium, 2023. – Режим доступу: https://consortium.pharo.org/ – Назва з екрану.
4. Стефан Дюкас Pharo 9 на прикладах / С. Дюкас, Дж. Ракіч [та ін.]; пер. з англ. С. Ярошко. – Львів : ЛНУ ім. Івана Франка, 2022. – 270 с. [Електронне видання] – Режим доступу: http://books.pharo.org/pharo-by-example9/ – Назва з екрану.
5. Johan Fabry The Spec UI Framework / Johan Fabry, Stéphane Ducasse – Square Bracket Association, 2017. – 80 p. [Електронне видання] – Режим доступу: http://books.pharo.org/spec-tutorial/ – Назва з екрану.

6. Desktop Guide [Електронний ресурс] / Microsoft Corporation // Microsoft Learn, 2023. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0 – Назва з екрану.
7. TrafficLightsBySpec [Електронний ресурс] / Сергій Ярошко // GitHub, 2023. – Режим доступу: https://github.com/LNUitTutor/TrafficLightsBySpec – Назва з екрану.
8. WpfCrossLightModel [Електронний ресурс] / Сергій Ярошко // GitHub, 2023. – Режим доступу: https://github.com/LNUitTutor/WpfCrossLightModel – Назва з екрану.

Стаття: надійшла до редколегії 18.09.2023
доопрацьована 04.10.2023
прийнята до друку 25.10.2023

BUILDING DESKTOP APPLICATIONS IN THE PHARO ENVIRONMENT USING THE SPEC LIBRARY

S. Yaroshko, O. Yaroshko

Ivan Franko National University of Lviv,
1, Universytetska str., 79000, Lviv, Ukraine
e-mail: serhiy.yaroshko@lnu.edu.ua

Pharo is a modern object oriented dynamically typed programming language. It is also an interactive environment, equipped with excellent tools for "live" programming, which allows to easily create, debug and share program code, as well as change it during execution. Pharo is a cross-platform system, which works perfectly well in multiple operating systems: OS X, Windows, Linux, Android, iOS та Raspberry Pi. Pharo is a multi-purpose language. With the usage of available libraries, it is possible to create desktop and web applications, interact with databases, solve the problems of artificial intelligence, etc.

This article, based on one training example, describes the process of creating a window application in Pharo using the Spec library. It also provides a comparison with the possibilities of Windows Presentation Foundation (WPF), programming language C# and development environment Microsoft Visual Studio for the creation of a similar window application. It presents the advantages and particularities of Pharo. The authors' intention is to popularize Pharo with this article.

The training example consists in the creation of a window application which models the street traffic light operation. It presents the lights of red, yellow and green colors, allows to switch them in manual and automatic modes. The duration of each lamp glow in automatic mode can be modified by a user. While the traffic light is working, the modification tools should be hidden. Such a task allows to describe quite some tools: declaration and implementation of classes responsible for the data model of the problem, usage of standard components for user interaction, graphic primitives, realization of different types of control of the application: by user commands and by the time flow (in automatic mode). A high level representation of graphic interface of the application is given at the scheme 1. In order to better understand the presented code snippets, check the syntax of Pharo explained in http://books.pharo.org/pharo-by-example9.

The two following classes are used for the data model: *Lamp* saves the color, size, coordinates and lamp glow duration, and *TrafficLights* contains a collection of lanterns and implements the algorithm of switching lights. The application window in Pharo is described with two entities: *InputPanelPresenter* which represents the lamp glow duration configuration panel, and which is part of the window implemented via *TrafficLightsPresenter*. The complete program code in Pharo and in C# can be downloaded from GitHub repositories *TrafficLightsBySpec* and *WpfCrossLightModel* respectively. General appearance of both applications is presented.

Pharo environment is designed for an incremental process of software application creation. Developer creates the programming code in smaller parts, which can immediately be tested and used, and dynamic typization and interaction via sending messages make it possible to modify the programming code during its execution. Spec library provides an

extensive set of tools to quickly and easily construct an advanced graphical interface of windows applications. The program code is well structured and available for reuse without any additional efforts. The ability of any layout – standard or implemented by a developer – to be displayed and executed in a separate window speeds up the complete development and debugging process.

Key words: Pharo, Spec, WPF, C#, desktop application, graphical user interface, visual component, class, object.