УДК 004.4:418(045)

Фабричев В. А., д-р техн. наук,
Скрипка В. М.

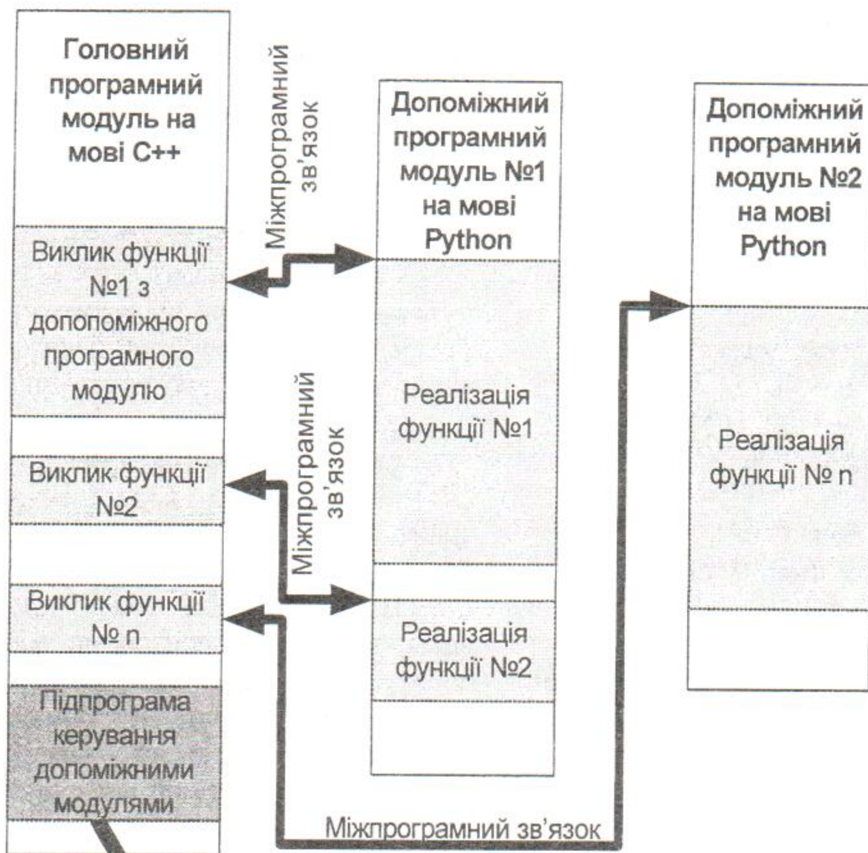
ВИКОРИСТАННЯ СИМБІОЗУ МОВ ПРОГРАМУВАННЯ ДЛЯ ШВИДКОГО ДОСЯГНЕННЯ МЕТИ ЕКСПЕРИМЕНТІВ

Інститут комп'ютерних технологій Національного авіаційного університету

В статті описано взаємодію мов програмування Python та С++. Визначено спосіб розробки програмного забезпечення на різних мовах програмування та інтерфейси взаємодії

Для вирішення різноманітних задач прикладного програмування існує велика кількість мов програмування. Деякі з цих мов є орієнтованими на вирішення вузько спеціалізованих задач, але є й універсальні мови програмування. Одна з найбільш потужних серед універсальних мов програмування це мова *C++*. На ній можливо писати з великим успіхом як програми керування апаратними ресурсами комп'ютерів, так й програми прикладного характеру. Але ціною універсальності є складність програмування. Крім того, сучасні системи не повинні перенавантажуватись повністю при заміні модулю, що не є життєво необхідним для програми.

Вихід з цієї ситуації – використання універсальної мови програмування, такої як *C++*, для створення керівного модулю програми та допоміжних модулів, що написані на інших, спеціалізованих мовах. Спосіб взаємодії наведений на рис. 1. Мови програмування, що мають бути використані в допоміжних модулях, повинні мати інтерфейс об'єднання з основною мовою. Крім того, бажано, щоб допоміжні модулі використовували сучасні методи роботи з апаратними ресурсами, тобто багатозадачність, оптимізацію швидкості та оперативної пам'яті, тощо. Таких найбільш потужних допоміжних мов є декілька – *Python*, *Perl*, *Lua* [4]. В результаті аналізу кількості та якості документації, швидкості реалізації алгоритмів, наявності програмних візуальних засобів розробки програмного коду, швидкості виконання готових програмних засобів, рівня взаємної інтеграції з С++, кількості та якості допоміжних бібліотек для цих програмних засобів та ряду інших факторів, лідером став *Python*. Це не означає, що він є безумовним лідером, але для кожного проекту є найголовніший фактор. Якщо цей фактор швидкість кінцевого виконання програми – то *Python* є найбільш швидким серед скриптових мов. тобто таких, що не потребують компіляції. Швидкість мови програмування Python частково зумовлена тим, що перед виконанням виконується автоматична компіляція коду в байт код, який оптимізовано під конкретну апаратну платформу. Таким чином, подальше використання допоміжного засобу є більш швидким, ніж перше, та на рівень вище за швидкість модулів, що використовують інші скриптові рішення. В мові програмування *Python* можливо використання технології, роботи з класами, методами та змінними, що працюють в модулі, який написано на *C++*, та навпаки. Це є великою перевагою і використовується все частіше. Скриптові мови програмування майже завжди легше від мов програмування, що мають компілюватись. Крім того, процес перезавантаження цих програм значно легший, ніж програмних кодів з процесом компіляції. Для збільшення швидкості розробки програмного забезпечення було запропоновано мову програмування *Python*. За декілька років він став ще більш популярним та потужним завдяки підтримці цієї мови програмування як і старими розробниками, так і новими. Зокрема, *Apache Software Foundation* підтримує *Python* в своєму *HTTP* сервері за допомогою *mod_python*. Подібний шлях був і в мови програмування *PHP*.



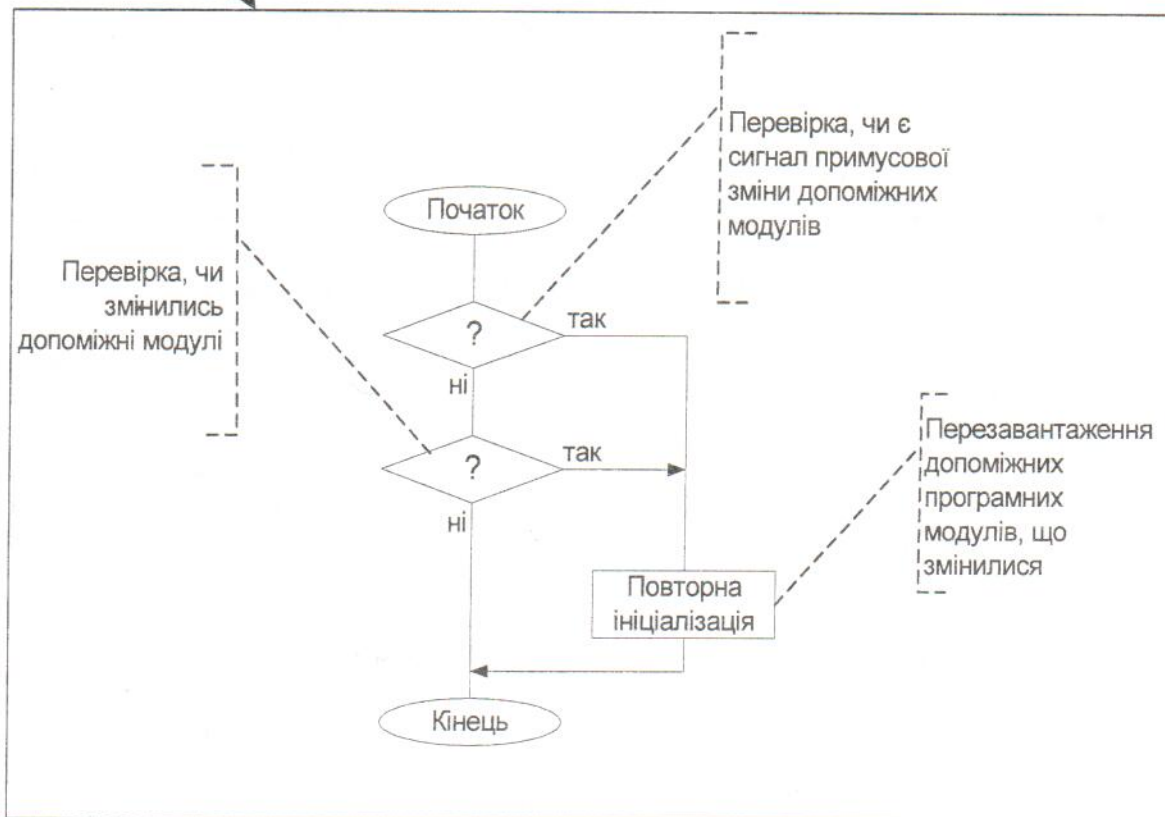


Рис. 1. Використання симбіозу мов програмування

Наведемо код, що дозволяє виконувати *Python* скрипти з модулів, що написані на *C++*.

```
#include "pyengine.h"
int
main(int argc, char *argv[])
{
        int i;
        PyEngine peng("source");
        PyObject *po=NULL;
        po=peng.PyOpcode(10,2,3,2);
}
```

Метод *PyOpcode* має опис *PyEngine::PyOpcode(uint32_t opcode,int argcount,...)*

та формує дані для виклику функції з модулю "*source.py*". В ньому існує функція , яка визначена як:

```
def f0000000010(a,b):
        print "This is function 10"
        c = 0
        for i in range(0, a):
                c = c + b
        return c
```

Виклик здійснюється за допомогою імені, сформованого допоміжним методом

```
PyEngine::GenPyName(uint32_t opcode)
{
        string str="f";
        str+=toFilledString(opcode,10,'0');
        return str.c_str();
}
```

Метод *toFilledString* сформує з числа строку з заданою кількістю визначених символів заповнення. Тобто, *toFilledString(opcode,10,'0')* сформує строку "1000000000".

Змінні перетворюються в нетипізовані змінні *Python*:

```
pValue=
PyLong_FromUnsignedLongLong(va_arg(
ap,uint32_t));
PyTuple_SetItem(pArgs, i, pValue);
```

Наприкінці йде виклик вже сформованого об'єкту:

```
pValue = PyObject_CallObject(pFunc,
pArgs);
```

Подібним чином *pValue* перетворюються в змінні мови програмування *C++*. Таким чином, основні задачі можуть бути вирішені на мові програмування *C++* з подальшим збереженням результатів в базі даних за допомогою *Python*, так як на цій мові, порівняно з *C++*, в 3-5 разів потрібен менший код для досягнення результату, а швидкість майже така, як в *C++*. Швидкість збільшена в *Python*, порівняно з іншими скриптовими мовами, за допомогою формування проміжного коду, такого як в *Java*, при першому запуску програми та подальшого використання поміжного коду доти, поки не буде змінено код програми [2]. Крім того, *Python* має керовану багатопоточність, що також дає значну перевагу між іншими скриптовими мовами програмування[2, 5].

Висновок

Сучасна система керування доступом до розгалужених кластеризованих ресурсів є досить складною й у кожному конкретному випадку має свій набір можливостей, котрі відповідають потребам замовників. Але, як показали дослідження майже завжди технічне завдання корегується в процесі розробки і виникає потреба в більшому наборі можливостей, ніж існує. Тому, в процесі проектування потрібно передбачити модульність системи в цілому та як можна зменшити залежність модулів один від одного [3, 5].

Завжди потрібно знати як поведе себе програмний засіб при великих навантаженнях та короткочасному ймовірному збої. Для відтворення подібної тестової моделі потрібно знати особливості операційної системи, де буде виконано програмний засіб, та способи виклику збою. У сучасних операційних системах такі ситуації швидко виправляються розробниками

операційної системи, тому необхідно самому створити допоміжні програмні засоби, що будуть імітувати вторгнення в операційну систему чи збій. Наведений приклад програми [7] є одним з можливих інсталяторів коду імітації збою в операційну систему. Так цей програмний засіб був використаний для тестування обчислювального кластеру на предмет усунення процесів – пожирачів процесорного часу. Він створював процеси, що містили нескінченні цикли та інші засоби, що заважали нормальному функціонуванню програмного забезпечення.

Програмне забезпечення, що розробляється, не завжди виконується в тому самому середовищі, де буде виконуватись остаточна версія програмного забезпечення. Таким чином неможливо гарантувати правильного виконання програмного коду без тестування в середовищі умовного замовника. Для зменшення можливості збою, системи тестування мають бути незалежними від середовища та скрізь виконуватись за однаковим принципом дії. Це приводить до використання таких мов як *Java* [6], *Python*. Для обробки результатів даних найбільш підходить *Java* [6]. Для того, щоб не перераховувати інтерфейсів таких програмних засобів, що були створені на мові *C++*, пропонується портування коду за визначеною в [1]. Більшість комп'ютерів мають з'єднання з мережею. Це, в свою чергу, дає можливість використання допоміжного програмного забезпечення для паралельних обчислень. Прикладом може бути бібліотека та керуючі програми *OpenMPI* чи подібне. Мови *C++*, *Java* та *Python* мають модулі *MPI*, що сумісні між собою. Цей спосіб взаємодії є більш універсальним за інші, але потребує більш опрацьованої архітектури програмного забезпечення, так як використання різних мов програмування на різних комп'ютерах пов'язано з використанням різної довжини змінних та інших факторів, що можуть вплинути на хід обчислень.

Список літератури

1. *Скрипка В. М., Фабричев В. А.,* Портування координатно розрахованого графічного інтерфейсу на мову програмування JAVA. – Проблеми інформатизації та управління : Зб. наук. праць. – К.: НАУ, 2005. – С. 147-150.

2. *Лутц М.*, Программирование на Python. – Пер. с англ. – С.Пб: Символ-Плюс, 2002. – 1136 с.

3. *Мейерс С.,* Эффективное использование STL. Библиотека программиста. – СПб.: Питер, 2003. – 224 с.

4. *Керниган Б., Пайк Р.,* UNIX. Программное окружение. – Пер. с англ. – С.Пб: Символ-Плюс, 2003. – 416 с.

5. *Фаулер М.,* Архитектура корпоративных программных приложений.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 544 с.

6. *Кей С. Хорстманн, Гари Корнелл.* Библиотека профессионала. Java 2. Том 2. Тонкости программирования. – М: «Вильямс», 2002. – 1120 с.

7. *Скрипка В. М., Фабричев В. А.,* Моделювання процесу вторгнення в операційну систему. – Матеріали V міжнародної науково-технічної конференції «Авіа 2003». – К.: НАУ, 2003. – С. 19.163.