ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 2, February 2019

Application Development using Kivy Framework

Aman Bhoyarkar [1], Anuja Solanki [2], Ashutosh Balbudhe [3]

U.G. Student, Information Technology, RGCER, Nagpur, Maharashtra, India[1,2,3]

Abstract: This paper focuses on Kivy Framework for application development and its implementation. The main aim is to provide sufficient information about the framework and get acquainted with the installation of the framework on different platforms, components of the framework and KV language, use and implementation of framework with Python Programming. and to serve as a practical guide for Kivy Framework.

Keywords: Kivy Framework , Python, KV Language, Graphical User Interface (GUI), Application Programming Interface (API), Windows, Raspberry Pi.

I. INTRODUCTION

Nowadays, every business organisation strives to leverage its services with mobile applications. Which permits wide coverage as well as global accessibility to the business services being offered? A number of software frameworks and libraries are distributed under different licences including proprietary, free and open source software. These software frameworks provide easy-to-use APIs (application programming interface) to the programmers, so that rapid and effective applications can be developed. With the use of mobile applications escalating each day, there is huge scope in the technology market to develop new applications for multiple uses and devices

Kivy is a cross-platform Python library, which can be used for the rapid development of applications that make use of innovative interfaces.

Kivy is a powerful library based on Python for the development of mobile applications including the natural user interface (NUI). Kivy is one of the most effective cross-platform libraries that can run on iOS, Android, Raspberry Pi, Linux, Windows, MacOS X with the distribution protocol under free and open source software. Which has been developed and is distributed by the Kivy organisation.

The features of Kivy include:

- Support for multiple inputs including the mouse, TUIO (Tangible User Interface Objects), keyboard and multi-touch.
- Powerful APIs for most mobile devices.
- One application for multiple operating systems.
- Support for networking protocols and remote login.
- Support for many widgets with multi-touch KV to design custom widgets.

II. ARCHITECTURE OF KIVY

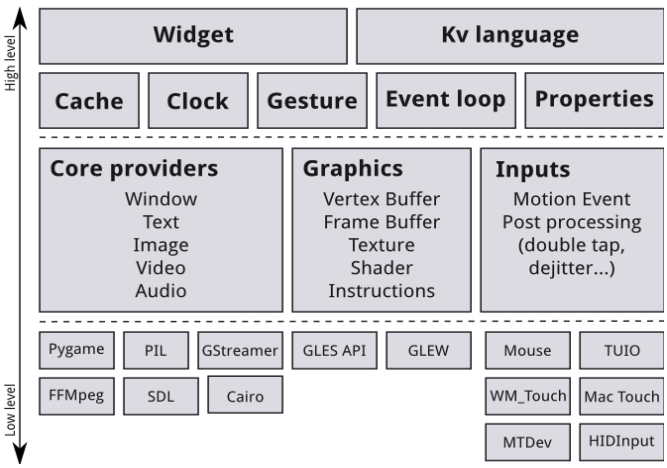

Fig 1. Architecture of Kivy

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 2, February 2019

Core Providers and Input Providers:
Kivy tries to abstract basic tasks such as opening a window, displaying images and text, playing audio, getting images from a camera, spelling correction and so on, called as core tasks. Which makes the API both easy to use and easy to extend. Most importantly, it allows us to use ‘ what we call ’ specific providers for the respective scenarios in which our application is being run. Consider an example, on OSX, Linux and Windows, there are different native APIs for the different core tasks. A piece of code that uses one of these specific APIs to communicate to the operating system on one side and to Kivy on the other (acting as an intermediate communication layer) is called a core provider. The advantage of using specialized core providers for each platform is that we can fully leverage the functionality exposed by the operating system and act as efficiently as possible. It also gives users a choice. Furthermore, by using libraries that are shipped with any one platform, it effectively reduces the size of the Kivy distribution and makes packaging easier. And also makes it easier to port Kivy to other platforms. The Android port is benefited greatly from this.

Kivy follows the same concept with input handling. The input provider is a piece of code that adds support for a specific input device, such as Apple’s track pads, TUIO or a mouse emulator. If we need to add support for a new input device, we can simply provide a new class that reads our input data from our device and transforms them into Kivy basic events.

Graphics:
Kivy’s graphics API is an abstraction of OpenGL. On the lowest level, Kivy issues hardware-accelerated drawing commands using OpenGL. Writing OpenGL code can be a bit confusing, especially to newcomers. That’s why kivy provide the graphics API that lets us draw things using simple metaphors that do not exist as such in OpenGL (e.g. Canvas, Rectangle, etc.).

All of kivy widgets themselves use this graphics API, which is implemented on the C level for performance reasons.

Another advantage of the graphics API is its ability to automatically optimize the drawing commands that the code issues. This is especially helpful if we’re not an expert at tuning OpenGL. This makes our drawing code more efficient in many cases.

Core:
The code in the core package provides commonly used features, such as:

Feature	Explanation
Clock	We can use the clock to schedule timer events. Both one-shot timers and periodic timers are supported
Cache	If we need to cache something that we often use, we can use class for that instead of writing own.
Gesture Detection	Kivy has a simple gesture recognizer that can be used to detect various kinds of strokes, such as circles or rectangles. We can train it to detect our own strokes.
Kivy Language	The kivy language is used to easily and efficiently describe user interfaces.
Properties	These are not the normal properties that we may know from python. They are kivy’s own property classes that link the widget code with the user interface description.

UIX (Widgets & Layouts):

The UIX module contains commonly used widgets and layouts that we can reuse to quickly create a user interface.

Widgets	Widgets are user interface elements that are added to the program to provide some kind of functionality. They may or may not be visible. Examples can be a file browser, buttons, sliders, lists and so on. Widgets receive MotionEvents.
Layouts	Layouts are used to arrange widgets. It is of course possible to calculate the widgets’ positions ourselves, but often it is more convenient to use one of kivy’s readymade layouts. Examples can be Grid Layouts or Box Layouts. We can also nested layouts.

Modules:
Modules are used to inject functionality into Kivy programs, even if the original author did not include it. An example can be a module that always shows the FPS of the current application and some graph depicting the FPS over time. We can also write our own modules.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 2, February 2019

Input Events(Touches):

Kivy abstracts different input types and sources such as touches, mice, TUIO or similar. These input types have in common a 2D onscreen-position with any individual input event. (There are other input devices such as accelerometers where we cannot easily find a 2D position for e.g. a tilt of our device. This kind of inputs are handled separately. In the following we describe the former types.)

All these input types are represented by instances of the Touch() class. (Note: this does not only refer to finger touches, but all the other input types as well. It is called Touch for the sake of simplicity, something that touches the user interface or our screen.) A touch instance, or object, can be in one of three states. When a touch enters one of these states, the program is informed that the event occurred. The three states in which a touch can be are:

State	Description
Down	A touch is down only once, at the very moment where it first appears.
Move	A touch can be in this state for a potentially unlimited time. A touch does not have to be in this state during its lifetime. A 'Move' happens whenever the 2D position of a touch changes.
Up	A touch goes up at most once, or never. In practice we will almost always receive an up event because nobody is going to hold a finger on the screen for all eternity, but it is not guaranteed. If we know the input sources our users will be using, we will know whether or not we can rely on this state being entered.

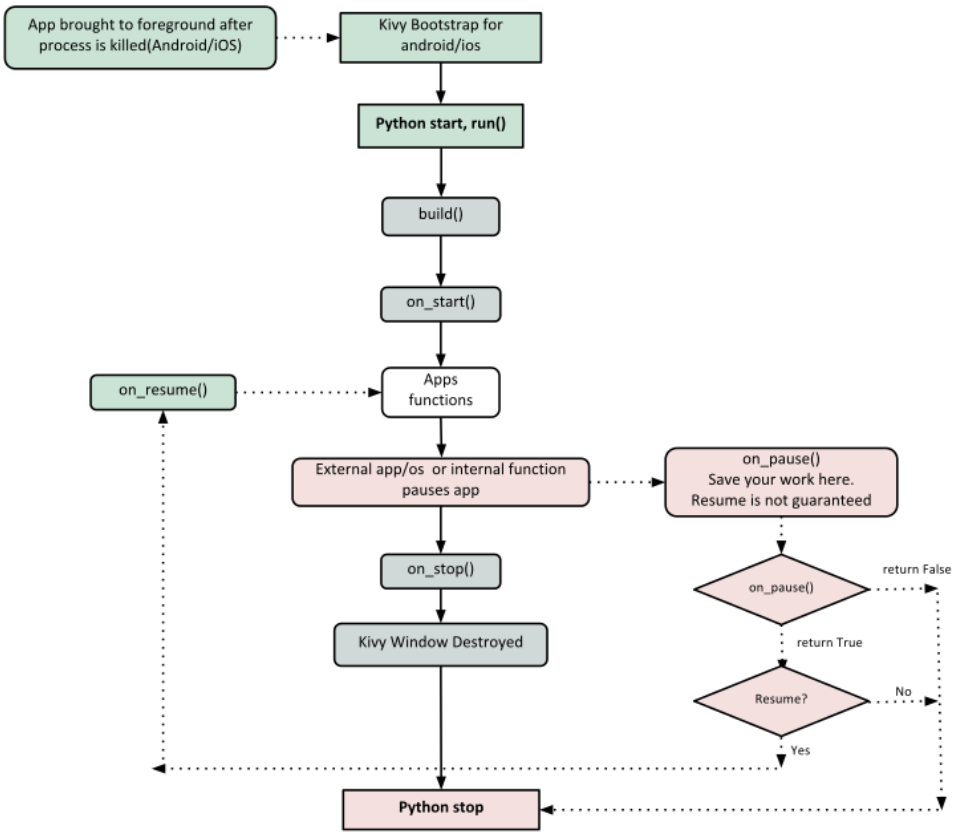

Fig 2. Kivy Application Life Cycle

III. KIVY INSTALLATION

Installation on Windows:

For installation of kivy, Python is necessary to be installed on windows machine. After installation of python:

1. Install the latest pip and wheel:

   python –m pip install –upgrade pip wheel setuptools

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 2, February 2019

2. Install the dependencies:

```
python –m pip install docutils pygments pypwin32 kivy.deps.sdl2 kivy.deps.glew
python –m pip install kivy.deps.gstreamer
```

for python 3.5+ we can also use angle backend instead of glew. This can be installed with:

```
python –m pip install kivy.deps.angle
```

3. Install Kivy:

```
python –m install kivy
```

Installation on Raspberry Pi:

Raspbian OS comes preloaded with python, hence we can directly begin with:

1. Install the dependencies:

```
sudo apt-get update
sudo apt-get install libsdl2-dev libsdl2-image-dev libsdl2-mixer-dev libsdl2-ttf-dev \
  pkg-config libgl1-mesa-dev libgles2-mesa-dev \
  python-setuptools libgstreamer1.0-dev git-core \
  gstreamer1.0-plugins-{bad,base,good,ugly} \
  gstreamer1.0-{omx,alsa} python-dev libmtdev-dev \
    xclip xsel
```

2. Install a new version of Cython:

```
sudo pip install -U Cython==0.28.2
```

3. Install kivy globally on your system:

```
sudo pip install git+https://github.com/kivy/kivy.git@master
```

4. Build and use kivy in place:

```
git clone https://github.com/kivy/kivy
cd kivy

make
echo "export PYTHONPATH=$(pwd):\$PYTHONPATH" >> ~/.profile
  source ~/.profile
```

IV. KV LANGUAGE

As the code grows more complex the construction of widget trees and explicit declaration of bindings, becomes complex and hard to maintain. The KV Language overcomes this drawback. The KV Language (kvlang or kivy language), allows us to create a widget tree in a declarative way to bind widget properties to each other or to callbacks in a natural manner. It allows for very fast prototyping and agile changes to our UI. It also allows to separate the logic of the application and the User Interface from each other.

Loading KV to the application:

There are two ways to load KV code into our application:

1. By name convention:

Kivy looks for a kv file with the same name as that of App class in lowercase, minus “App” if it ends with ‘App’

MyApp -> my.kv

If this file defines a *Root Widget* it will be attached to the App’s *root* attribute and used as the base of the application widget tree.

2. By Builder:

We can tell kivy to directly load a file or a string. If this string or file defines the root widget, it will be returned by the method.

```
Builder.load_file(‘path/to/file.kv’)
Or
Builder.load_string(kv_string)
```

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 2, February 2019

A Kv source constitutes of *rules*, which are used to describe the content of a Widget, you can have one *root* rule, and any number of *class* or *template* rules.

The *root* rule is declared by declaring the class of your root widget, without any indentation, followed by *:* and will be set as the *root* attribute of the App instance:

Widget:

A *class* rule, is declared by the name of a widget class between < > and followed by *:*, it defines how any instance of that class will be graphically represented:

<MyWidget>:

There are three keywords specific to Kv language:

- *app*: it always refers to the instance of your application.
- *root*: it refers to the base widget/template in the current rule
- *self*: it always refer to the current widget

To access python modules and classes from kv,

#:import name x.y.z

#:import isdir os.path.isdir
#:import np numpy
is equivalent to:

from x.y import z as name

from os.path import isdir
import numpy as np
in python

To set a global value:

#:set name value

Is equivalent to:

name = value

V. A SIMPLE EXAMPLE

We take here a simple example to create a basic application which can swap between two windows when clicked on the provided buttons.
The code below is to create a User Interface in KV Language

Trial.kv

```
ScreenManagement:
   transition: FadeTransition()
   MainScreen:
   Screen2:

<MainScreen>:
   name:"main"
   BoxLayout:
      orientation: 'vertical'
      Label:
         text:"This is Main Screen"
      Button:
         on_release: app.root.current = "scr2"
         text: "Next Screen"


<Screen2>:
   name:"scr2"
   BoxLayout:
      orientation: 'vertical'
      Label:
         text:"This is Screen 2"
```

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 2, February 2019

```
    Button:
        on_release: app.root.current="main"
        text:"Home screen"
```

The code below shows the python file for the swapping application.

TrialKivy.py

```
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager,Screen
class MainScreen(Screen):
    pass

class Screen2(Screen):
    pass

class ScreenManagement(ScreenManager):
    pass

presentation=Builder.load_file("Trial.kv")

class Trial(App):
    def build(self):
        return presentation

if __name__=="__main__":
    Trial().run()
```

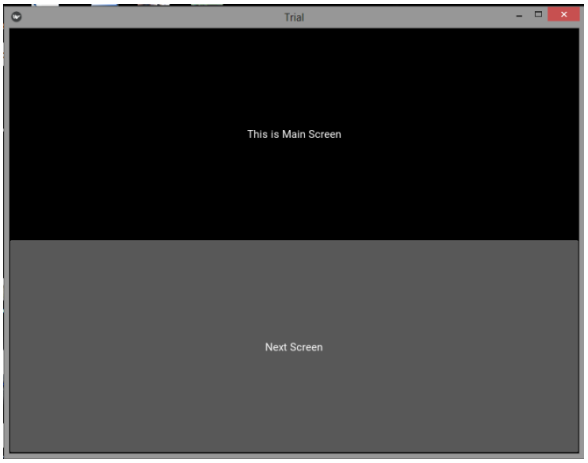

Fig3. Main Screen

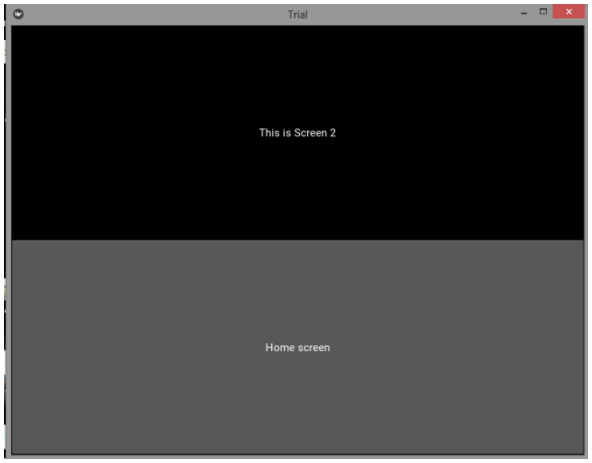

Fig4. Screen2

CONCLUSION

In this paper we tried to provide some information about the kivy framework for building cross-platform applications, along with its components, working and installation of the framework on multiple platforms. Also the KV language and its integration with python is discussed in the paper.

REFERENCES

[1] https://is.muni.cz/th/b1nab/dp.pdf
[2] https://opensourceforu.com/2016/02/developing-python-based-android-apps-using-kivy/
[3] https://media.readthedocs.org/pdf/kivy/master/kivy.pdf
[4] https://kivy.org