Python course

Objective

This course provides an introduction into advanced features of Python such as classes and objects, installing and using third party libraries (python-Qt, python-vlc, ...) and some advanced programming constructs such as state engines and how to coordinate asynchronous program flows.

The course is highly practical - we show the concepts by designing a simple media player application which implements all the concepts we teach. To make things more interesting we implement two different UI types, a local GUI and a web GUI.

I'm not the best imaginative namegiver, but for reference, let's call our wonderful product *MyPlayer*. Any ideas for a better name?

Prerequisites

Windows/Linux (advanced user)

Python (basic programming skills: Variables & statements, control structures, functions).

Basic Git usage (Setting up repo, commit, branch)

A lot of persevearance, googling and pushing through! Explanations are sometimes sparse on purpose!

Requirements

Average Laptop / Desktop machine plus peripherals.

OS: Windows (10) or (K)Ubuntu Linux 18

Python 3

Python IDE (PyClipse)

Programmer friendly editor (Windows: Notepad++, Linux: Kate or GEdit)

TortoiseGit (Windows) + deps, Just Git (Linux)

Agenda

1. Installation
    1. Python 3
    2. PyClipse
    3. python-qt
    4. python-vlc
    5. Git + friends
2. Program structure
    1. Components and their purpose
3. The GUI
    1. Visual design
    2. Architecture: Delegate pattern
    3. Logic, GUI-Only
4. The controller
    1. State machine
    2. Integrating the GUI
5. The backend
    1. The audiosystem (vlc) - Introduction
        1. Instantiating
        2. Functions
        3. Events
    2. Integrating vlc into backend
    3. Playlist
6. The controller II
    1. Integrating the backend into controller
    2. Integrating volume slider
    3. Passing runtime info to GUI (Track position/info)
7. Web GUI, visual design
8. Web service
    1. Web GUI delegate
    2. Comms channel to web client (web sockets)
9. Integrating WEB GUI
    1. The Javascript library
    2. Trying it out

Lesson 1: Installation (2019-08)

Objective: Set up the Development environment: Python+libs, git, IDE

Linux

We use KUbuntu, 64 bit, and assume a working system. For the installation you need an open Terminal (emulator). KUbuntu offers you a program for that (aptly named **`Konsole`**). There you will type everything.

VLC player

That's for your media player backend to play your muzac.

```
~:$  sudo apt install vlc
```

Qt

This is the framework which puts your local GUI on your screen.

```
~:$  sudo apt install qt4-designer
```

git

A source code management system. You need it to keep track of your project.

```
~:$  sudo apt install git
~:$  sudo apt install git-gui
~:$  sudo apt install git-man
```

Python + libs

You also need python plus some libs to bind qt and vlc into it.

```
~:$  sudo apt install python3
~:$  sudo apt install pyside-tool
~:$  sudo apt install python-qt4
~:$  sudo pip3 install python-vlc
```

Java 8

Needed for LiClipse. We'll be using the official Java8 distro from Oracle (can't be more official than that).

```
~:$  sudo add-apt-repository ppa:webupd8team/java
```

You'll see a message like this:

```
~:$   Oracle Java (JDK) Installer (automatically downloads and installs
     Oracle JDK8). There are no actual Java files in this PPA.

~:$  .... snip ....

~:$  Press [ENTER] to continue or Ctrl-c to cancel adding it.
```

Press ENTER to continue. then do

```
~:$  sudo apt update

~:$  sudo apt install oracle-java8-installer
```

Your system will download the JDK from Oracle. Accept the license agreement and see it install the stuff for you (I hope you noticed that you just signed your life away [mwahahahahaha]?)

Python IDE (LiClipse)

We'll be using LiClipse. It comes as a trial version with full functionality. After 14 days you will get a tender reminder (popup box) that you really *should* buy a license (software will continue to work though). It will cost $80, and I'd recommend buying it as it supports the developer (devs do need to eat, too - even if they say that their food is code).

Head over to https://www.liclipse.com/download.html. You'll probably have a 64 bit version of KUbuntu, so download the 64 bit version. You will get a file with the ending **`.tar.gz`** (e.g. **`liclipse_5.2.4_linux.gtk.x86_64.tar.gz`**). For your info - that's a compressed tar archive.

Once the file has downloaded, make yourself a directory **`/home/your-user-name/bin/liclipse`** and copy the file into that one. Then

- open the folder in Dolphin
- right click it
- choose `Extract -> Extract archive here, autodetect subfolder`

This will create another sub folder. To start LiClipse **`cd`** into that sub folder and execute **`LiClipse`**. (Hint) - you can make that a menu item in your KDE startup menu.

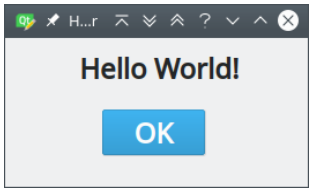
⇒ LiClipse comes with git support inbuilt so you can use git from inside your dev environment. Outside of LiClipse you can use the TortoiseGit features directly inside the Windows explorer, via right click on a (repository) folder.

Windows (10)

Whilst we could install everything directly into Windows there's a better way by setting up a virtual machine with a Kubuntu guest and then to use that as development machine. For the VM environment you can use VMWare (expensive, but if you have it, why not) or Virtualbox. With respect to setting up a VM and installing Linux, this would exceed the scope of this guide. Suffice to say that you should give the VM a harddisk with at least 30 GB and at least 4GB of memory.

Also, the VM guest (i.e. VM thing you set up) needs a working connection to the internet and a working audio output. When you have the base system working, follow the Linux part of this chapter (above) to get your dev environment ready.

Task: Using `QtDesigner` and `pyuic`, make python put a helloworld window onto your screen! Your window should look something like this:



Lesson 2: Program structure

Objective: Show the general structure of the player program.

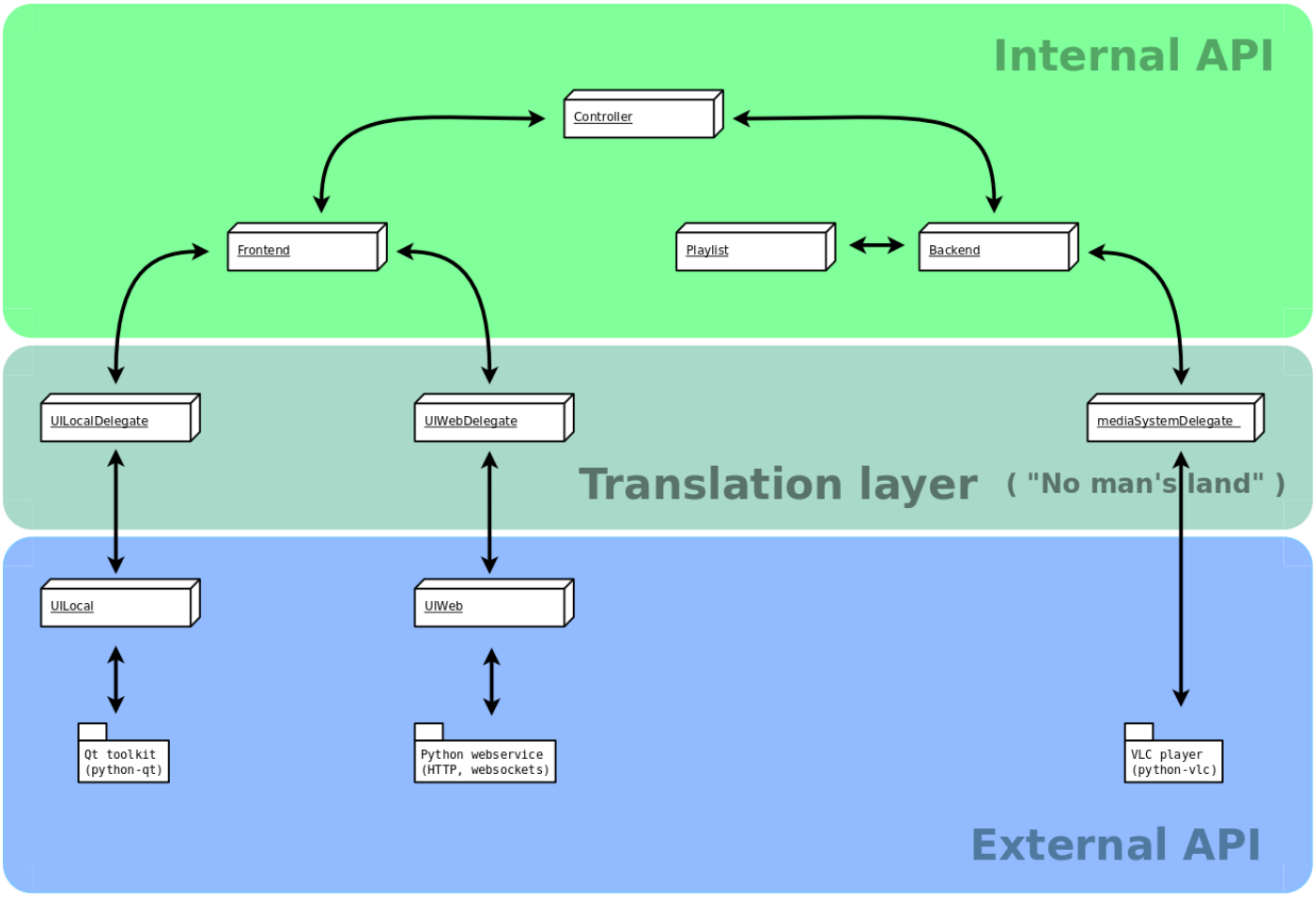


fig 1: Program overview

This chapter is a first get-to-know-me in broad strokes. We present the overall design of our player application. Subsequent chapters will go into much more detail, but this one wants to present the "big picture".

A good program design is hard to get to and needs experience! In today's interconnected world everything is super complicated! Design flaws are hard to beat once the project grows, so it's very important to get things right early! Several principles guide the development of MyPlayer.

- *Separation of concerns*. Example from the real world: A well designed company structure will result in employees who are competent in what they do and communicate well. Software design is similar! We take take a big problem, split it up into sub problems then write components that tackle those sub problems independently and work well together. Result: Program works well and can be maintained by dummies!

  MyPlayer realizes this principle by splitting up the whole application into smaller independent components. Furthermore, we isolate the components in separate layers. We have a system layer which operates the system parts (Local GUI, web GUI, VLC player), a control layer which represents the player's view of everything (controller, frontend and backend facades) and a translation layer (delegates) which translates what the control

components want to a "language" which the system components understand (and vice versa).

- Talking about "language" - *API, API, and again, API !* Did I say API? Good separation of concerns is only half the story. What good is the independence if the stuff doesn't talk to each other? We need a well defined interaction between the different components. In programmer's mumbo jumbo such interaction way is called *API* (*Application programming interface*). API thinking is all over the application.
  - Each component provides a clean interface - a bunch of methods to set getters and setters (methods that get and set stuff inside the component) and event handlers - which respond to other components wanting something. All methods follow a consistent naming scheme. For example, getters always start with `get`.
  - The components in the system layer all have very different requirements and thus very different methods. To present a more consistent API of the system components we use a layer of delegates that "translate" between the resp. system components and the control components. As a result the control components have a consistent view onto the system components. For example, the two frontends (Local GUI and web GUI) are very different in what they do and how they, but thanks to their respective delegates they look the same to the Frontend facade. This approach reduces complexity so we can pretty much plug any frontend into the running application and access them all same way.

    For the media player component (VLC) we also use a delegate; technically this isn't necessary, but we still use it in case we want to use a different media framework. Plus, it's just more consistent - now *any* system component talks to our application via a delegate. This means one less special case and one thing less to consider!

Getting a design right is the hardest part of software design. This setup is just an attempt to make the application easier to understand and maintain. We meet the design challenge by splitting the whole setup into separate components and providing a clearly defined interaction between those components.

Some code to layout

Pardon the weird language, but *layout* is jargonese for the way our source code is organized! In our longish introduction we talked about layers and modules. This reflects directly in how we set up the packages and modules - or in everyday speech - the folder structure and what source files we put there. I'm sure there are more interesting ways of laying out code, but I sugggest we organize it somewhat along the layers and modules we presented above. Fig. 2 shows what it looks like.

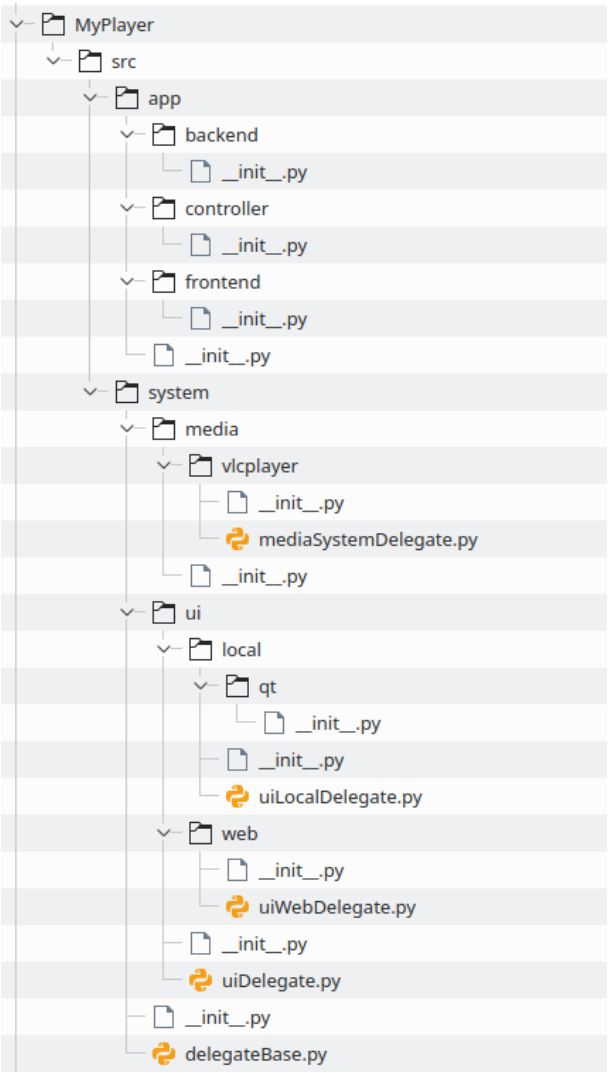


fig 2: Source layout

Noticed something? Blimey, where are those delegates???? If you look closely you will see them lumped with the respective system components! The local GUI has got a **`uiLocalDelegate`**, the web UI has got a **`uiWebDelegate`** and the VLC driver has got a **`mediaSystemDelegate`**. The app layer components will use the delegates, and *only* the delegates to use the various system components. We'll see the details of all this later, but for now we note that the app layer talks in appanese with the delegates who translate that stuff into systemese for the system layer (and vice versa). If we pull this off correctly then it will be a breeze to integrate the system components into the app components!

The astute reader will have observed that there's no test code! That's right - but this guide is already demanding enough! Throwing in unit testing would be like adding rocks to your full grain muesli!

Task: In LiClipse, set up the project and create a structure of python modules implementing the program structure we explained.

As a plus, maybe make a GUI and let python execute it.

Using **`Konsole`**, put the project under git control.