Master Thesis

Design, Implementation and Application of Linux-Based Video Application Development Framework

Jun Bao

Student ID: 4342512

Jun.bao@fu.berlin.de

Department of Mathematics and Computer Science

Institute of Computer Science

AG Intelligent System und Robotic

Supervisor: Prof. Dr. Raúl Rojas

Free University of Berlin

July 2012

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 17. Jul. 2012

Jun Bao

Abstract

With the increasing popularity and need of video applications, there has been a lot of interest being invested both by the research and the industrial community to bring new tools to develop and present video applications. In development of applications, a framework is usually applied to reduce the cost and insure quality of the software. In this thesis, I will introduce you an OO-framework, which is designed with MVC model. This framework can be used to develop a video application under Linux. In addition to the framework we provide two libraries, i.e. UVC API and HID API, to access and control devices. Finally it will be introduced that how to apply the framework and libraries in a migration project from Windows to Linux.

Keywords: framework, UVC, HID, Qt, Linux, object-oriented, MVC

Table of Contents

List of Figures

List of Tables

Abbreviations

ACM	Audio Compression Manager
API	Application Programming Interface
ASF	Advanced Systems Format
AVI	Audio-Video Interleaved
COM	Component Object Model
GUI	Graphical user interface
HID	Human Interface Device
J2SE	Java 2 Platform, Standard Edition
JMF	Java Media Framework
MP3	MPEG Audio Layer-3
MPEG	Motion Picture Expert Group
MVC	Model–View–Controller
USB	Universal Serial Bus
UVC	USB Video Device Class
V4L	Video for Linux
V4L2	Second Version of V4L
VCM	Video Compression Manager
VfW	Video for Windows
WAV	Waveform Audio File

WDM	Windows Driver Model

1 Introduction

1.1 Motivation

If you are a professional programmer, or just an amateur developer, you definitely want to know how to develop applications in a short time and guarantee the quality? I think one of the best ways is choosing a suitable framework.in the process of development

In computer programming, a software framework[1] consists of a set of code or libraries used by developers to implement the standard structure of an application for a specific development environment. A Framework is a reusable “semi-complete” application that can be specialized to produce custom applications. Frameworks promise higher productivity and shorter time-to-market through design and code reuse[2]. A good framework can not only reduce the cost but also improve the quality of software.

In this thesis I will introduce you a new framework for developing video applications. Even though there have been a few frameworks, such as DirectShow[3] and JMF, can be used to develop video applications, these frameworks are unable to meet all requirements. The DirectShow is a platform dependent framework, which is based on the Microsoft Windows Component Object Model framework[4], because it is a product of Microsoft. The JMF is a Java library that enables audio, video and other time-based media to be added to Java applications and applets. The JMF allows development of cross-platform multimedia applications. But for non-Java developers under Linux, they still don’t have a useful framework for developing video applications. The purpose of this paper is to design a new C++ and Linux-based video application development framework. This framework includes some libraries by using v4l2 and hidapi to support UVC devices and HID[5] devices. With the help of this framework the developers don’t need to devote attention any more for handling “lower level” tasks. Finally we will apply this framework in a project to test and verify the usability and reliability.

1.2 Statement of Problem

One framework can't support more devices. In recent years, new video devices come one after another, followed by a rapid growth of associated applications. We need a framework that is capable of application development for more devices.

The framework can't be too complex. It must be easy to use. Several times I have heard that some developers complained that the existing frameworks are too large. Although the framework can meet various requirements of users, most of the functionalities have no chance to be used. Using this kind of framework maybe can reduce the difficulty of development, but it increases the burden on the developers, even probably reduces the efficiency of the program. A light-weight development framework is what we need.

As we know most video devices follow a standard, but more or less they have some features of one's own. In order to adapt to the new features of the new equipment, the framework itself should be capable of advancing. It is should be involved to consider the scalability during the design of the framework.

1.3 Related Work

1.3.1 DirectShow (Windows)

DirectShow is Microsoft's media streaming framework and is part of the DirectX SDK[6]. Currently, it is distributed as part of the Windows SDK. DirectShow contains all of the components for audio/video capture and playback in a multitude of formats. It supports a wide variety of formats, such as ASF, MPEG, AVI, MP3 and WAV. It supports also a wide variety of audio/video input devices including DV camcorders, Web cams, DVD drives and TV tuner cards. DirectShow is based on the COM[7]. To write a DirectShow application, you don't need to implement your own COM objects, because DirectShow provides the components you need. For example, Windows Media Player is one of the applications based on DirectShow and uses it for playback, while Windows Movie Maker is used for capture, editing and playback.

DirectShow uses a modular architecture, in which the application mixes and matches different software components called filters. DirectShow make it simple to create digital

media applications on the Windows platform, by providing filters that support capture and tuning devices based on WDM, as well as filters that support legacy VfW capture cards, and codecs written for the ACM and VCM interfaces.

Filters are the basic elements of DirectShow. There are three types of filter: Source, Transform and Rendering. Source Filters are used to get data from outside source, such as file system, cameras or TV tuners. Rendering Filters are on the opposite end and write media to a file or render the video or audio. In the middle are transform filters which could do a lot of things. Transform filters get data from source filters and transform the source data into a format that a rendering filter can play. If the source data are compressed, a transform filter would uncompress the data first and then send to the next filter. Transform Filters can also be used to parse a data stream, which includes both audio and video data and after processing combine these two data streams back into one stream.

Figure 1-1 shows the relationship between an application, the DirectShow components, and some of the hardware and software components that DirectShow supports.

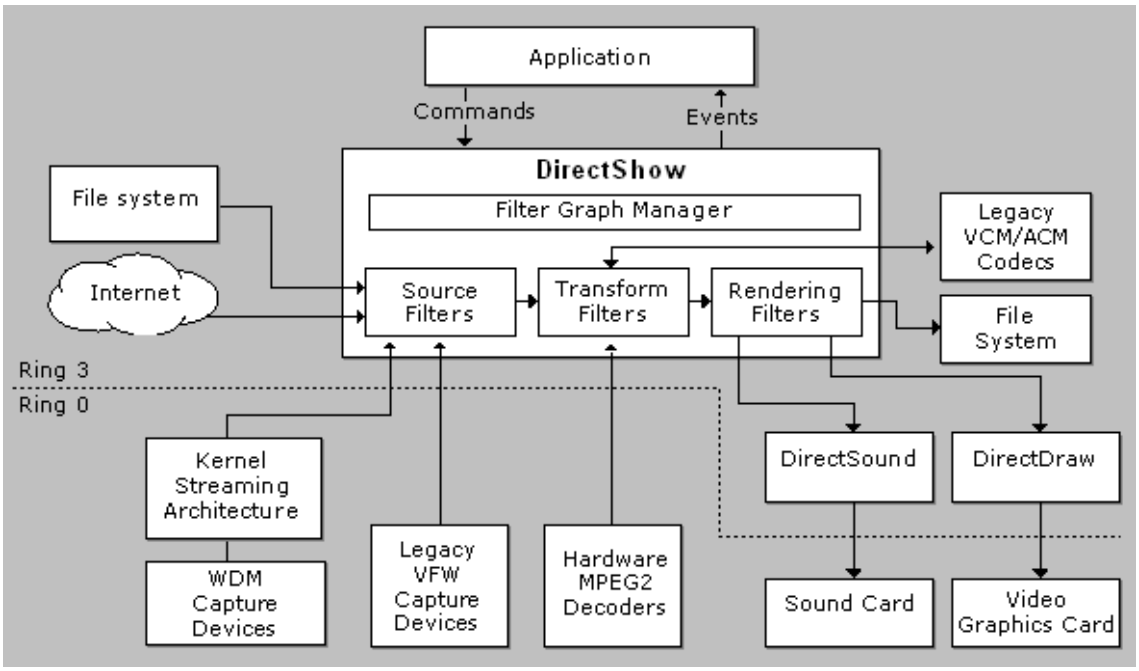

Figure 1-1 DirectShow System

As illustrated here, DirectShow filters communicate with, and control, a wide variety of devices, including the local file system, TV tuner and video capture cards, VfW codecs,

the monitor (through DirectDraw or GDI), and the sound card (through DirectSound). Thus, DirectShow insulates the application from many of the complexities of these devices. DirectShow also provides native compression and decompression filters for certain file formats.

But Microsoft plans to completely replace DirectShow gradually with Media Foundation[8] in future Windows versions. Windows Vista and Windows 7 applications use Media Foundation instead of DirectShow for several media related tasks.

1.3.2 JMF (Java)

The JMF is a Java library that enables audio, video and other time-based media to be added to Java applications and applets[9]. JMF specifies a unified architecture and messaging protocol for capture, processing and storing of time-based media. It supports a wide variety of formats, such as WAV, AVI, MPEG, QuickTime and so on. Java is a platform independent programming language. With the help of JMF programmers can develop applications that can run on any kind of platform, because JMF provides a cross-platform API to access the underlying media framework.

An initial, playback-only version of JMF was developed by Sun Microsystems, Silicon Graphics, and Intel, and released as JMF 1.0 in 1997. JMF 2.0, developed by Sun and IBM, came out in 1999 and added capture, streaming, pluggable codecs, and transcoding. JMF is branded as part of Sun's "Desktop" technology of J2SE opposed to the Java server-side and client-side application frameworks. The latest version of JMF API also supports data transmission and reception over the network.

The main goal of JMF is to provide an easy-to-use API[10] for developing media applications. Figure 1-2 shows the layered architecture of JMF[11].

As we see, JMF consists of two layers of API. The higher-level API, called the JMF Presentation and Processing API, supports for capture, presentation, and processing of time-based media. The lower-level API manages the integration of custom processing components and extensions. JMF can be integrated with Java application, applets or JavaBeans seamlessly.

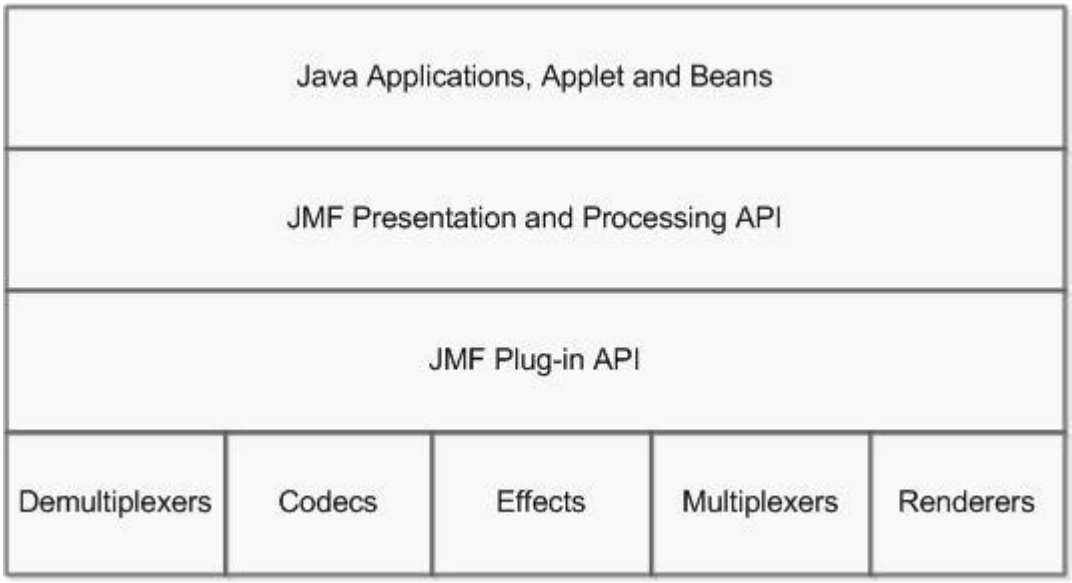

Figure 1-2 JMF Architecture

The main classes of high-level API are:

MediaLocator: describes the location of a media stream.

Datasource: encapsulates the location of media.

Player: processes a stream of data.

Processor: a special type of player that can provide control over how the media data are processed before they are presented.

Datasink: reads media content delivered from a *DataSource,* and renders the media to some destination, such as a file.

Manager: a factory class which constructs the instance of *Player, Processor, Data-Source* and *DataSink* class.

The low-level API consists of five "plug-in" components:

Demultiplexers: extracts individual tracks of media from a multiplexed media stream.

Multiplexers: joins individual tracks into a single media streams.

Codec: performs media stream encoding and decoding.

Effect filters: modifies the track data in some way, often creates some special effects.

Renderers: delivers the media stream to the presentation device.

1.4 Structure of the Thesis

Section 2 discusses the technical background necessary to understand the development of a framework for video applications.

Section 3 discusses the strategies and the process to design a new framework for developing video applications.

Section 4 discusses the implementation of the framework.

Section 5 introduces a migration project in which our framework applies.

Section 6 concludes this work and discusses future directions.

2 Background, Basic Concepts

2.1 Environment

2.1.1 Linux

Linux is a free open-source operating system based on UNIX[12]. Linux was originally created by Linus Torvalds and first released on 5th October 1991. Typically Linux is released in a format known as a Linux distribution. In the past twenty years, Linux has evolved into several distributions. Some popular Linux distributions are Debian (and its derivatives such as Ubuntu), Fedora and openSUSE. Linux distributions include the Linux kernel, supporting utilities and libraries and usually a large amount of application software.

Due to its open source policy, Linux has become one of the most popular operating system. Although Linux was originally design as a free operating system for personal computer, it has been ported more computer hardware platform than any other operating system. Today Linux can also run on embedded systems such as mobile phone, tablet computers, network routers and video game consoles.

With the increasing popularity of Linux, there has been a lot of interest being invested both by the research and the industrial community to develop new applications for Linux[13]. Usually the programmers develop software with several versions to support different operating systems, in which Linux should be involved. For some applications which have been widely used under Windows, the developers are considering to transplant them from Windows to Linux. In this paper I will describe a video application framework and its application in a Windows-to-Linux migration project.

2.1.2 Qt SDK

The Qt SDK [14]combines the Qt framework with tools designed to streamline the creation of applications for different platforms, such as Microsoft Windows, Mac OS X, MeeGo and Linux.

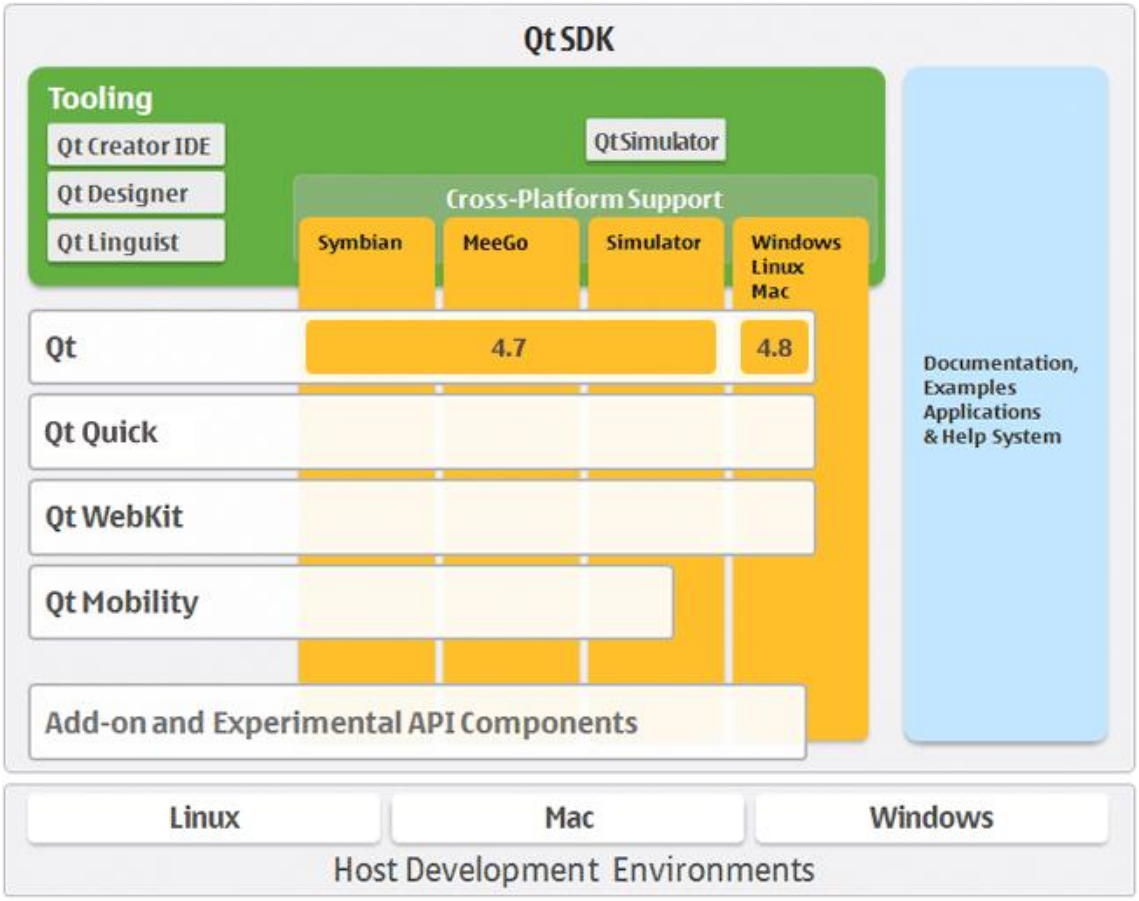

Figure 2-1 Structure of Qt SDK

Qt is a cross-platform application and UI framework based on C++. Qt is primarily a GUI framework that includes a rich set of widgets, graphics rendering APIs, layout and stylesheet that can be used to create a cross-platform application. Widgets range from simple objects such as push buttons and labels to advanced widgets such as menus, calendars and dockable toolbars. The Widgets are native look and feel on different target platform. For instance, the widgets look like native Macintosh widgets on Mac OS X, while the same widgets will look like native Windows widgets on Windows.

Qt Creator is a cross-platform integrated development environment (IDE) that provides you with tools to design and develop applications with the Qt application framework. Qt

Creator provides you with tools for accomplishing your tasks throughout the whole application development life-cycle, from creating a project to deploying the application on the target platforms.

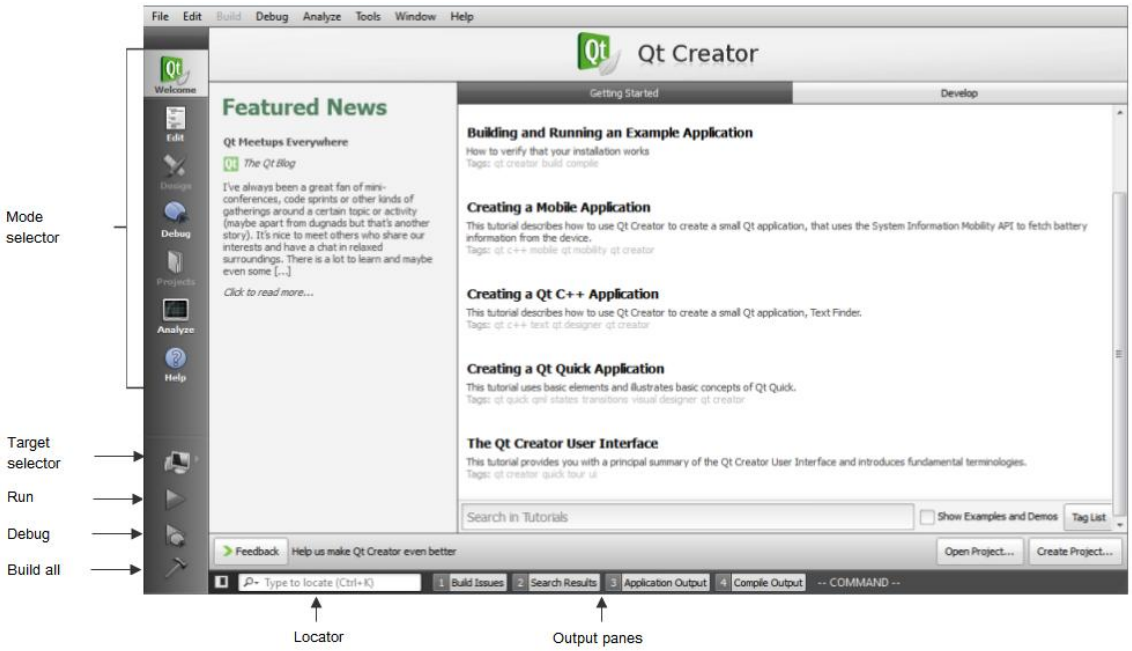

Figure 2-2 Qt Creator

2.2 Device

2.2.1 UVC

The USB Device Class Definition for Video Devices applies to all devices or functions within composite devices that are used to manipulate video and video-related functionality. This would include devices such as webcams, digital camcorders, analog video converters, analog and digital television tuners, and still-image cameras that support video streaming.

The latest revision of the USB video class specification carries the version number 1.1 and was defined by the USB Implementers Forum in a set of documents describing both the basic protocol and the different payload formats[15].

Among a variety of devices meeting the specification of UVC, webcams were the first devices to support UVC standard and they are currently the most popular UVC devices. A webcam is a video camera that captures images in real time. Webcams are used with many personal computers, because the low manufacturing cost and flexibility. And most of the operating system start providing support for UVC devices. For example, since Linux 2.6.26 the UVC driver is included in kernel source distribution.

2.2.2 HID

As the name suggests, a human interface device (HID) is a type of computer device that interacts directly with, and most often takes input from humans and may deliver output to humans. But it doesn’t necessarily have a human interface. All devices follow the HID protocol are called HID device. The term "HID" most commonly refers to the USB-HID specification.

A HID device includes 2 entities: the "host" and the "device". The device is the entity that directly interacts with a human, such as a keyboard or mouse. The host communicates between the device and human. It receives input data from the device. Output data flows from the host to the device. The most common example of a host is a PC but some cell phones and PDAs also can be hosts.

The HID protocol makes implementation of devices very simple. Devices define their data packets and then present a "HID descriptor" to the host. The HID descriptor is a hard coded array of bytes that describe the device's data packets. This includes: how many packets the device supports, how large are the packets, and the purpose of each byte and bit in the packet. The host is expected to be a more complex entity than the device. The host needs to retrieve the HID descriptor from the device and parse it before it can fully communicate with the device.

Since HID's original definition over USB, HID is now also used in other computer communication buses. HID seems more and more useful in the future. It is widely used in many areas. For example, it can be used with a video device to control brightness, saturation, frequency and so on.

2.3 APIs

2.3.1 V4L2

V4L is a video capture application programming interface for Linux. Many USB webcams, TV tuners, and other devices are supported. V4L is closely integrated with the Linux kernel.

V4L2 is the second version of V4L. The original V4L was introduced in Linux 2.2 by Alan Cox's V4L API. V4L2 fixed some design bugs and started appearing in the 2.5 kernels. V4L2 includes a compatibility mode for V4L and supports a wider range of devices.

In most cases, a V4L2 application may include the next steps:

- Open the device.
- Properties Negotiation (video input, video standard, and more)
- Data Format Negotiation
- Input/Output Method
- Main Loop
- Close the device

The V4L2 applications must be programmed using the *ioctl* system function. *ioctl* is used with the following syntax:

ioctl (<device_descriptor>, <operation_code>, <pointer_to_structure>)

For example, ioctl (fd, VIDIOC_QUERYCAP, v4l2_capability).

It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The ioctl takes a pointer to a struct v4l2_capability which is filled by the driver. When the driver is not compatible with this specification the ioctl returns an EINVAL error code.

2.3.2 HIDAPI

HIDAPI is a cross-platform library which allows an application to access HID devices on Windows, Linux, and Mac OS X. While it can be used to communicate with standard HID devices like keyboards, mice, and Joysticks, it is most useful when used with custom HID devices. Many devices do this in order to not require a custom driver to be written for each platform. HIDAPI is easy to integrate with the client application, just requiring a single source file to be dropped into the application.

Programs which use HIDAPI are driverless, meaning they do not require the use of a custom driver for each device on each platform. HIDAPI provides a clean and consistent interface for each platform, making it easier to develop applications which communicate with USB HID devices without having to know the details of the HID libraries and interfaces on each platform.

HIDAPI 0.1.0 was released in 2010 as the first version and the latest version is 0.7.0 which was released in November 2011. The API provides an easy way to enumerate HID devices attached to the system, and easy access to the functionality of the most commonly used HID functions including transfer of Input, Output, and Feature Reports.

Table 2-1 lists all the functions provided by HIDAPI.

Function Name	Description
hid_init	Initialize the hidapi library.
hid_exit	Finalize the hidapi library.
hid_enumerate	Enumerate the HID Devices.
hid_free_enumeration	Free an enumeration Linked List.
hid_open	Open a HID device using a Vendor ID (VID), Product ID (PID) and optionally a serial number.
hid_open_path	Open a HID device by its path name.

hid_write	Write an Output report to a HID device.
hid_read_timeout	Read an Input report from a HID device with timeout.
hid_read	Read an Input report from a HID device.
hid_set_nonblocking	Set the device handle to be non-blocking.
hid_send_feature_report	Send a Feature report to the device.
hid_get_feature_report	Get a feature report from a HID device.
hid_close	Close a HID device.
hid_get_manufacturer_string	Get The Manufacturer String from a HID device.
hid_get_product_string	Get The Product String from a HID device.
hid_get_serial_number_string	Get The Serial Number String from a HID device.
hid_get_indexed_string	Get a string from a HID device, based on its string index.
hid_error	Get a string describing the last error which occurred.

Table 2-1 Functions included in HIDAPI 0.7.0

3 Design of the New Framework

3.1 Introduction

Designing a framework is not an easy work. There are many things must be taken into consideration. In this section we will discuss the major points and issues related to the design of an application framework. The high points are listed below.

- Establishing Requirements
- Determining the Scope of the framework
- Abstract Design

3.2 Establishing Requirements

Probably the most important factor influencing the success or failure of an application framework is identifying the problem you want to solve exactly. If you identify the wrong problem to solve, the solution is definitely wrong for your need. You have to know your requirements clearly, if you want to design a useful and powerful application framework.

Before you design the framework, there are some questions you have to ask yourself.

What kind of applications do you build?

Since the framework provides the skeleton for your application, the type of the application will influence the structure of the framework. Each type of the application would require a different type of interaction with the user. Each type of the application would have a different data source. Each type of the application would work with different devices.

In this paper we will introduce a framework used to develop video applications based on Linux. The video applications work with multimedia streams containing large amounts of data, which must be processed very quickly.

Who will use the application framework?

In final analysis, the framework is designed for the programmers to develop an application faster and better. When we design the framework, we must take into account the actual conditions of the users. What programming languages the developers use? What platform the developers work on? All of these problems will influence the design of the framework.

The framework introduced in this paper is based on Linux and was developed for a migration project, in which the original application under Windows was implemented in C++. In order to keep the continuity, the new framework was implemented also in C++.

Is there already any similar framework?

Before the design of the framework, we have to do some research. One most important work is to find out and analyze the frameworks that can provide similar functionalities. Each framework has its own disadvantages and disadvantages. Through the comparison of the existing frameworks, we could take the advantages and overcome the disadvantages.

The framework introduced in this paper was developed for a migration project. The original application was developed for Windows. As we know, there is a famous multimedia framework called DirectShow, which is a product of Microsoft and have been widely used for developing video applications. Another popular multimedia framework is JMF, which is implemented in Java and easily integrated in a Java application, applet or Java Beans.

3.3 Determining the Scope of the framework

As we know the DirectShow is a product of Microsoft and JMF is a product of Sun. The structure of DirectShow and JMF must be complicated. They provide many functionalities, most of which could be never used in development of a normal application. Maybe

it is not a problem that gets more than you want. But for an experienced developer choosing a right framework can not only make the development efficiently but also reduce the cost.

You need to determine for what the framework is responsible and for what the developer is responsible. Don't use or build a framework that makes the developer work harder to get the result they want.

Video application is application software based on images captured by cameras and handles the processing of digital video sequences on a computer. Usually the video applications use special-purpose, fixed-function device to capture images. But our framework just provides general-purpose solutions that are independent on the special hardware and functions. Figure 3.1 shows the media processing model.

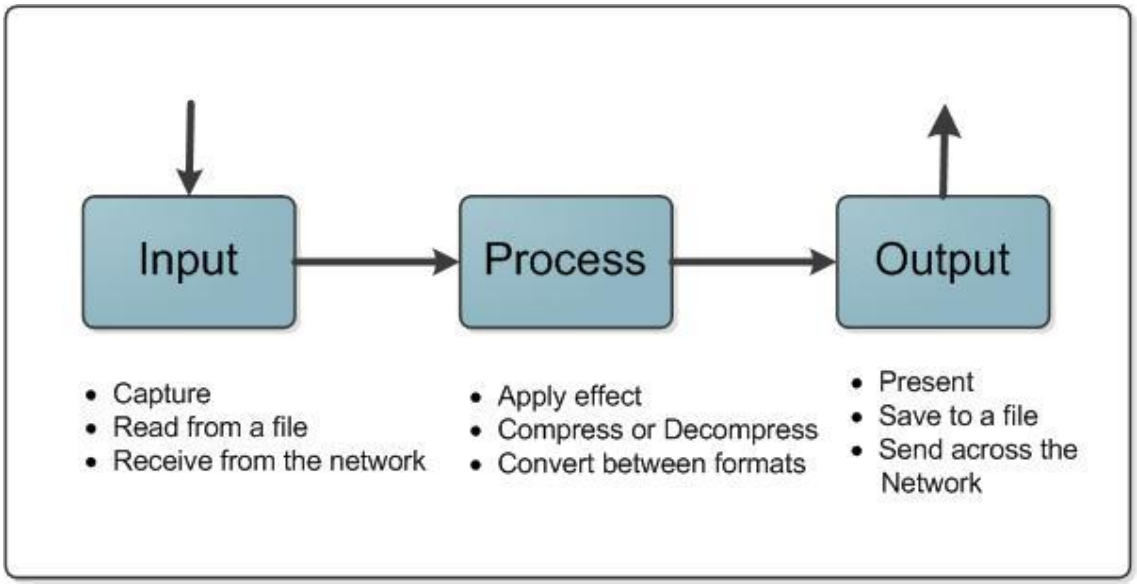

Figure 3-1 Media Processing Model

As shown in figure 3-1, the media processing model include three parts, input, process, and output. The input module takes charge of communicating with the devices and getting media data. Then the media data are transferred to process module, in which the media data are processed by some custom designed methods, such as applying effect, compressing or decompressing, and converting from one format to another. After the processing, media data come to the last step. In this step the media date will be presented to users or stored in a file.

3.4 Abstract Design

A framework is a universal, reusable software platform used to develop applications, products and solutions. It is not just a collection of libraries. A framework can be extended or specialized by the users' code to provide specific functionalities. The framework code, in general, is not allowed to be modified. User can extend the framework, but not modify its code. Nowadays developers usually use object-oriented programming techniques to implement frameworks such that the unique parts of an application can simply inherit from pre-existing classes in the framework.

Object-oriented (OO) application framework is a promising technology for software designs and implementations in order to reduce the cost and improve the quality of software[16]. In contrast to earlier OO reuse techniques based on class libraries, frameworks are targeted for particular business units and application domains.

If the object-oriented technology brings benefits on modularity, reusability and extensibility, then MVC makes the structure of software clearly. Model–View–Controller (MVC) [17]is considered as a design pattern of software design.

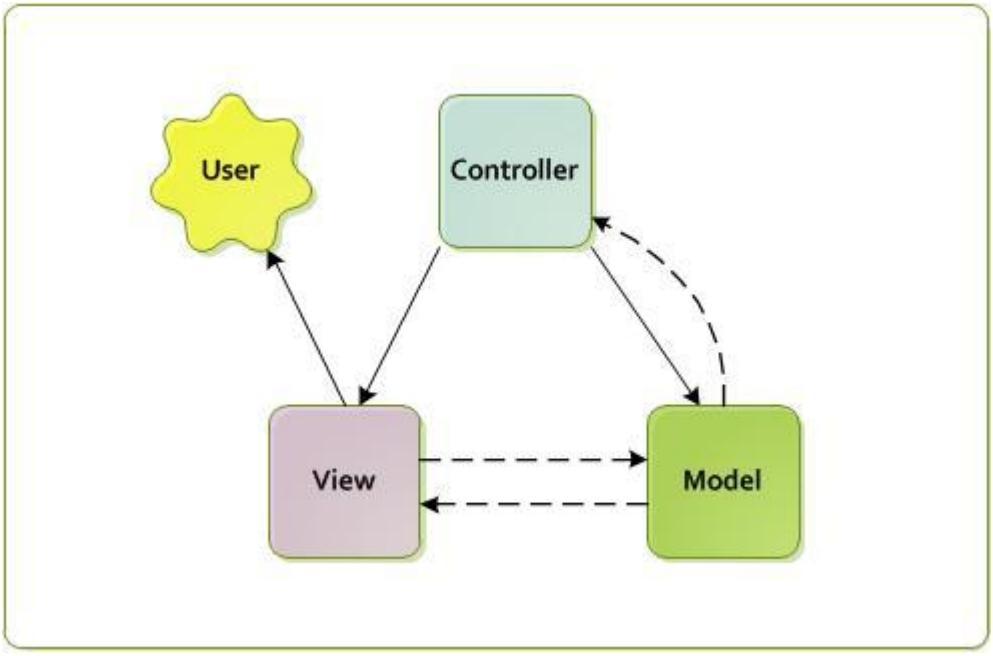

Figure 3-2 Relationship of MVC and User

As shown in Figure 3-2, The Model represents the data of the application and the business rules used to manipulate the data. The View can be any output representation of data, such as a chart or a diagram. The Controller manages the communications between the Model and View.

3.4.1 Design of Framework

According to the MVC model, we design a framework for development of video application. Figure 3-3 shows the architecture of the framework.

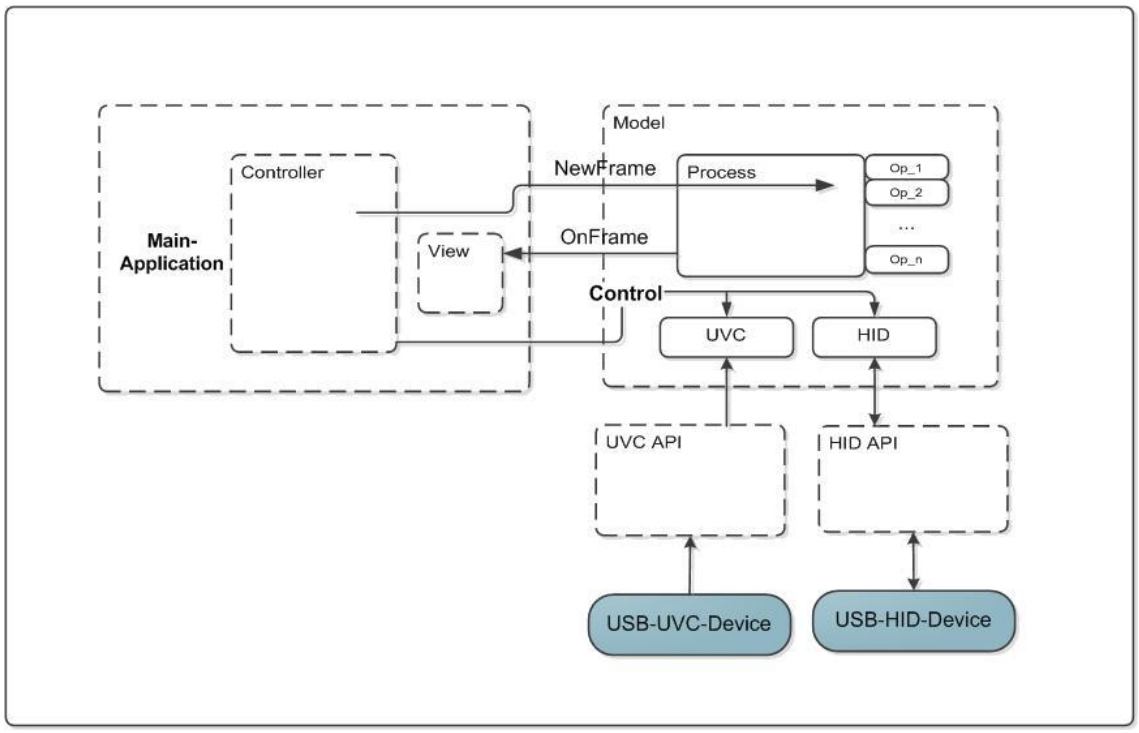

Figure 3-3 Architecture of the Framework

OnFrame is a callback used for communication between view and process. In computer programming, a callback is a reference to a piece of executable code that is passed as an argument to other code. This allows a lower-level software layer to call a function defined in a higher-level layer. As shown in figure 3-3, callback *OnFrame* is called, when a frame of image is completely processed and ready for the next step. It is like an event-driven programming that the flow of the program is determined by events.

The View is the component takes charge in representation of the output. It requests from the Model the information that it needs to present for users. A view can be any

output representation of data, such as a chart or a diagram. As we know, Qt Framework includes a GUI module which contains the functionality needed to develop advanced graphical user interface applications. The Qt GUI module supports the native graphics API of Linux that a native look and feel user interface is possible. One the greatest feature is that you can create customized UIs with a unique look and feel with the Qt GUI. For example, widgets are the primary elements for creating user interfaces, because widgets can display data and status information, or provide a container for other widgets. In Qt the *QWidget* class provides all functionalities which a widget should have. All UI elements that Qt provides are either subclasses of *QWidget*, or are used in connection with a *QWidget* subclass.

The Model is the components developed by users to provide specific functionalities. In MVC this is the part of your application that defines its basic functionality behind a set of abstractions. It is the core of an application. Data access routines and some business logic can be defined in the model. In our framework, a few classes are defined as models. For example, class UVC and HID represent UVC device and HID device respectively, and define all operations associated with the devices in the classes. Class process is an abstract class can be inherited and extended by subclasses, which are in charge in processing the data, such as change format, congress or uncongress, and apply effect.

In addition to stability, efficiency is also an important factor for a video application, because multimedia streams contain large amount of data, which must be processed very quickly. Multithreading is definitely a good solution. A thread is the smallest unit of processing that can be scheduled by an operating system. Multiple threads can exist within the same process and share resources. Qt provides platform-independent threads called *QThread*, which can be used to control UVC device. Each thread represents a UVC device so that they can work parallel with the main application.

An API is a specification intended to be used as an interface by software components to communicate with each other. An API specification can take many forms, including an International Standard such as POSIX or vendor documentation such as the Microsoft Windows API, or the libraries of a programming language, such as Standard Template Library in C++ or Java API. The UVC API and HID API are two useful libraries provided by our framework. The API describes and prescribes the expected behavior while the library is an actual implementation of this set of rules. A single API can have multi-

ple implementations in the form of libraries that share the same programming interfaces. We don't need to worry about the change of libraries, because the APIs are stable. The design of UVC API and HID API are described below.

3.4.2 Design of UVC API

Since Linux 2.6 a driver for UVC device is included in the kernel source distribution. But there are still many people using the earlier version of Linux which includes no UVC driver and we can't promise that the UVC driver will not be replaced by another driver in future versions of Linux. In order to promise a continuity and stability, we decide to develop a library called UVC API for UVC device. This library is based on V4L2 and used to capture images from the UVC devices.

V4L2 is a video capture application programming interface for Linux that we have introduced in section 2.3.1. V4L2 includes some functions used to access V4L2 device. These functions can be directly called by applications. Table 3-1 lists these functions.

Function Name	Description
open ( )	Open a V4L2 device
close ( )	Close a V4L2 device.
ioctl ( )	Program a V4L2 device.
mmap ( )	Map device memory into application address space.
munmap ( )	Unmap device memory.
read ( )	Read from a V4L2 device.
write ( )	Write to a V4L2 device.
select ( )	Synchronous I/O multiplexing.
poll ( )	Wait for some event on a file descriptor.

Table 3-1 V4L2 API functions

In practice usually through the execution of function *ioctl()* to set and read the contents of V4L2 data structure, table 3-2 lists some important *ioctl* functions.

Function Name	Description
ioctl VIDIOC_CROPCAP	Information about the video cropping and scaling abilities
ioctl VIDIOC_G_CROP, VIDIOC_S_CROP	Get or set the current cropping rectangle.
ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT	Get or set the data format, try a format.
ioctl VIDIOC_G_STD, VIDIOC_S_STD	Query or select the video standard of the current input.
ioctl VIDIOC_QBUF, VIDIOC_DQBUF	Exchange a buffer with the driver.
ioctl VIDIOC_QUERYCAP	Query device capabilities.

Table 3-2 Common used ioctl functions

According to the definition of V4L2, a V4L2 application usually consists of these steps:

- Opening the device
- Changing device properties, selecting a video and audio input, video standard, picture brightness a. o.
- Negotiating a data format
- Negotiating an input/output method
- The actual input/output loop
- Closing the device

In practice most steps are optional and can be executed out of order. It depends on the V4L2 device type. In figure 3-4 it shows the basic work flow of a video capture de-

vice[18]. At first, it opens a device and queries the capabilities. If devices support the video capture interface, the V4L2_CAP_VIDEO_CAPTURE flag in the capabilities field of struct v4l2_capability returned by the VIDIOC_QUERYCAP ioctl must has been set. Then you have to set an image format and prepare the memory to store the captured images. After that you can start the working loop and capture images until a stop signal has been got. The capture process will be finished, when the device is closed.

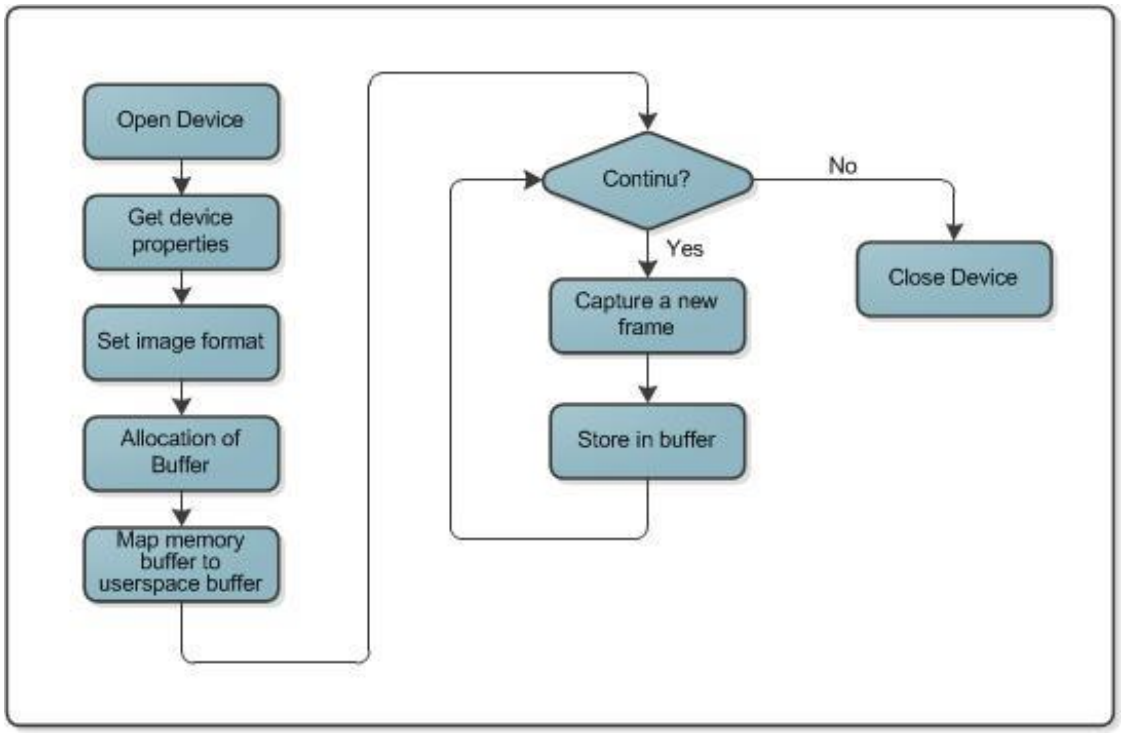

Figure 3-4 Basic Work Flow of the Applications based on V4L2

According to the work flow of V4L2 we design a set of interfaces for our framework to develop applications for UVC device, so it called UVC API. With this interface applications can control the capture process and move images from the driver into user space. With this interface developers don't need to care about the "lower" detail any more, it makes the development faster and reduces cost. Table 3-3 list the operations and functions defined in UVC API.

Function name	Description
open_device	Open a UVC device by its path name.

close_device	Close a UVC device.
init_device	Initialize the UVC device.
start_capturing	Start to capture images.
stop_capturing	Stop to capture images.
uninit_device	Finalize the UVC device.
get_frame	Capture a frame and store in buffer.
unget_frame	Release a filled buffer.
get_framesize	Get the size of frame and the frequency.

Table 3-3 Definition of UVC API

3.4.3 Design of HID API

HID device is different from UVC device. HID device is a type of computer device that interacts with humans, while UVC device can only deliver output to humans. The HID standard was developed by a working committee with representatives from several prominent companies. Most operating system can recognize standard USB HID device without needing a specialized driver.

The HID protocol makes implementation of devices very simple. Devices define their data packets and then present a "HID descriptor" to the host. The HID descriptor is a hard coded array of bytes that describe the device's data packets. This includes: how many packets the device supports, how large are the packets, and the purpose of each byte and bit in the packet. The device typically stores the HID descriptor in ROM and does not need to intrinsically understand or parse the HID descriptor. The data packets defined by HID descriptor are called report. These reports are used to communicate between device and host. The main task of HID API is how to request and send report.

hidapi is an existing library which allows an application to interface with HID devices on Linux. It can be used to communicate with standard HID devices like keyboards,

mice, and Joysticks. In order to support more devices especially custom (Vendor-Defined) HID devices, we design the HID API based on *hidapi* for our framework. Table 3-4 lists the operations and functions defined in HID API.

Function name	**Description**
get_device_info	Get the information from a HID device.
open_device	Open a HID device.
close_device	Close a HID device.
get_send_buffer_len	Return the length of input report.
get_receive_buffer_len	Return the length of output report.
command	Send a command to the HID device.
get_hardware_rev	Get the value of hardware revision.
read_from_device	Read an Input report from a HID device.
write_to_device	Write an Output report to a HID device.

Table 3-4 Definition of HID API

4 Implementation of the New Framework

4.1 Implementation of Controller

The Qt Framework has provided a class *QApplication*, which manages the GUI application's control flow and main settings. *QApplication* contains the main event loop, where all events from the window system and other sources are processed and dispatched. It receives events from the underlying window system and dispatches them to the relevant widgets. By using *sendEvent()* and *postEvent()* you can send your own event to widgets. *QApplication* is like the controller of a Qt application. To create a *QApplication* in *main()*, the code looks like this:

```
#include <QtGui>

// include header files for application components.
//…
int main (int argc, char *argv[ ])
{
   QApplication app (argc, argv);

   //Set up and show widgets.
   //…
    return app.exec ( );
}
```

4.2 Implementation of View

As we know the graphical user interface (GUI) is widely accepted and applied in development of modern applications to interact between users and electronic devices, because it is more intuitive and easy to understand than text commands. But it is difficult to design a native look and feel user interface for a software developer without design experience. In order to create a custom user interface quickly and easily, we decide to involve Qt GUI in our framework[19]. Then how to use Qt GUI to develop an advanced graphical user interface in our framework?

For example, we are going to develop a video application to present images captured by video devices in real time. How can we deign the user interface? At first, we should at least need a main window to present the images. Then a menu bar is necessary that it can provide access to such functions as open device, pause, or stop.

```
MainWindow::MainWindow (QWidget *parent)
    : QMainWindow (parent)
{
    …
    openAct = new QAction (tr ("&Open..."), this);
    openAct->setShortcuts (QKeySequence::Open);
    connect (openAct, SIGNAL (triggered ()), this, SLOT (open ()));

    newAct = new QAction (tr ("&Pause"), this);
    newAct->setShortcuts (QKeySequence::Pause);
    connect (newAct, SIGNAL (triggered ()), this, SLOT (pause ()));

    fileMenu = menuBar ()->addMenu (tr ("&Device"));
    fileMenu->addAction (openAct);
    fileMenu->addAction (pauseAct);
    …
```

Generally, we subclass *QMainWindow* and set up menus. To add a menu bar to the main window, we simply create the menus, and add them to the main window's menu bar. Once actions have been created, we can add them to the main window components.

4.3 Implementation of UVC API

Video devices capture images usually follow a standard process, which is shown in figure 4-1.

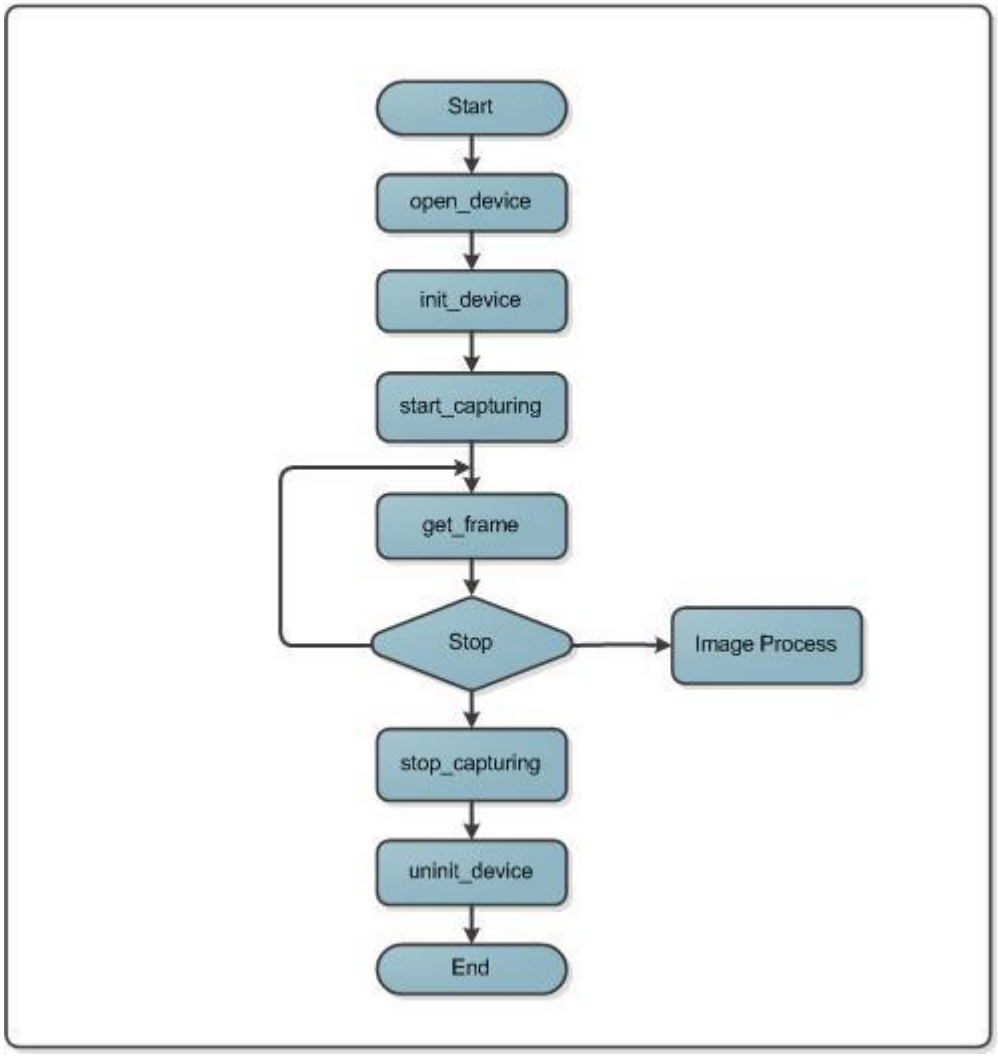

Figure 4-1 A Sample Work Flow of the UVC API

int open_device (const char *path)

Open a UVC device by its pathname.

Parameters:

path The pathname of device to open

Returns:

This function returns 0 on success and -1 on error.

int close_device ()

Close a UVC device.

Returns:

This function returns 0 on success and -1 on error.

int init_device (int width, int height)

Initialize the UVC device and set

Parameters:

width The width of image.

height The height of image.

Returns:

This function returns 0 on success and -1 on error.

int start_capturing ()

Start to capture images.

Returns:

This function returns 0 on success and -1 on error.

int stop_capturing ()

Stop to capture images.

Returns:

This function returns 0 on success and -1 on error.

int uninit_device ()

Finalize a UVC device.

Returns:

This function returns 0 on success and -1 on error.

int get_frame (void **frame_buf, size_t* len)

To Enqueue an empty buffer in the driver's incoming queue. Capture a frame and store in the buffer.

Parameters:

frame_buf The pointer of buffer where stores the frame captured by camera.

len The length of frame.

Returns:

This function returns 0 on success and -1 on error.

int unget_frame ()

To dequeue a filled buffer from the driver's outgoing queue.

Returns:

This function returns 0 on success and -1 on error.

int get_framesize (int *frame_width, int *frame_height, int *frequency)

Get the size of frame and the frequency.

Parameter:

frame_width The width of the frame.

frame_height The height of the frame.

frequency The frequency to get frame.

Returns:

This function returns 0 on success and -1 on error.

4.4 Implementation of HID API

int get_device_info (DEVICE_DESCRIPTION *ptr_devinf)

Get the information from a HID device, such as pid, vid and serial number. The parameter ptr_device is a pointer to type struct DEVICE_DESCRIPTION containing information about the HID devices attached to the system.

Parameters:

ptr_devinf a pointer to type struct DEVICE_DESCRIPTION containing information about the HID device attached to the system.

Returns:

This function returns 0 on success and -1 on error.

bool open_device (DEVICE_DESCRIPTION *ptr_devinf)

Open a HID device and store the information of the opened device in ptr_device, which is a pointer to type struct DEVICE_DESCRIPTIO.

Parameters:

ptr_devinf a pointer to type struct DEVICE_DESCRIPTION containing information about the HID device attached to the system.

Returns:

This function returns true on success and false on error.

int close_device (void)

Close the HID device, which is opened.

Returns:

This function returns 0 on success and -1 on error.

int get_send_buffer_len ()

Return the length of input report.

Returns:

This function returns the value of *Capabilities.InputReportByteLength*.

int get_receive_buffer_len ()

Return the length of output report.

Returns:

This function returns the value of *Capabilities.OutputReportByteLength*.

int command (unsigned char *out_array, unsigned short len, bool with_answer, unsigned char **answer)

Send a command to the HID device. The command consists of a set of bytes. For example, command {MSP,MSP_DEVIN} is used to get temperature of chip. MSP is defined as 0x4D and MSP_DEVIN is defined as 0x14. Then the temperature of chip will be returned and stored in the buffer that pointer answer refers to.

Parameters:

out_array	an array of char, containing the command sent to the device
len	the length of the array
with_answer	if there is an answer
answer	a buffer storing the date returned from HID device

Returns:

This function returns 0 on success and -1 on error.

int get_hardware_rev (void)

Get the value of hardware revision.

Returns:

This function returns the value of *DeviceHardwareRev*.

int read_from_device ()

Read an Input report from a HID device

Returns:

This function returns 0 on success and -1 on error.

int write_to_device (unsigned char * OutputReport)

Write an Output report to a HID device.

Parameters:

OutputReport	the data to send

Returns:

This function returns 0 on success and -1 on error.

5 Optris Project

5.1 Introduction

Optris GmbH is one of the leading innovative companies in the wide range of non-contact temperature measurement through infrared radiation since it established in 2003. The camera series optris PI is a kind of thermal imager, which can be used in many industry areas. The cameras are connected to a PC via USB 2.0 interface and can immediately be used after connection. It includes two logical USB devices: one USB video device (UVC) and one human interface device (HID). The UVC is a sensor, which provides row data and the HID controls processing. The corresponding software optris PI Connect displays the captured temperature data as a thermal image.

Optris PI Connect can only work under Windows so far. In order to better meet the demands of customers, a Linux version of optris PI Connect is required. This project is going to transplant PI Connect from Windows to Linux.

5.1.1 PI Series

The camera series optris PI comprises online thermal imaging systems with an outstanding state of the art cost-performance-ratio. The cameras are connected to a PC with an USB 2.0 and can immediately be used after connection.

The infrared cameras optris PI are based on a small uncooled bolometer (UFPA) with 160 x 120 pixels. They deliver thermal images in real time with a frequency of up to 128 Hz. Fast processes can be captured and stored as snapshots or video sequences. It is possible to detect smallest temperature differences at an object due to the very good thermal sensitivity of the cameras. With the help of BI-SPECTRAL technology of optris PI200, a visual image can be combined with a thermal image. Both images can be captured time synchronously.

In a ready to use mode, the cameras' weight is not more than 320 gram, including the lens and cable. They represent the smallest thermal imagers in the world (dimensions:

46 mm x 56 mm x 90 mm). In combination with a tablet PC the cameras are used as a mobile solution for preventive maintenance or construction thermography and cover the existing gap between portable infrared snapshot cameras and pure fixed devices.

5.1.2 PI Connect

The software PI Connect provide by the company without any additional costs. PI Connect is modern software with intuitive user interface that displays the captured temperature data as a thermal image. Under Windows PI Connect provides many extended functionalities to support temperature data analysis and documentation, such as radiometric snapshots, data with color information for standard programs, or text files including complete temperature information. But in this migration project, we just need to implement the basic function, i.e. display the captured temperature data as a thermal image.

5.2 Software Architecture

PI Connect is implemented in C++, which is a kind of object-oriented program language. Objects are discrete, independent, and loosely coupled; they communicate through interfaces. Actually, the software can be divided into three parts: User Interface Layer, Application Layer and Infrastructure Layer. The user interface layer can also be called view layer or UI layer; it defines exactly what is presented to the user. The application layer is responsible for calculating and analyzing images data. This layer is platform independent; it needn't be changed when software is transported from one operating system to another. The infrastructure layer is mainly responsible for communicating between software and camera; it includes two libraries, UVC API and HID API.

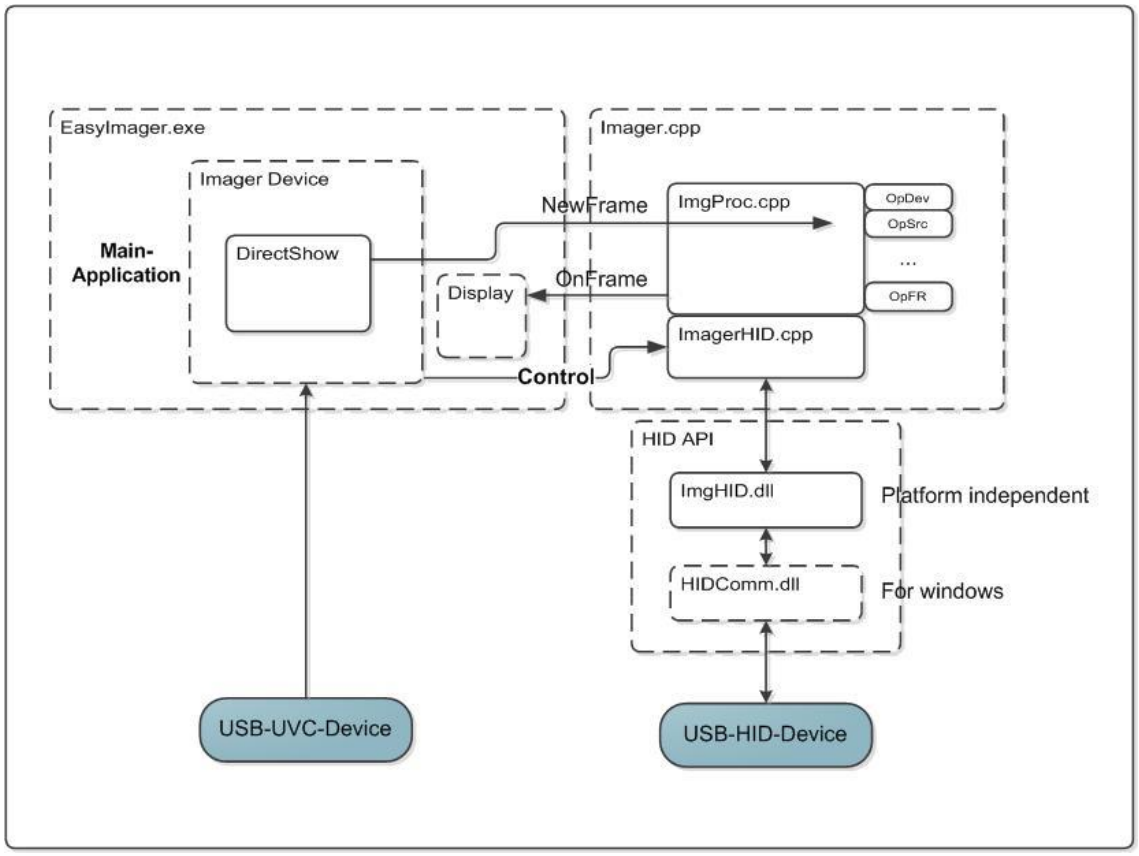

Figure 5-1 Architecture of PI Connect under Windows

Figure 5-1 represents the architecture of PI Connect under Windows. As shown, the camera includes two parts, UVC device and HID device. At first, images are captured by the UVC device and transferred to DirectShow, in which the image data will be transformed and stored in buffer. While a frame is ready, the function *NewFrame()* will send a message to class Imager. The frames keep staying in buffer until five frames are captured. Then program reads all of these five frames at one time to do further calculation and optimization, and finally return only one frame. The process of calculation and optimization is the core part of PI Connect. During this process it needs to use some parameters, which are detected by HID device. HID APIs are implemented to get this job done. HID APIs are a set of interface that access and control HID device. At last, when an image is ready, it calls function *OnFrame()* and the image will be displayed on the screen.

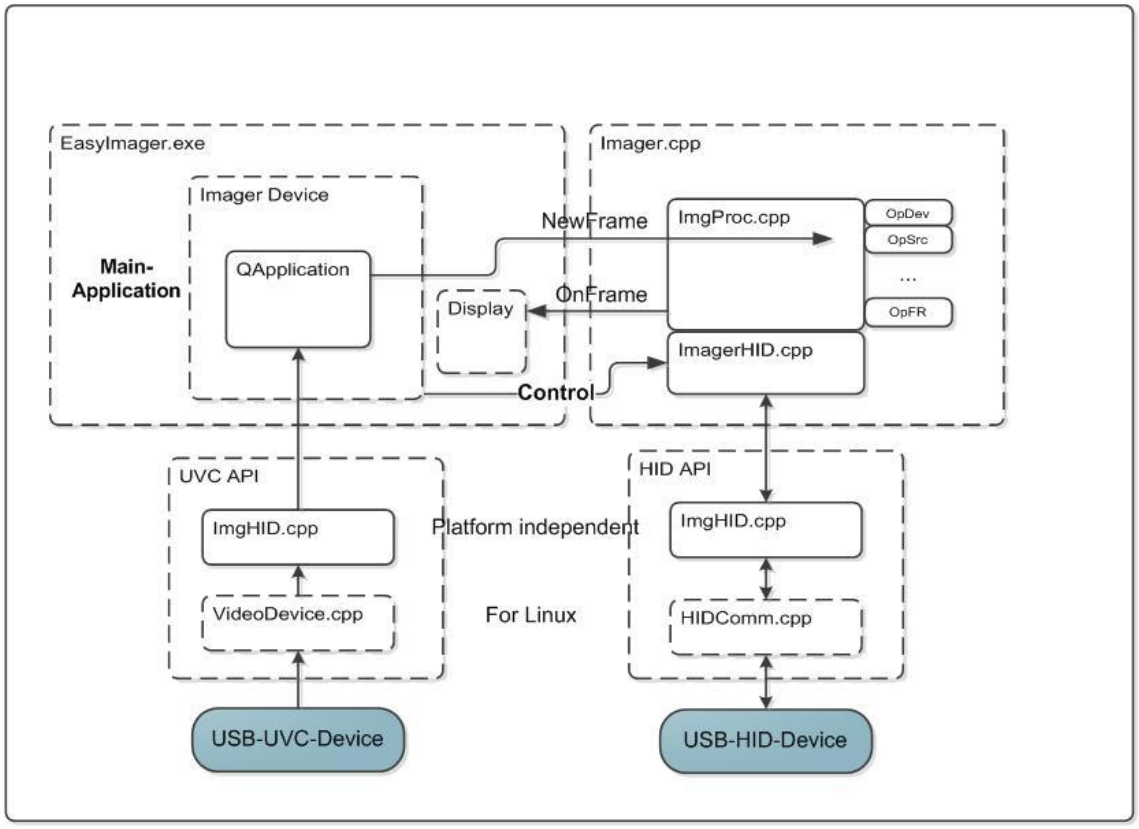

Figure 5-2 Architecture of PI Connect under Linux

Figure 5-2 represents the architecture of PI Connect under Linux. As we see there is no significant change in the structure of the program, because the OO design pattern allows the codes to be reused. Except the part communicating with the operation system, other parts are all platform independent. In this program the only part needs to communicate with operation system is the camera, i.e. UVC device and HID device.

As we know DirectShow is a framework that is provided by Microsoft. It can't be used under Linux. So this module must be replaced by a new one. Under Linux UVC API is used by UVC device to communicate with system. On the other hand, a new HID API will replace the one that is used under Windows. As same as the API under Windows, the new HID API is also use layered designing concept, but extend the capability.

Otherwise Qt Framework is used in this project. Qt Framework is a C++ based cross-platform development framework. It provides a set of GUI Components for building both native look & feel and custom UI's for the desktop environment. It is good news for the developers, who are not familiar with UI design. They can focus on the devel-

opment of business logic. This is also the main reason why we use Qt Framework here. As in figure 5-2 shown, Widget is one of the applications of Qt Framework. Certainly Qt is so powerful that not only can be used to design UI.

5.3 Application of Qt

5.3.1 GUI

Qt provides a range of standard user interface elements, called widgets, for each supported platform. Widgets can be used as containers for other widgets, as windows, and as regular controls that the user interacts with.

Widgets are the basic building blocks for graphical user interface (GUI) applications built with Qt. Each GUI component (e.g. buttons, labels, text editor) is a widget that is placed somewhere within a user interface window, or is displayed as an independent window. Each type of widget is provided by a subclass of *QWidget*, which is itself a subclass of *QObject*.

QWidget is not an abstract class. It can be used as a container for other widgets, and it can be subclassed with minimal effort to create new, custom widgets. *QWidget* is often used to create a window inside which other *QWidgets* are placed.

After introduction to Widget's functionality and basic principles, let's have a look at how it's applied in this project. At first, we have to know that a typical *main()* function in main.cpp looks like this:

```
#include <QtGui>

// include header files for application components.
//…
int main (int argc, char *argv[ ])
{
   QApplication app (argc, argv);

   //Set up and show widgets.
   //…
    return app.exec ( );
```

```
}
```

First, a *QApplication* object is constructed, which can be configured with arguments passed in from the command line. After the widgets have been created and shown, *QApplication::exec()* is called to start Qt's event loop. Control passes to Qt until this function returns. Finally, *main()* returns the value returned by *QApplication::exec()*.

Then we just need to add the code of Widget to the appropriate position. For example, we add a main window and set a default size. The code looks like this:

```
#include <QtGui>

// include header files for application components.
//…
int main (int argc, char *argv[ ])
{

   QApplication app (argc, argv);

   MainWindow w;
    w.resize (320,240);
    w.show ( );

    return app.exec ( );
}
```

Class MainWindow is a user defined class for providing a main application window. A main window provides a framework for building an application's user interface. The code looks like this:

```
#include "MianWindow.h"

MainWindow::MainWindow ()

{
      createMenus ( );
      createActions ( );
      w = new QWidget;

      label = new QLabel ( );
      layout = new QVBoxLayout ();
      layout -> addWidget (label);
      w -> setLayout (layout);

}
```

First, we add a menu in menu bar. This is a pull-down menu and the action items of the menu list all the devices which are connecting with the system. Then we create the widgets we want in the layout. We create a *QVBoxLayout* object and add the widgets into the layout. Finally, we call *QWidget::setLayout()* to install the *QVBoxLayout* object onto the widget.

5.3.2 Thread

Since computers came out, we have been looking for all possibilities make the program to run faster and better. Multithreading is one of them. In computer science, a thread is the smallest unit of processing that can be scheduled by an operating system. A thread is a lightweight process. Multiple threads can exist within the same process and share resources such as memory, while different processes don't share these resources.

In Qt, *QThread* provides platform-independent threads. A *QThread* represents a separate thread of control within the program; it shares data with all the other threads within the process but executes independently. Instead of starting in *main()*, *QThreads* begin executing in *run()*.

In this project, *QThread* is used to control UVC device. With thread the camera can continuously capture images, while the main program is processing the images that are captured before. The codes below describe the procedure how a UVC device run in a thread works. When a frame of image is captured and stored in buffer, function N*ew-Frame()* will be called. This thread will not stop, until the UVC device is closed.

```
void Mythread::run ()

{
    uvc -> OpenDevice ( );
    uvc -> InitDevice ( );
    uvc -> Start ( );
    while (Close == false) {
        uvc -> GotFrame (buf);
        if(SGBuffer[0].data&&SGBuffer[0].Timestamp&&SGBuffer[0].TimestampMedia)
        {
            memcpy (SGBuffer[0].data, buf, Width*Height*2);
            *(SGBuffer[0].Timestamp) = timeGetTime ( )*1E4;
            (*SGBuffer[0].TimestampMedia)++;
            NewFrame (0, &SGBuffer[0]);
        };
        uvc->UngetFrame ( );
    }
```

```
        uvc -> Stop ( );
        uvc -> CloseDevice ( );
}
```

6 Conclusions and Feature Work

6.1 Summary

This thesis comes from a migration project of Optris GmbH. Optris PI is a kind of thermal camera, which is a product of Optris GmbH. PI Connect is the corresponding software works with Optris PI. PI Connect could only execute under Windows. Now they want to develop a new version for Linux. In order to make the migration faster and better, I design a light-weight framework under Linux. This is an OO-framework that means all components in this framework are object-oriented. It can be reused and extended easily. The design of this framework is based on MVC model, which provides a clearly structure and separated responsibility for each part. In addition to the framework, two libraries, UVC API and HID API, are developed to help to access and control devices, which are widely used in practice. Finally the new framework and libraries are applied in the migration of PI Connect.

6.2 Feature Work

Although the new framework and libraries work well in the migration project of PI Connect, we wish that our framework can be used in more applications, even more platform. We have two objectives in the future work. On one hand, improve the framework for more applications. The framework will definitely support more devices and provide more libraries, which can help to process image data. On the other hand, extend the framework to more platforms. With the increasing popularity of tablets, more and more applications will support tablet, so will our framework.

7 Reference

[1] Weinand, A. & Gamma, E. & Marty, R. *ET++—an object oriented application framework in C++*. ACM SIGPLAN Notices - Special issue: 'OOPSLA 88 Conference Proceedings' Volume 23 Issue 11, Nov. 1988 Pages 46 – 57.

[2] Riehle, D. *Framework Design: A Role Modeling Approach*. Zurich: Swiss Federal Institute of Technology, 2000.

[3] Chatterjee, A. & Maltz, A. *Microsoft DirectShow: A New Media Architecture*. SMPTE Mot. Imag J. December 1, 1997 Vol. 106 No. 12 Pages 865-871.

[4] David N. Gray & Hotchkiss, J. *Modern languages and Microsoft's component object model*. Communications of the ACM, Volume 41 Issue 5, May 1998, Pages 55 – 65.

[5] Ni, CQ. & Chen, Y. *Design of USB HID Device Based on Single Chip C8051f340*. Xi'an University of Science and Technology, 2007.

[6] *Microsoft MSDN Library*. October 2001.

[7] Corry, C. & Mayfield, V. & Cadman J. *Waite Group's COM/DCOM Primer Plus*. Sams Publishing, December 1998.

[8] Polinger, A. *Developing Microsoft Media Foundation Applications*. Microsoft Press, October 2011.

[9] Gordon, R. & Talley, S. *Essential JMF: Java Media Framework.* Prentice Hall PTR, January 1999.

[10] Bloch, J. *How to design a good API and why it matters.* Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Pages 506-507

[11] Sun Microsystems, Inc. *Java Media Framework 2.0 API Guide.* November 1999.

[12] Bach, Maurice J. *The Design of the UNIX Operating System.* Prentice-Hall, 1986.

[13] Corbet, J. & Rubini, *A. Linux Device Drivers, 2nd Edition.* O'Reilly Media, June 2001.

[14] Rischpater, R. & Zucker, D.*Working with the Nokia Qt SDK.* Beginning Nokia Apps Development, 2010, Part II, pages 39-57,

[15] USB Implementers Forum. *Universal Serial Bus Device Class Definition for Video Device*. June 2005.

[16] Fayad, Mohamed & Schmidt, Douglas. *Object-Oriented Application Frameworks.* Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, Vol. 40, No. 10, October 1997.

[17] Glenn E. Krasner & Stephen T. Pope *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.* ParcPlace Systems, 1988.

[18] Chen, L. & Pei, H. L. *Programming for the Capture of Image on Video4Linux2*. College of Automation Science and Engineering, South China University of Technology, Guangzhou, 2009.

[19] Blanchette, J. & Summerfield, M. *C++ GUI programming with Qt 4*. Prentice Hall Press, 2008.