Video Editing with Single Responsibility Principle

Master of Science (Tech.) Thesis
University of Turku
Department of Computing
Software Engineering
2023
Aleksi Papalitsas

UNIVERSITY OF TURKU
Department of Computing

Aleksi Papalitsas: Video Editing with Single Responsibility Principle

Master of Science (Tech.) Thesis, 53 p.
Software Engineering
May 2023

Non-linear video editors are typically extremely big and complicated programs that are able to do many complicated operations for all kinds of video formats. A professional video editor should be able to encode and decode all kinds of formats, cut and join video clips, apply filters to those videos (including color correction), analyze those videos in real time (histograms and tracking), do audio editing with digital signal processing, render 3D graphics and show them in real time, render texts, animate all effects and texts with their properties (position, opacity, etc.) with keyframes, and finally a professional video editor should be able to play these all clips in real time while showing the current output of the video editing. Of course, there is also rendering, but encoding was mentioned earlier and when the video is already processed in real time, the rendering should be trivial.

The main challenge, which this thesis attempts to solve, is that these applications are extremely complex and contain multiple components that could be applications on their own. If one component crashes, the whole program crashes. The exact point of single responsibility principle (can be seen as part of UNIX philosophy in this case) is to have applications that do one thing and do it well. This means that all the parts of the modern complex video editor would be divided into small modular parts that function on their own but are able to talk with each other. This also means that the end user does not lose all the progress that has been made, which in turn will make the video editing much easier and smoother experience. This kind of modularity and its possibilities are researched in this thesis.

Keywords: Video Editing, Single Responsibility Principle, UNIX, Web, Decentralized

Contents

1 Introduction

When video editing was first introduced, it was done by literally cutting and gluing films together. It was extremely time consuming and tedious work and what made it harder was that the film can be only accessed sequentially. This meant that a clip can only occur once in the final product and also skipping to different parts of video material was harder since the film had to be cued to the right position. This kind of editing is nowadays called linear video editing. [1]

What mostly replaced linear video editing was digitization, which allowed the video to be stored in digital non-destructive format, where the desired clip can be accessed in any part of the full material and also can be copied to appear multiple times in the final product. This kind of editing is called non-linear video editing.

There are multiple different brands of video editors, however only a few of them have enough features that they can be used by power users — let alone professionals. As video editors now need more and more features, the bar for video editor to be considered to be an industry standard is quite high.

Currently there are only handful of video editors that work in a way that they can be considered to be an industry standard. Adobe® Premiere Pro and Avid Media Composer are the ones preferred by professionals. [2] [3] There is also Blackmagic Design's DaVinci Resolve, which was previously only used for color correction, but after Australian digital cinema company Blackmagic Design Pty Ltd. bought da Vinci Systems in 2009, the scope of the software has been extended for more general-

purpose non-linear video editing. [4]

1.1 Video Editing Utilities and Categories

There are many programs that can edit videos, but are not necessarily video editors. For example, Nuke and its open source counterpart Natron are able to edit videos, but they are not video editors in the sense that they operate on much lower level and are not optimized for large scale editing. These kinds of editors are called compositors since the idea for these is produce more fine-grained post-production visual effects and not necessary edit and cut videos. While there are some things that both video editors and compositors do, there still is a fine line between those two. Figure 1.1 illustrates what programs are used for what purposes. Both proprietary and open source solutions are provided. The video editing utilities are categorized by these characteristics.

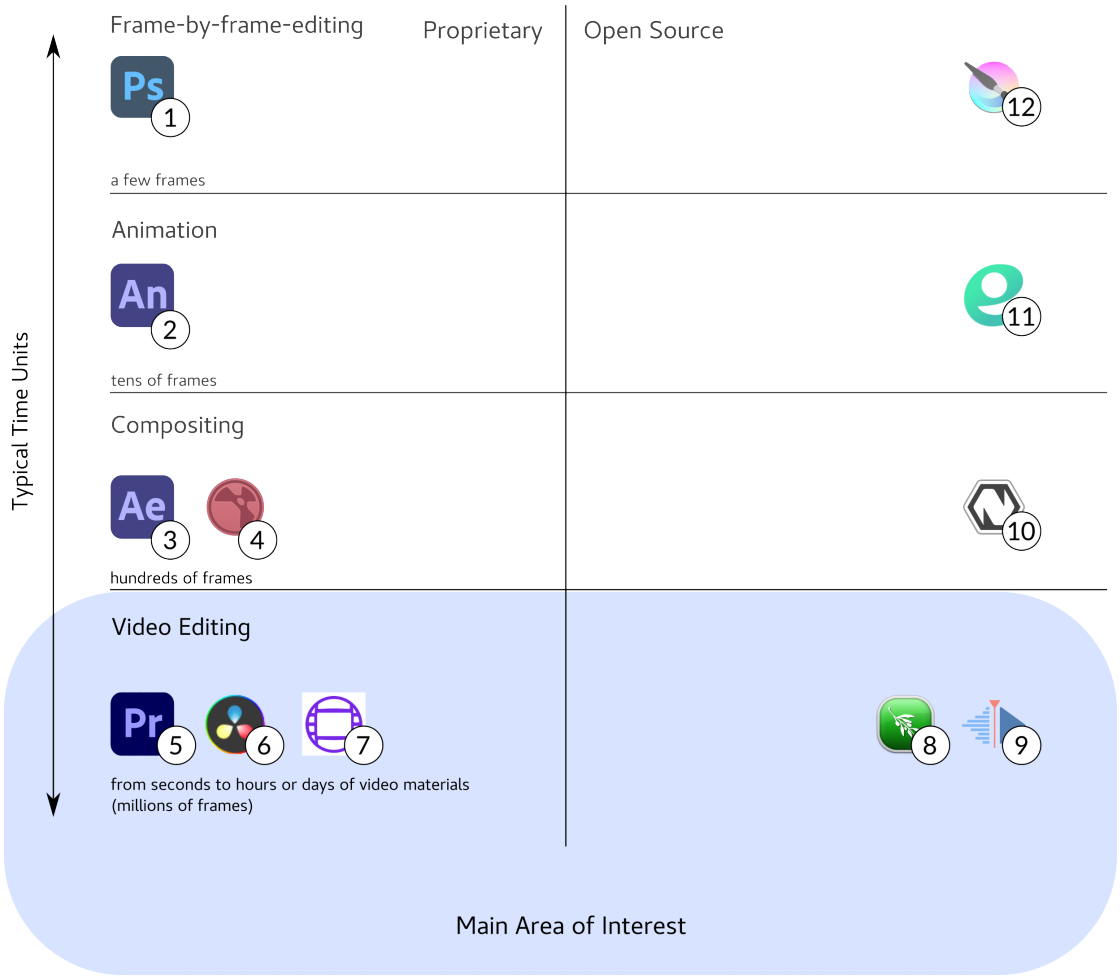

Figure 1.1: Common Open Source and Proprietary video editing utilities in their own categories by their typical time units. Video editing utilities in the picture numbered: (1) Adobe Photoshop, (2) Adobe Animate, (3) Adobe After Effects, (4) Nuke, (5) Adobe Premiere, (6) DaVinci Resolve, (7) Avid Media Composer, (8) Olive, (9) Kdenlive, (10) Natron, (11) Enve, (12) Krita.

As seen in the picture (Figure 1.1) Adobe® has products for every role for every video editing category and these are usually bundled together. There are also free open-source alternatives, but they are not often as stable or feature-rich as the proprietary ones. Especially in the video editing the open-source options Olive and Kdenlive are not as stable and feature-rich as their proprietary alternatives Adobe® Premiere, DaVinci Resolve and Avid Media Composer. There is FFmpeg as a free open-source option, which is both stable and feature-rich, but that program only has a command line interface and not a proper graphical user interface.

1.2 Research Questions and Topic

The main topic of this thesis is to discuss of the possibility to build a video editing program that is able to make different command line based programs work with each other. Not only limited to the command line-based programs, but the possibility of running programs with a GUI provided by VST-plugins is also explored.

There are four main research questions in this thesis:

- **RQ1:** How to improve video editing software modularity?
- **RQ2:** How to improve video editing software stability?
- **RQ3:** How to make a user interface, that allows the end user to use different modules in one program?
- **RQ4:** How to integrate collaboration with different people to the user interface?

The aim is to make the individual components of the video editor more modular and secure. This kind of implementation opens new possibilities, like building web-based front-end, which allows multiple people to collaborate with the program.

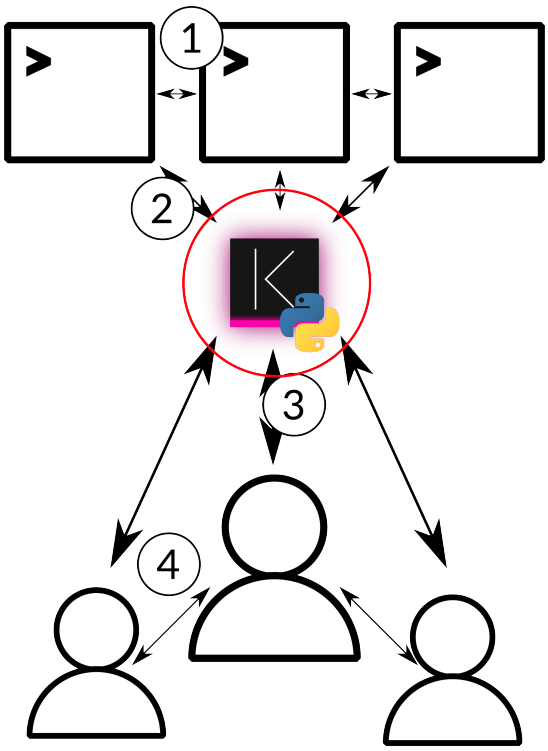

Figure 1.2: (1) Communication between the programs, (2) Communication with the central program, (3) User interface, (4) Collaboration. Circled box with letter ”K” is the desired program to be researched

This means that there is a program that manages all the programs that process the data either with command line interface or python interface and it also manages the data moving between these programs. Practically this means either separate video files of each phases or piping the raw data from program to another or storing each frame as individual images (png-, jpg-, webp-files).

Figure 1.2 is a graph illustrating the architecture of the desired program. Here the circled logo with letter ”K” and python logo represents our program that communicates with the end users and different processes that process the video. In

other words the program just acts as a (web-based) front-end for the video editing frameworks that actually process the video.

1.3 Thesis Overview

The thesis follows a typical problem-solution-evaluation-pattern, which means that first the problem is introduced, then the solution for this problem and then there is evaluation, which discusses how well the solution solves the problem.

This thesis first introduces definitions and research questions in Chapter 2, these definitions will be used multiple times in this thesis in later chapters. After that current existing solutions and their problems are introduced in Chapter 3. The chapter firstly attempts to explain the flaws of the current solutions and why this new kind of solution would be needed. In the later sections also the open source components are introduced, which can be used as parts of the modular video editor. Typical user groups are introduced for different video editors in Chapter 4. The chapter introduces different user groups that use video editors, so it can be later evaluated, how fitting this kind of video editor would be for different user groups. After that the solution is presented in two different chapters. First there is Chapter 5, which is more theoretical and discusses the architecture and concept of the solution. The latter Chapter 6 is chapter that shows the more practical and concrete details of the implementation. After all this solution is evaluated in Chapter 7, the research questions will be answered, and finally the conclusion is drawn in the final chapter.

2 Background

This chapter contains definitions, applications, formats and standards crucial to this thesis and the video processing discussed in this thesis. These will be later referenced to in this thesis.

2.1 Definitions

2.1.1 Non-linear Video Editor

Non-linear video editor is a video editor that allows video material to be edited in random access-manner and non-destructive copying rather than traditional linear video editing that used films and tapes that only allow sequential access to the video material and physical film was cut, which caused damage to the film itself. [5]

2.1.2 Single Responsibility Principle

Historically the definition of single responsibility principle (SRP) has been described in varying ways, but the final form of it is typically described as: ”A module should be responsible to one, and only one, actor.” Typically this means that a class should have only one reason to change. [6] For example, if a filter needs to be changed, the whole module that containing decoders, encoders, transitions and other components does not need to be changed. Only the module containing the filter needs to be changed.

There is similar principle compared to single responsibility principle. The UNIX philosophy, that defined how the programs in the UNIX operating systems were written. The UNIX philosophy is defined by Douglas McIlroy in the Bell System Technical Journal from 1978 [7]:

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new ”features.”

2. Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.

3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.

4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.

Unix philosophy can be seen as much stricter version of single responsibility principle. The SRP states that one module should be responsible for one actor, but UNIX principle states, that one program should do one job and do it well among the other requirements.

In this thesis the single responsibility principle can be seen as an intermediate position between these two philosophies. The main principle here is that these modules are isolated from the core program and the modules are built in a way that, if one fails or crashes, it does not crash the entire program.

2.1.3 Cathedral vs Bazaar Models

Eric S. Raymond mentioned the Cathedral and Bazaar models in his book The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. In nutshell these two models mean that cathedral model means the software is created with restricted number of developers and the source code is released when the software is released in contrast to bazaar model where the source code is open and available for everyone, and anyone can contribute to the development of the software. [8]

In this thesis we focus rather on the software ecosystem than developing the software. Meaning that the components of the software can be developed with the cathedral model, but as the software itself follows single responsibility principle, which makes it easier for other people to extend the functionality of the software ecosystem. This makes the software ecosystem itself more bazaar-like.

2.1.4 Transaction

Transaction processing is typically used in databases. What transaction means is that the processing is divided into indivisible operations called transactions. Typically in databases this means one query to a database, where for example all instances with expiration date expired get removed. [9]

Since this thesis is about videos and not databases, the definition of transaction is similar, but slightly different. The transaction in this thesis means that video (and/or audio) as data gets processed into another form of data. This can mean compressing or decompressing, filtering, compositing with multiple tracks or combining all tracks into one file. The main principle is that the operation is atomic, meaning it cannot be divided into smaller operations.

These transactions form the backbone of this thesis and typically these transactions are the tasks that different modules end up processing in their processes.

These transactions will be referenced later in multiple chapters, so this principle is crucial to understand.

2.2 Applications and Formats

2.2.1 Make

Make is a command line program usually found in UNIX-like systems that takes a recipe called ”Makefile”, that contains all the recipes to make the files wanted by the Makefile. Makefile simply lists targets and its dependencies and how to make the desired file from those dependencies. [10]

Make is usually used for compiling sources, but it could be technically used for everything that uses command line-based processing. It can be also used to render videos with FFmpeg, but it makes no practical sense to do that since there is no acceptable video editing graphical user interface to see the video while editing and all the timestamps for each cut have to be written manually.

The aim of this thesis is exactly to design such graphical user interface that is able to process video files like *make* would, but just in a much more user friendly way. There could be also better collaboration options. These kind of details will be discussed more specifically in chapters 5 and 6.

2.2.2 Yuv4mpeg

Yuv4mpeg is video stream format typically used in pipe-based video processing software. What this practically means, is that when video gets processed and it cannot be processed in one process, other process is going to need the data as lightly compressed as possible. This practically standardizes a protocol to send video data between different processes. [11]

2.3 Extendibility and Existing Plugin Standards

2.3.1 OpenFX-plugins

OpenFX is an open industry standard for visual effects plug-ins. It is supported by most professional video editors (like Adobe Premiere, Sony Vegas Pro and DaVinci Resolve) and there is also a toolkit for Python that provides OpenFX-support called TuttleOFX. OpenFX simply provides an interface for video editor to add new video effect plugins. [12]

2.3.2 VST-plugins

Virtual Studio Technology or shortly VST-plugins are used to process audio effects. They provide a graphical user interface that is typically embedded into a digital audio workstation. They can be used by video editors and often are also supported by video editors. VST-plugins are shared libraries that contain processing functions and a user interface. This also means that the program loads the code as a part of the program which means that if the plugin has faulty code inside it that will crash the plugin, its host will also crash (as demonstrated by Figure 2.1). If the program does not isolate the host properly (isolating properly means that the host process is different process to the main program) the entire program will crash. [13]

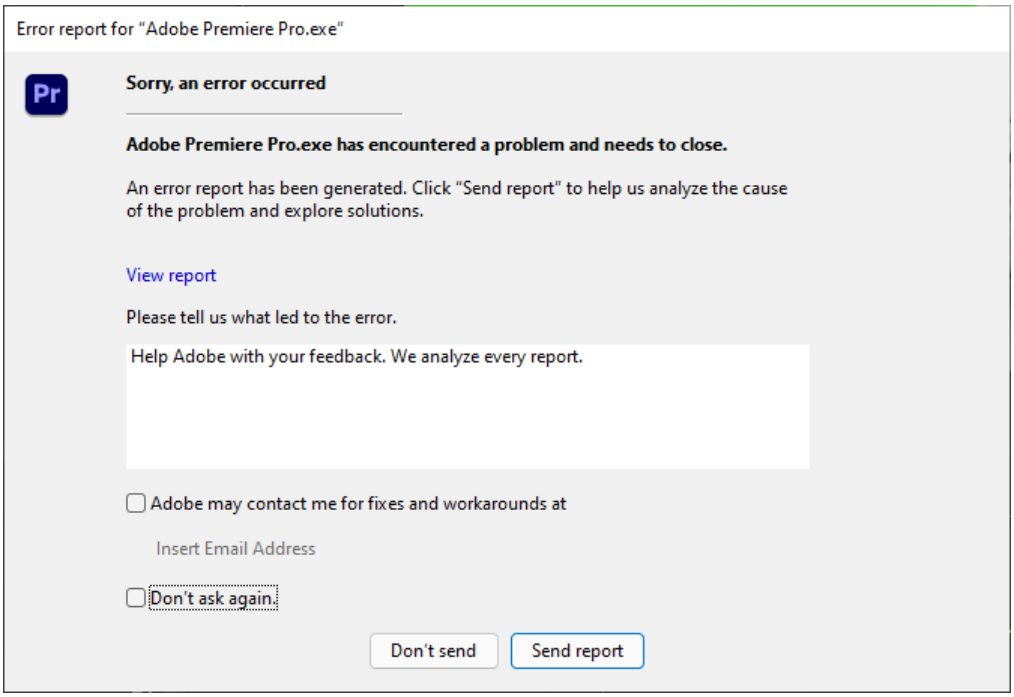

Figure 2.1: VST-plugin crashing the whole program with a segmentation fault

There are also several VST-host libraries written for Python, like RenderMan [14], Dawdreamer [15], cython-vst-loader [16] or pyvst [17]. This would make it easier to handle those kind of plugins. However, since the plugins tend to be either 32-bit or 64-bit, there would be a need for two python versions, which is not acceptable. So, there should be one small program (with both 32-bit and 64-bit versions) that loads the VST-plugin and communicates with the python-server.

3 Current Solutions

3.1 Challenges of Current Commercial Industry Standard Video Editors

There are practically two industry standard video editors: Adobe Premiere Pro and Avid Media Composer.[2] [3] While both of these fill the common needs for most people, they also have many problems that cannot be fixed as the problems are part of the architecture. These problems will be discussed in this chapter.

3.1.1 Monolithic Design

Current solutions for video editors are quite monolithic, which is partially understandable since many of those features are necessities required by professionals. Those features can be however externalized outside the program making it modular and much easier to manage. This would make the program more stable and extendable. Plugins partially allow these kind of extensions. But because those are loaded to be part of the process rather than separated into different (sandboxed) process, these extensions tend to make the editor even more unstable. The monolithic design also tends to make the video editor much slower as all the components must be loaded before the program can start.

3.1.2 Limited Hardware and Platform Support

Some video editors require very specific support from video cards and their drivers. For example DaVinci Resolve uses OpenCL, which has only limited support on Linux with open source drivers. Not only that, but also some versions of NVIDIA drivers on Windows can cause DaVinci Resolve to dysfunction. It is often recommended to use NVIDIA Studio drivers instead of NVIDIA Game Ready drivers on Windows. This is stated for example on NVIDIA's Studio driver's web page. [18]

Video editors can be also work only on certain platforms. Final Cut Pro only works on Mac operating systems. Sony Vegas Pro or nowadays Magix Vegas Pro video editors tend to work only on Windows operating systems. There are also video editors that work on multiple platforms. But they almost never work on Linux.

3.1.3 Instability

Because of the monolithic design of video editors, they often tend to have multiple complex component. Quality of those components may vary, which means that some components may cause the program to crash. When the whole program to crashes and all the progress done in the video editor is lost, if the user has not saved the project. This is a major inconvenience for the end user and in worst cases can make the video editor unusable.

Typically these crashes are caused by bugs at the core of the video editor or plugins loaded into the main process without isolating it into a different process. Typical bug that causes program to crash is when a buffer is allocated and then used and freed and then used again after it is freed. This kind of error typically occurs when video or audio data is accessed in another thread and the data is not managed properly.

Because the data is no longer allocated the data should not be accessed and the operating system has to stop the process, since the programs tries to access

area that should not be accessed. This is known as access violation in Microsoft Windows-based systems and segmentation fault in UNIX-based systems. In some older operating systems, that did not support protected memory, these kind of errors could have caused the whole operating system to crash. [19] [20]

Besides bugs other reasons for instability include running out of RAM (random access memory) or running out of CPU (central processing unit) time. The former is a resource that can only be managed by the user by simply buying more ram or a better computer or using smaller files. The latter however may be a bug or bad design. Some times heavy processes are executed in the main thread, which causes the whole program to freeze until the operation is completed. This can be avoided by having another thread do the processing or better would be another process, which could also prevent from crashes caused by plugins. Heavy processes might still freeze the computer if there is not enough CPU-power, which might cause resource starvation on operating systems that do not have a good scheduler. This is however relatively rare with modern operating systems and multi-core processors. [21]

3.2 Existing Components in POSIX-compatible Systems

There are several open source options for video editing as well. But they are often lacking features that exist on commercial industry standard video editors and are often also quite unstable. There is an open source video processing program is FFmpeg, which was discussed in Section 3.2.1. However, that program does not have a graphical user interface, it is completely a command line-based program. There are also open source graphical user interface-based programs as well (such as Kdenlive, Olive, Shotcut or Openshot), but they usually are unstable or lack the

necessary features required by professionals (see Section 4.3) or both.

There is also Natron Compositor. However as its name suggests, it is not a video editor, but rather a compositor, which has a different purpose as mentioned in Chapter 1.

3.2.1 FFmpeg

FFmpeg is command-line utility that is essentially a Swiss army knife for video editing, as it allows multiple operations (like decode, encode, transcode, mux, demux, stream, filter and play) to be done for videos and is probably the most advanced video editor there is. [22] The only problem for the end user is that unfortunately the whole program is command-line based, making it very user unfriendly as the editor cannot even preview the video material while editing. Some of the commands are also quite hard to remember which definitely would not help with the user friendliness.

3.2.2 MLT Framework

MLT-framework is a multimedia framework, which implements the most common functionalities of a video editor and also serves as a backbone for video editors such as Kdenlive, Shotcut and Openshot. While the framework itself has a lot of functionality, it suffers from stability issues and the video editors built with tend to be quite unstable. [23] [24]

Luckily there also is a command line interface for this framework called melt. It allows access to MLT- frameworks functionalities via command line, which means that it could serve as an extension to another programs including modular programs utilizing command line interface. [23] [24]

3.2.3 GStreamer

GStreamer is a multimedia framework for UNIX-systems [25]. Compared to MLT framework it tends to be much general purpose API for a multimedia processing as it is commonly mostly used to play (decode) videos, so they can be showed in the application.

GStreamer also has a commmand line interface called ”gst-launch”. It allows access to GStreamer's functionalities and allows to do quite similar things compared to FFmpeg. This means that GStreamer could work as an alternative for FFmpeg.

The reason why FFmpeg needs an alternative is, because FFmpeg has patent-related issues in some countries. This is because FFmpeg has multiple patented codecs and formats implemented in it and FFmpeg community has not paid any royalties for those patents. This is because those patents are not valid in the country where FFmpeg's development is hosted. However, using FFmpeg in countries like United States can be a challenge, where these patents are still valid. [26]

3.2.4 TuttleOFX, Vapoursynth and Much More...

In this section some additional existing tools are introduced. These tools can be used as a part of a modular video editor and therefore provide an extension for such video editors.

TuttleOFX is a OpenFX-host library for python, as described earlier. It can be used to implement support for OpenFX-standard. However, since TuttleOFX has not been maintained for a while, its Windows-version still seems to use Python 2, which might cause compatibility issues as Python 2 is no longer supported. [27]

Vapoursynth is a video processing framework written in python. It is inspired by AVIsynth, which is a domain specific language for video editing. Vapoursynth allows similar capabilities, but with Python-code instead of domain specific language. [28]

Glaxnimate is an animation framework and also a program to create animations. Glaxnimate also has a python interface to render the animations as individual pictures or convert them to another animation format. There is also another animation program called Enve. That does not however have a python interface nor a command-line based interface (as of the time of writing), so integrating it to our program would be much more challenging. Enve has more features than Glaxnimate (like blur effects), which would make it more desirable as a back-end. Luckily Glaxnimate and Enve both support Animated SVGs (SMIL) so they can process each others files, but Glaxnimate cannot render some of objects and elements like Enve can. [29]

4 Practical User Scenarios

Video editors are used by different users, who have different needs for video editing. This varies from home video editing to professional video editing. The point of this chapter is to highlight the requirements of each user group, so they can be later evaluated in Chapter 7 to test how well the solution would fit to the needs of each user group. This chapter and Chapter 7 together will answer to research question 3 (RQ3): How to make a user interface, that allows the end user to use different modules in one program?

4.1 Typical Home Video Editing

This can be seen as the bare minimum what even the simplest video editor should be able to do. These standards were practically set by Windows Movie Maker, especially the Windows XP -version of it, which set the base standard for many video editors. As Windows Movie Maker came bundled with Windows, Windows XP -version practically set standard for bare minimum video editor as it was the most used operating system until 2012. [30]

In this thesis we define the crucial features as criteria for a basic video editor:

- Decoding and Encoding different video formats
- Cutting clips and placing them into the timeline
- Simple transitions

- Adjusting volume
- Adding texts and images
- Previewing the result in real time

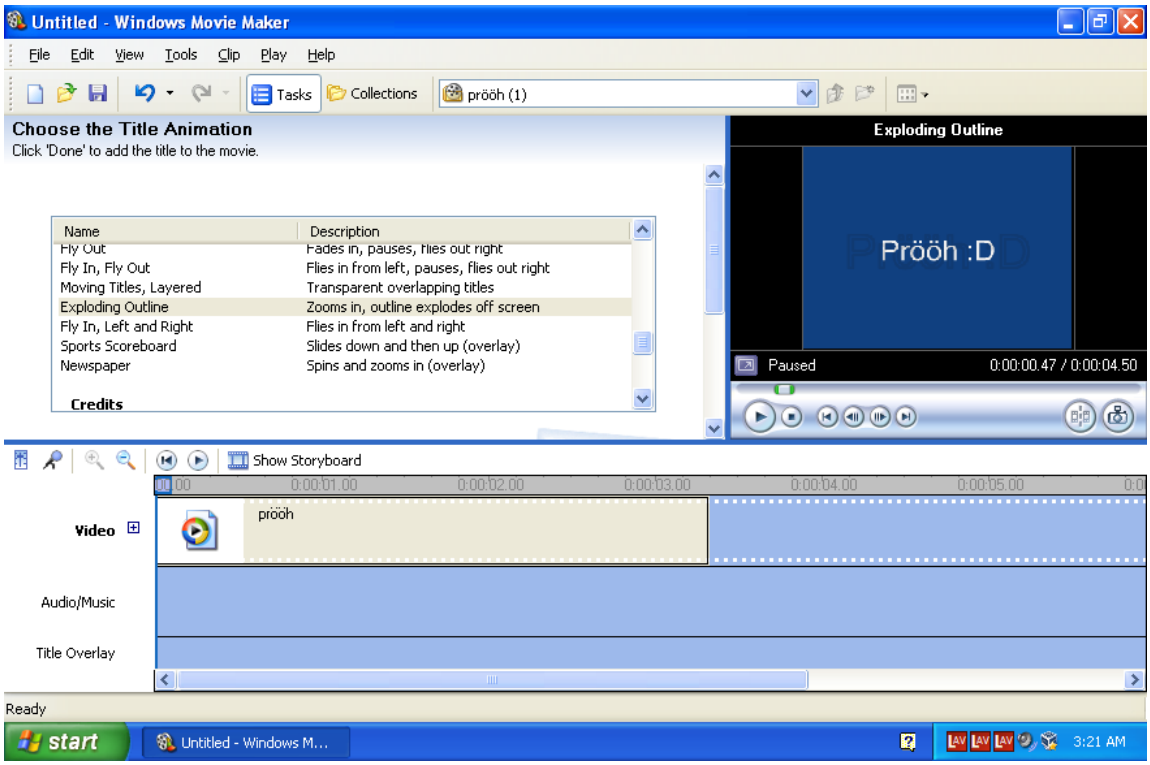

Figure 4.1: Windows Movie Maker was bundled with Microsoft Windows-operating system since Windows Me. This screenshot is taken from Windows XP SP3 -version.

4.2 Video Editing for Education

Educational use can be seen as an extension to basic video editing. In addition to basic video editing for educational use we have defined following requirements based on feedback we have received:

- Adding subtitles to video. Possibly also automatically create those with machine learning.
- Switching between different angles. (For example during lectures there is a camera in the lecture hall and then there is the view where there are only slides)
- Somewhat advanced audio processing such as normalization or noise removal should be added as the volume levels and audio quality can vary in those videos a lot.

4.3 Professional Video Editing

Professional video editing can be seen as the largest set of features that a video editors should offer. Only a handful of video editors offer those and practically Adobe Premiere and Avid Media Composer are currently the industry standards. [2] [3]

Here are the things a professional video editor should be able to do:

- Apply filters to those videos (possibly support OpenFX-extensions)
- Color correction
- Analyze videos (histograms and tracking)
- Ability to do audio editing with digital signal processing (should be extendable with VST support)

- Ability to render 3D graphics and show them in real time
- Render texts, animate them with all the effects (position, opacity, etc.) with keyframes
- Real-time preview for all aforementioned features

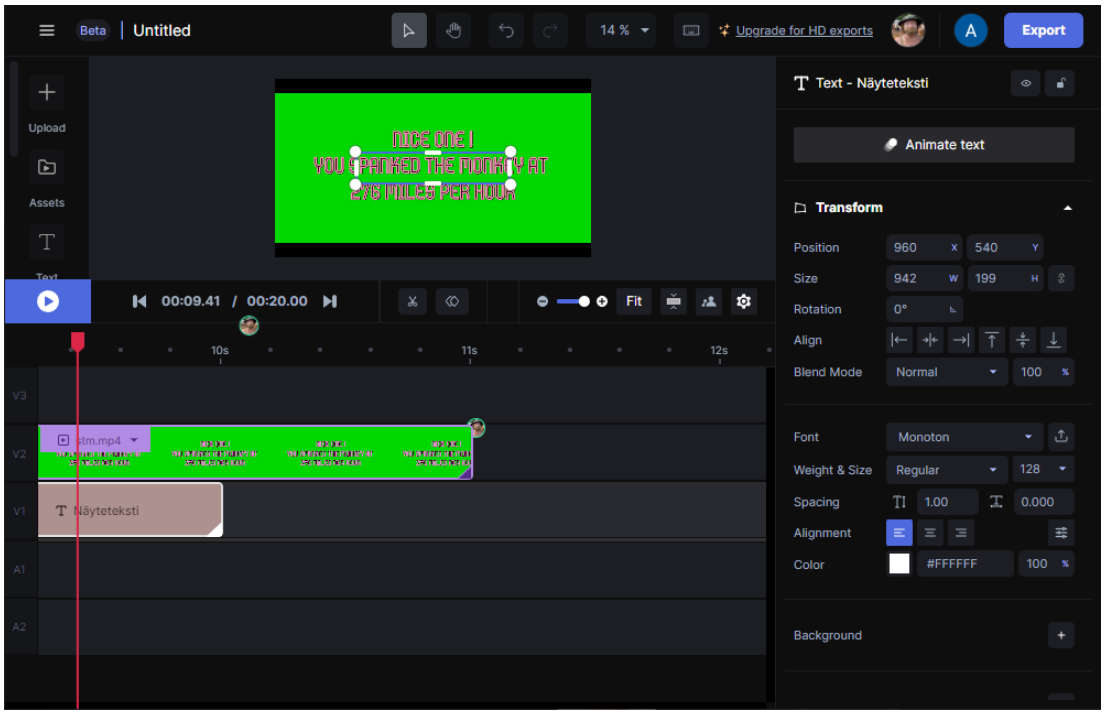

Figure 4.2: Runway ML video editor running in browser demonstrating collaboration features. Playhead position of another user can be seen in the picture.

4.4 Editing with a Team

None of the professional video editors support sufficient real time collaboration. ”Sufficient” here meaning that you cannot see the other user editing and moving clips in real-time. There is however a software called Runway ML, that allows editing with teams. It shows which clip is selected by which user and it also shows all the edits done by different users in real time as demonstrated by Figure 4.2. This is however a very limited video editor in a browser, that does not support extensions.

Professional video editors such as Adobe Premiere and other tools in Adobe suite have plugins for collaboration like Frame.io for it. But that does not necessary integrate these features to the video editor, but rather allows easier communication with the users and rather seems like the collaboration was just an afterthought, not something tightly integrated to the core of the video editor.

5 Our Vision and Solution

5.1 Basic concepts

The most common actions video editors do is cutting, adding filters (either video or audio), compositing, adding titles and animations and also transitions from one scene to another. It does not matter whether the user is a beginner or a professional video editor, the building blocks of the video editor are essentially the same.

All these actions can be completed with 5 types of building blocks:

- **Sources:** These are the source items that hold the resources. They can represent videos, images or audio clips.
- **Filters:** These modify the stream (either audio or video) in some way. They have one input and one output.
- **Mixers:** These combine multiple streams into one in parallel. This means that the streams play at the same time.
- **Playlist:** These combine multiple streams into one in a serial manner. This means that the streams will be played one after another. There are also **transitions** in these that can be seen as a special case of a **mixer**; they play the streams at the same time, but they always have two inputs and a progress value (percentage of transition from one scene to another).
- **Drain:** The output that represents the final product.

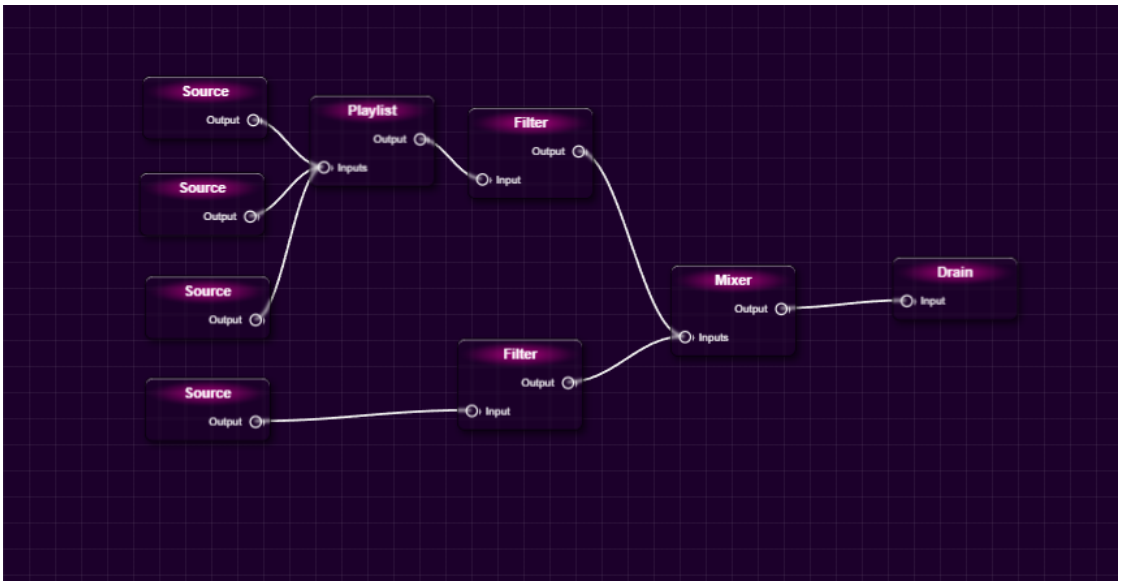

Figure 5.1: Typical video editing project drawn as a graph with the 5 basic building blocks

Figure 5.1 is a graph demonstrating common video editing scenario. The idea of these 5 building blocks in also present in Mlt-framework (discussed in Section 3.2.2). The names are slightly different in Mlt framework (source is called producer, drain is called consumer, etc.). [23]

The way Mlt framework implements these building blocks is by having all these building blocks implemented as modules defined in shared libraries. The code of these modules is executed in one process. This means that if there is one faulty module the whole editor will crash. This unfortunately tends to be the case with Mlt-based editors (such as Kdenlive, Openshot, Shotcut) at least with the experience of the author of this thesis.

5.2 Architecture and Modularity

Instead of having all the tools and utilities in a single process there should be multiple processes specialized in one task. What this practically means is usually

video editors have one program that does encode and decode all kinds of formats, cut and join video clips, apply filters to those videos (including color correction), analyze those videos in real time (histograms and tracking), edit audio with digital signal processing, render 3D graphics, render texts, animate all effects and texts. All this in one process. In our model these tasks are separated and run in different processes. Even if the program is able to do multiple things in one process. There might be different process instances of the same program doing different things (if that is what is necessary).

The main idea is to have a video editing program that works similarly to *make*, an old Unix command line program that creates results from different dependencies and eventually the end result as was explained in Section 2.2.1.

The motivation behind this is modularization is that individual people can develop parts to the program even without knowing how the internally program works or what kind of APIs does it have. It is also possible to support applications that were created before this kind of program and add support to it without modifying the program in any way. This would be ideal case of implementing an application that supports the bazaar-like ecosystem as was described in Section 2.1.3.

This kind of modular architecture adds several improvements but also drawbacks to the video editing software. The biggest improvement is stability and the biggest drawback is delay caused by the different complicated rendering mechanisms. These will be evaluated more accurately in Section 7.2.

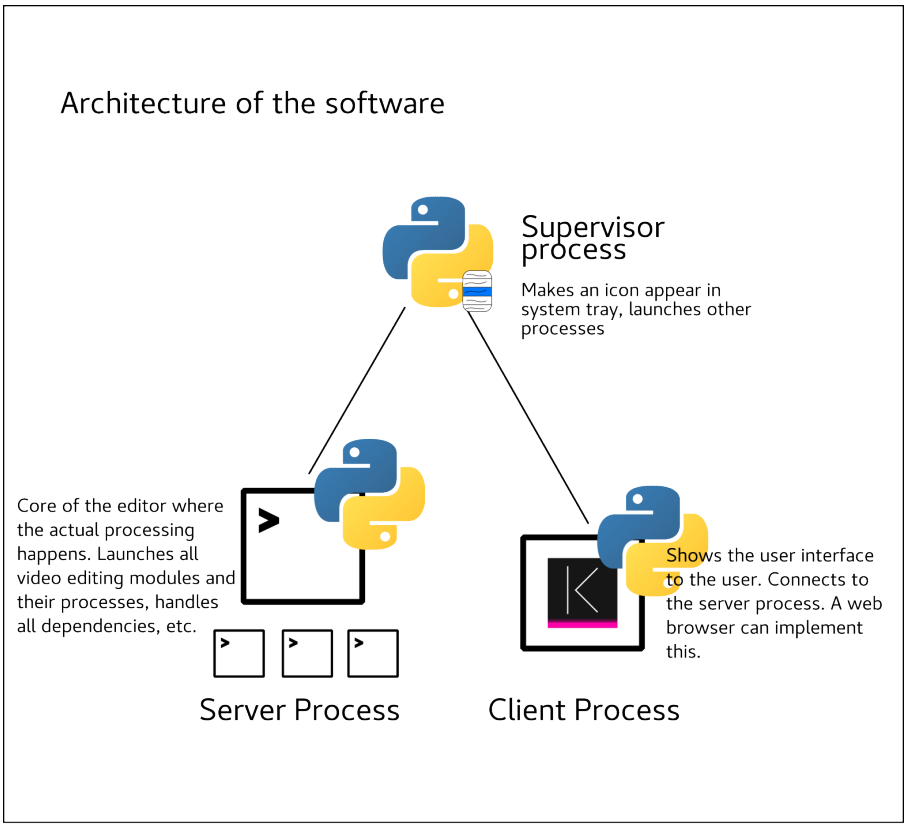

Figure 5.2: Processes and their sub-processes

The program itself can be made with two or three parts. Client and Server and the third one is the process that supervises these two. The relationships between these processes are explained in Figure 5.2. It is important to understand that the client shows the user interface and the server actually is where the processing happens. Supervisor process handles crashes and also starts the server and client processes when the user needs them.

5.3 Collaboration and Decentralization

This kind of modularization opens up possibilities for decentralization and collaboration. The decentralization in this case means that some programs or processes can be run on different computers. This allows the resources to be allocated more

evenly and also opens up new possibilities for collaboration.

What this all practically means is that one person is creating (and rendering) a title for the video and other person is adding subtitles to the video (which could be partially automated). This would mean that these two people and their computers are doing different tasks for the same video. All these changes can be synchronized to the either with central server or the servers would simply synchronize with each other.

5.4 Design Principles of the User Interface

This section covers the interface design of the most commonly used industrial video editors. Most users are used to the video editor they already are using so it is usually best to adopt the design patterns that other videos already have.

The basic idea is to compare them and see what kind of similarities they have and how they could be implemented in modular environment as stated by research question 3 (RQ3) defined in Section 1.2.

5.4.1 Typical UI Components

Like Figure 5.3 shows us, many video editors tend to share similar layouts and components. There are effect properties in one place, timeline in another and then there is the layout selection which allows different views to be visible. For example resources panel that contains all the clips in the project is not always visible, but in one view specifically created for it, allows user to drag and drop those clips to the timeline. At the top center there is typically a preview component that shows the preview render of the result. Typically there are also specific components for color correction and audio editing as well.

The main point here is that user interface of video editor is typically made of

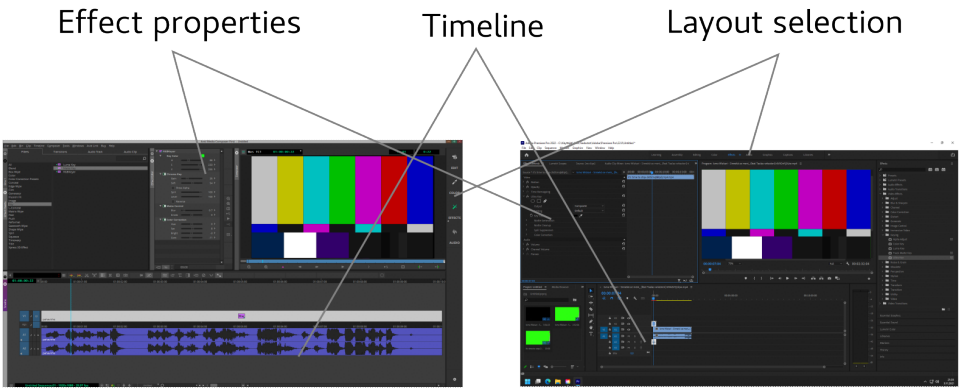

Figure 5.3: The most popular video editors have similar user interfaces. Left: Avid Media Composer, Right: Adobe Premiere Pro

smaller UI components, that form the whole functionality of the video editor. From existing video editors we can conclude that typical UI components in video editors are timeline, properties, preview, effect library and clips.

5.4.2 Customization of the Layout

These different components typically occupy small part of the window and they also interact with each other. However video editing can vary from cutting to color correction to audio editing, there needs to be customization of the user interface. What this means is that these all components can be dragged to different parts of the window and they would naturally fit them to be a new part of the user interface. Since all UI components are not needed at all times, some of them can be entirely removed from the view. This is why there needs to be different layouts and user needs to be able to customize these layouts for their own purposes. Users should be able to save these layouts and load these when certain layouts are needed. There also needs to be presets the layouts so certain functionalities have their defined layouts.

5.4.3 Layout Presets

These layout presets are typically for different phases of the project. There is one for managing resources, one for cutting, one for adding effects and transitions, one for color correction and one audio editing. There may be more layouts available, but these are the most typical ones. Some video editors may have very limited components specifically limited to these presets (or views or lenses as some video editors may call them) and others allow full customization and even creating new presets as mentioned in the previous chapter.

6 Reference Implementation

Up to this point, everything about the program has been theoretical implementation. But a reference implementation was also created with this thesis. This chapter is about that reference implementation. The reference implementation is also not fully complete and is indeed in very early pre-alpha-state.

Things covered in this chapter might have changed in later versions of the reference implementation. As previously mentioned in Section 5.2, the implementation has two main components and a launcher, which launches and supervises the two processes: the client and the server. The client is mostly a browser, which has the user interface written as a web page. The server is where all the processing happens.

6.1 Storage

The implementation has to store all the files (clips, source videos, metadata) somewhere. All the files must be stored under one folder, so the project can be easily exported into one file as a tape archive (.tar file, tape archives can also be compressed) and moved to another computer.

Each project folder holds the resources themselves, but also caches of rendered clips and so-called artifacts, which hold generated pre-rendered content, which is later used in the video editing project. These artifacts may be modified by the user. Latency may be issue for the end user, and these issues will be discussed in Section 7.2.1.

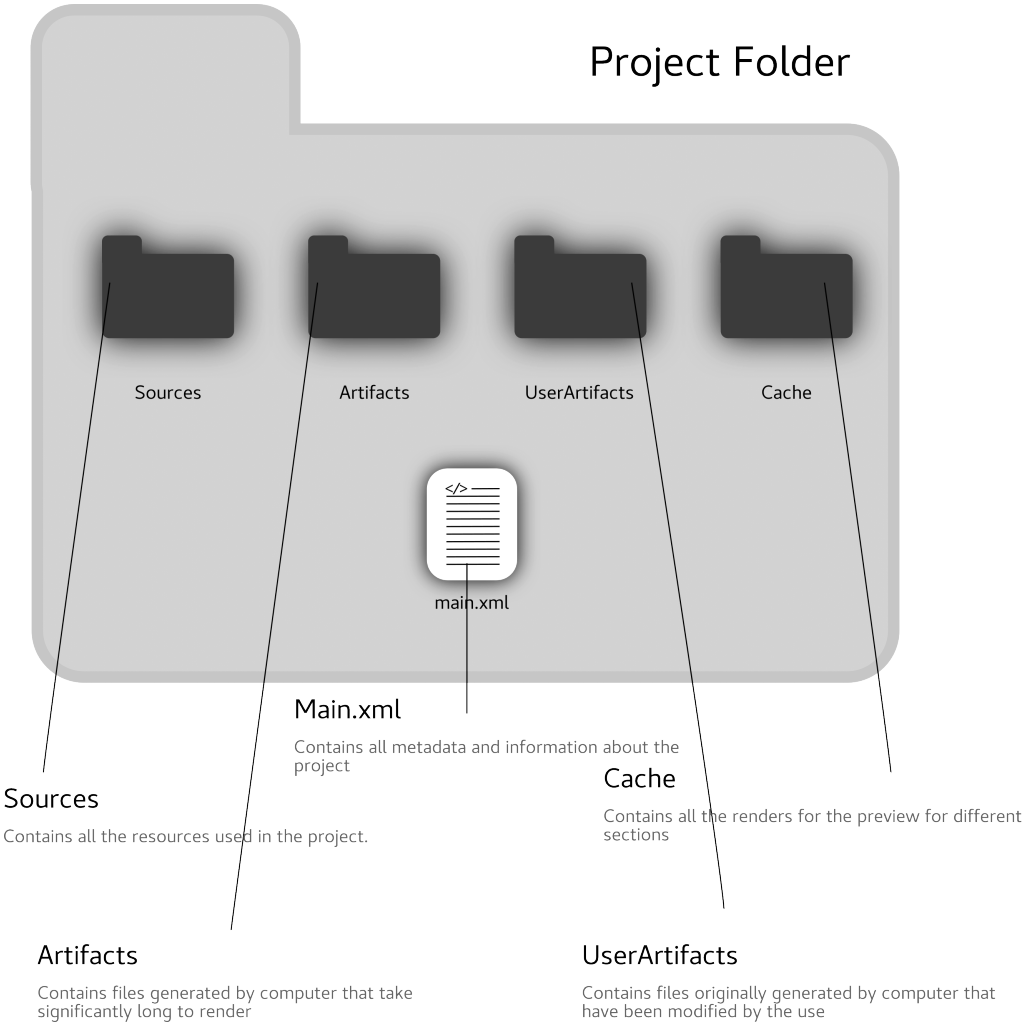

Figure 6.1: Folder structure of a project

The structure of the project folder is quite straightforward. There are directories for raw resources, cache and two artifact folders where the other one contains artifacts which were modified by the user. This separation is done because artifacts are not copied by default so when the folder is compressed, exported, and moved to another computer, these temporary cache- and artifact-files are cleared. User-modified artifacts are still copied.

There is also an XML-file in the project folder that contains all metadata, project state and possibly project history.

6.2 Standardization and Extendability

6.2.1 Introduction to plugins

Like mentioned earlier in Section 2.3, VSTs, OpenFX and other plugins are used to extend the functionality of video editor software. However, in our case of our video editor the most significant functionality comes from different programs which have their own extensions to control remote programs.

Most functionality can be provided by FFmpeg for example, but there are numerous reasons why the program cannot be just a frontend for FFmpeg:

- It has patent-related issues in some countries (as mentioned in the Section 3.2.3)
- While it is able to do most video editing tasks smoothly, there are some things it is not able to do
- It does not support VST- or OpenFX-extensions on its own
- Some encoders might have poorer quality compared to what is available

- To extend to latest technologies (like the latest innovations in artificial intelligence), there has to be API for extensions

6.2.2 Process Abstraction Layer

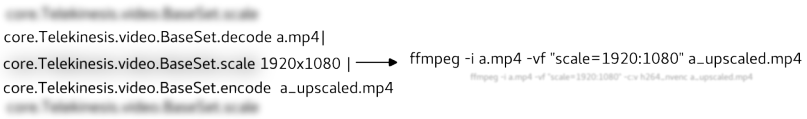

Figure 6.2: Demonstration of the abstraction layer — The commands work like aliases expect the arguments get translated too, this allows commands to be translated to FFmpeg-commands or to any other supported commands

The implementation aims to be as platform agnostic as possible. To achieve this, there is a process abstraction layer, which aims to create generic abstract processes or transactions (as defined in Section 2.1.4), which will be translated to actual native processes during the rendering phase. Practically this means that the commands are not directly set to FFmpeg for example, but to an abstract program, which then translates to FFmpeg-commands.

This also allows the program to optimize the process to the target machine. This means that there can be a separate program module, that allows GPU-acceleration. Because GPU-acceleration tends to be vendor-specific, different cards from different vendors need different parameters. This means, that the command to transcode the video file might be entirely different on different machines.

There can also be a unit test-like script, that tests that each implementation of an abstract command does what it is supposed to do.

6.2.3 Installation and Functionality of the Plugins

Installing plugins should be made as easy as possible for the end user and it would be good to have some kind of package manager for this kind of plugin based system. However first plugins would be most likely installed by decompressing a zip file into the plugin-folder.

The basic idea however is that python-files are stored in the plugin-folder and when the video editor starts it scans all the files in the folder and loads them into the main server process. The plugin declares the commands it implements and those commands will be added into a hash map (or a dictionary as they are called in python) and the program will then use the commands provided by the plugin when they are needed.

6.3 User Interface

The user interface attempts to be as similar and adaptable in the implementation compared to current video editors as possible. The layout of different existing video editors was discussed in Section 5.4.

The implementation adds node graph editor as core component to the mix. This is due to the modular architecture of the editor, which attempts to encourage the end user to create pipeline where the entire video gets processed.

Node graph editors as a concept are not as foreign as it may seem. There are several existing video editing programs or video editing related programs that user node graph editors. [31] [32] [33] Other Components, such as resource manager, timeline, properties, and preview modules should be familiar from existing video editors.

The implementation has context specific property editor, which allows the user to change the values of currently selected object. Some properties can also be animated

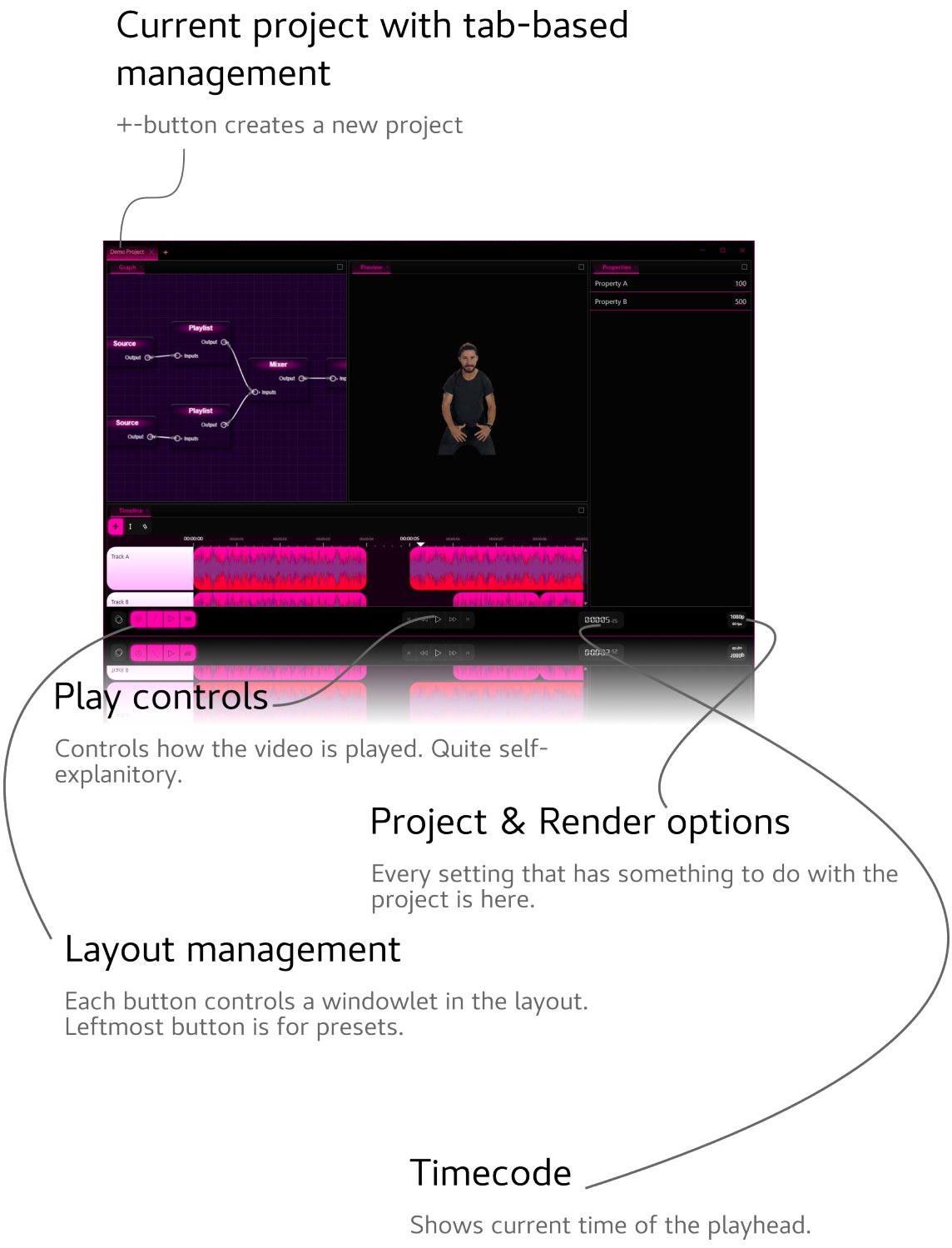

Figure 6.3: User Interface of the reference implementation. The preview-windowlet contains a still frame from #INTRODUCTIONS (2015) by LaBeouf, Rönkkö & Turner released under a Creative Commons Attribution Non-Commercial Share-Alike license

meaning that so called key-frames will be set here. These key-frames make animating possible and allow values to be interpolated to certain time period.

The user can freely move different modules in the user interface around and customize the layout of the program. The Figure 6.3 shows default user interface of the video editor. There are buttons at the bottom of the user interface. These cannot be moved as they control the state of the entire project. The layouts can be saved and loaded with the button at the bottom-left corner.

6.4 Security

Security is not necessarily the focus of this thesis. However, as this project connects to its server via internet, some discussion about the security is necessary.

The program uses HTTPS, where it is possible, to communicate with the remote or local server. The communication is done with WebSocket-technology. As these are standard web-browser technologies they are very much researched and they should be as secure as possible. The web browser must be kept up-to-date in order to fix existing vulnerabilities in older versions of the browser framework. The browser provided by the User Interface Library may not be up to date, this happens to be the case with cross platform user interface library Qt version 5. Qt5's WebEngine seems to use Chromium 87.0.4280.144, which is quite old and has many severe vulnerabilities. [34] [35] Qt6 seems to use newer browser engine, but it does not support Windows 7. Windows 7 still has 5% market share while writing this thesis (March 2023). [36].

The extensions that provide the functionality of the application allow to execute software that might potentially be malicious on the server machine. That is why it is under responsibility of the maintainer to only allow or install plugins that cannot be exploited.

Even if the access is limited to IP-addresses in local area network or other limited pool of IP-addresses, there is still risk of a man-in-the-middle-attack and TCP-sequence prediction attack, if https is not enabled.[37] Even then there is a risk of having incorrectly assigned certificates on the machine. [38]

7 Evaluation and Performance

7.1 Methods of Evaluation

This chapter will evaluate how well this modular video editing works from different aspects. Each chapter will answer to different research question and use different evaluation methods suited for them.

Section 7.2 will be about modularity and it answer the first research question: How to improve video editing software modularity? The chapter will list benefits, but also drawbacks of the modular architecture. The main idea is to discuss whether the usage of modular architecture genuinely makes video editing better and how much such architecture can be utilized.

Section 7.3 will be about stability and it answers to the second research question: How to improve video editing software stability? The chapter will demonstrate with code examples, how unstable code behaves under controlled environment. The main evaluation method here is to demonstrate, how much instability one unstable component can cause and how such damage can be mitigated.

Section 7.4 will be about usability for different user groups and it answers to the third research question: How to make a user interface, that allows the end user to use different modules in one program? The chapter will discuss how such product would fill the needs of different user groups. These user groups were defined in Chapter 4. This chapter is mostly about estimating needs for different groups as

currently such product does not exist. This is done by comparing how well this new product would be able to mimic the existing products, that these user groups typically use.

Section 7.5 will be about collaboration between users and it answers to the fourth and final research question: How to integrate collaboration with different people to the user interface? This discusses how the project can be expanded to support collaboration and how such feature should be integrated to be part of the video editor. After that there are discussion about other features that can be supported in the future.

7.2 Modularity

This chapter answers to **research question 1 (RQ1)**: How to improve video editing software modularity?

When it comes to modularity, the architecture allows each task or transaction to be isolated in one process. These kind of transactions were defined in Section 2.1.4. In practice this means, that each transaction gets its own process and none of the operations is executed in the main server process. This allows easily extendable and stable ecosystem for the video editor. This kind of architecture has many benefits, but also some drawbacks. Below are pros and cons of this kind of architecture listed:

PROS	**CONS**
Better stability	Latency of the real-time rendering
Easier extendability	Old wire format standards
Decentralized editing tools	Fragmentation
Collaboration	

Modularity provides better overall stability as all the operations are run in separate isolated processes. It also provides better extendability as plugins do not have to be loaded as libraries with defined APIs (even though that is possible as well, see Section 2.3).

Because the extendibility is much better, the core tools can be implemented outside of the main process meaning that much higher variety of codecs can be supported and much wider hardware acceleration can be achieved as well.

The last benefit of the modularity is that these processes can be run on different machine. This means that multiple users can connect to the server and edit the project at the same time. This would allow collaboration between different users.

There are some drawbacks as well. Because the previews have to be encoded before streaming, there will be significant latency caused by the encoding. The preview will be most likely encoded in 5 seconded segments. Some streaming services use 10 second segments, but in video editing in 5 second segments seem much better option. Latency between different codecs will be discussed further in Section 7.2.1.

The processed video data can be shared between processes by mainly by two different methods. Either by encoding the video segment into one file or by streaming the raw data to standard input and output or to a pipe, a named pipe or a socket. Shared memory may also be an option, but that does not seem to be as widely supported and has challenges with synchronization. There is a wire format for sharing the video between two processes with sockets, named pipes or standard input and output. This wire format was introduced in Section 2.2.2. The only drawback with this standard is that it is relatively old and does not seem to have standardized official support newer innovations such as HDR colors. However some unofficial extensions of Yuv4mpeg exist for HDR support. FFmpeg for example supports 9-bit, 10-bit, 12-bit and even 16-bit variations of YUV420 pixel format, which would allow support for HDR. However since this is not standardized extentsion of Yuv4mpeg

format, and it is not guaranteed to be supported by other programs. [39]

The third and last drawback is fragmentation, which is often an issue with plugin-based platforms. This means that the quality and user interface standards may vary between different plugins. In this implementation this issue can be avoided by defining core set of plugins, which have defined inputs and outputs. These core plugins also set the standards for other plugins.

7.2.1 Latency

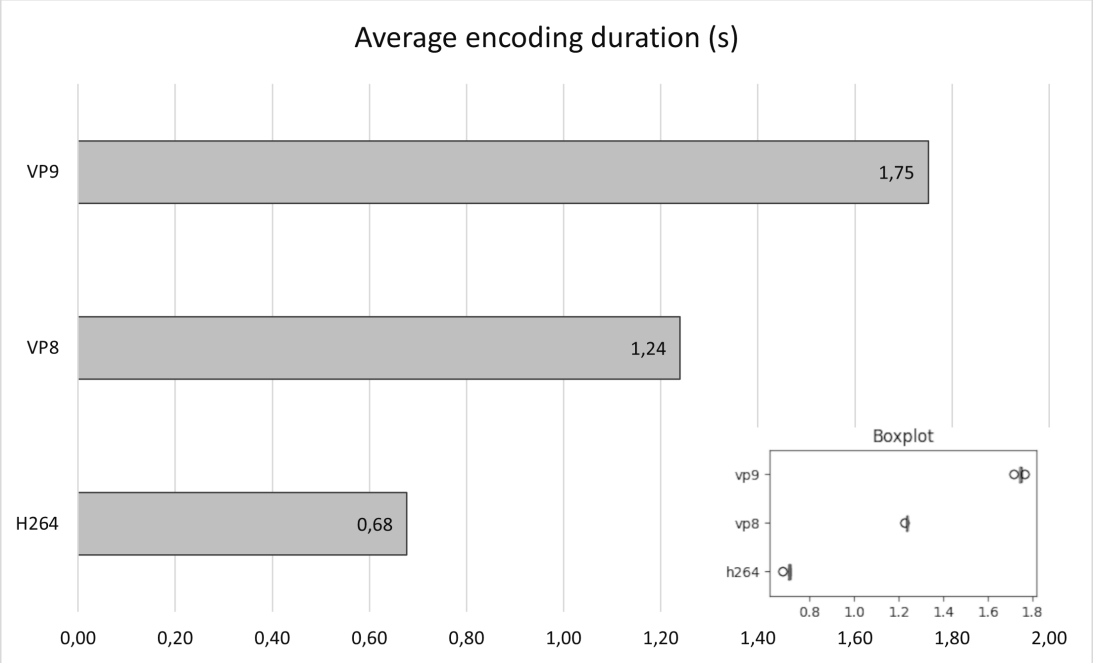

Figure 7.1: Comparison of encoding times of different browser compatible codecs (h264, vp8, vp9) in FFmpeg. The length of the video is 5 seconds and it is Full HD (1920x1080 pixels) and has 24 frames per second. The processor is Intel® core™ i5-6600K @ 3.5 GHz. The results are average of 5. Boxplot is also visible at the bottom right corner. The videos were encoded with fastest settings available without hardware acceleration. Lower value is better.

As demonstrated in the Figure 7.1, the encoding also takes a significant time before it can be streamed to the client. The encoding is the most significant latency issue in modular video editing. Software encoding is used in the comparison, hardware acceleration increases the performance significantly, but they are dependent on the servers hardware. Otherwise fastest presets were used without hardware acceleration. [1]

7.2.2 Summary

The answer to **research question 1 (RQ1)**: "How to improve video editing software modularity?" is by isolating different operations or transactions to different processes and defining an API or wire format, which allows them to communicate with each other. While modularization has some drawbacks, they can be mostly mitigated and the benefits give more extendable, stable and flexible environment for the end user.

7.3 Stability

This chapter answers the **research question 2 (RQ2)**: How to improve video editing software stability? There will be several code demonstrations in this chapter.

The modularity in the previous Section 7.2 means in practice that ideally all operations get their own process. Operation in this case means decoding, encoding, filters, etc. In practise, it is possible that multiple operations are done in one process and that in most cases is fine as long is it stable enough to not crash the main process

[1] Performance was measured with following commands:

time ffmpeg -i input.webm -deadline realtime -speed 10 -an -c:v libvpx-vp9 -threads 4 -b:v 10M -t 5 output.webm

time ffmpeg -i input.webm -speed 4 -an -c:v libvpx -threads 4 -b:v 10M -t 5 output.webm

time ffmpeg -i input.webm -an -c:v libx264 -threads 4 -b:v 10M -preset ultrafast -t 5 output.webm

or significantly degrade the performance of the video editor.

Because each transaction is isolated in their own processes, it does not matter if one process crashes. If a crash happens the process is either simply rebooted or replaced with another program that implements the same functionality. In worst case scenario the functionality cannot be processed and the functionality has to be replaced with another similar functionality or simply omitted.

In terms of stability, the architecture isolates as much functionality as it is reasonable to isolate functionality from the core and the core is written in a programming language that does not allow unsafe pointer operations. This means that in theory the server should never crash.

Following code demonstrates an unstable process handled by Python:

```
import subprocess
import time
import platform

process = subprocess.Popen(["test.exe"]) #Launch the native process

def idle_loop():
    while True:
        returnval = process.poll()
        if returnval:
            if ( (returnval == 0xC0000005 and platform.system() == "
    Windows") or
             (returnval == -11 and platform.system() != "Windows") ):
                print("Process crashed with a segmentation fault")
            else:
                print("Returned with value:"+str(returnval))
            break

        time.sleep(1)

idle_loop()
print("This runs only if THIS process does not crash")
```

Below is the code of the native process (test.exe) in C:

```
int main(int argc, char** argv) {
    char* ptr = 0;
    *ptr = 0; //This will crash the program;

    return 1;
}
```

Running the Python-code gives following output in the console:

```
Process crashed with a segmentation fault
This runs only if THIS process does not crash
```

What this practically means is that unstable code can be run in separated processes with Python's *subprocess*-module. Unstable Python-code can be run with *multiprocessing*-module, which similarly runs the code in separate process.

The code that runs in the main process is still able to crash the entire video editor. Therefore it has to be taken care that minimal amount of code gets executed in the main process. The code in the main process has to be carefully written and only use libraries that are stable.

Therefore the answer for the **research question 2 (RQ2)** is that, minimal amount of code is executed in main process, instead the heavy processing gets its own separate process, where data can be processed without crashing the whole main process where all unsaved progress is lost.

7.4 Usability for Different User Groups

This chapter answers to the **research question 3 (RQ3)**: "How to make a user interface, that allows the end user to use different modules in one program?" This research question is different to the previous two ones as it has human element in it. The previous two ones were entirely technical therefore the answer was objective. Here the answer however will be more subjective as it relies on different user preferences.

As was outlined in Chapter 6, the current implementation is still incomplete at its current state and therefore cannot be evaluated. However, this chapter tries to evaluate how such a complete video editor would work to specific user group.

7.4.1 Typical Home Video Editing

The typical video editor features (specified in Section 4.1) are present in the implementation. Transitions need their own program for proper implementation and average user might find the node system of the video editor quite confusing. A simplified version of the video editor might be appropriate.

The features specified in Section 4.1 can be easily implemented. Decoding and Encoding different video formats, cutting clips and placing them into the timeline and simple transitions can be seen as basic features. Adjusting volume and Adding texts and images can also be implemented, even though they need some more work. Text and image support can be done with scalable vector graphics (SVG), which is already standard in web browsers, so the composition window can be easily implemented. Adjusting volume should not be a challenge, however for different tracks the files need to be sent separately so their volume can be adjusted.

Biggest challenge is previewing the result in real time as there is the encoding latency discussed in Section 7.2.1. One frame can be also sent as a png-file for faster preview, but the result needs to be re-rendered always when a clip or its effects are modified.

7.4.2 Video Editing for Education

As mentioned in Section 4.2, video editing in educational fields is quite similar to typical home video editing. Basic operations such as cutting, transitions and audio mixing are used. On top of that there might be some more specific requests such as transcription of speech to text and handling multiple video sources such as monitor view and the camera in the lecture hall. As lectures tend to be streamed, real-time editing of these would be preferable.

While it is possible to do transcription of the speech to text. There are multiple speech-to-text services some paid, some cloud-based, some better than others. As

a speech to text framework, that is able to process audio locally, VOSK-framework seems quite promising as it already used by free open source video editor Kdenlive. [40] It has voice models for many languages, but support for Finnish language has not been implemented to this date. As a side project I decided to implement one. However I simply did not have enough data to have every single word in Finnish language to be recognized by the model.

Multiple video tracks in a video container is currently not in the scope of this project as the final output should have one video track and one audio track. It is possible to implement multi-track rendering in the future, but currently that is out of scope.

Real-time rendering is however completely out of scope. This kind of video editing is entirely designed for post-production and real-time streaming would limit its functionality too much and would most likely produce a product that is neither good for real-time editing nor post-production. This is also the reason why it is not recommended for streams (for example streaming lectures). While it is possible for some schools to use these programs utilizing this kind of processing method, mostly this kind of program is built for different audience.

7.4.3 Professional Video Editing

Profession video editors are the most demanding group as they have highest requirements for features. Practically only there are only two industry standard video editors. Some smaller video editors also exist, but those are rarely used by bigger teams.

While the list of features is long and demanding, they are not impossible to implement. Filters and OpenFX support is possible to do in a command-line program and there exists python implementation for that. Color correction can be seen as a special filter, that has many parameters to control the output. Analyzing videos

is special kind of filter as it requires slow pre-processing before the filter can be applied. This typically means calculating motion vectors to stabilize image. This can be implemented in modular way.

Audio editing with digital signal processing can be done in modular way even though it cannot be implemented in real time. Audio needs to be processed in segments or the entire track needs to processed every time when parameter is changed. VST plugins will operate in similar way.

3D graphics rendering can be done with free open source 3D-modelling program Blender. It is Python-based and also has APIs, which allow other programs to render files with it. Some programs such as Openshot already use this feature. [41]

Scalable vector graphics (SVG) have an extension called SMIL, which makes it possible to animate vectors graphics with keyframes. They can be converted into still svg-image, which will be converted to png-image and then into video file.

The hardest part is to have these all shown in real time as there is significant latency in encoding as previously mentioned. Not only that, but the heavier the effects, the more it takes time to render.

Biggest challenge here is not however the latency. It is the vendor lock-in. People tend to use video editors that have been very long in the development. People do not seem to willingly switch their video editors unless there is a very good reason to do so. Switching to a different complex application always comes with a huge learning curve. In larger projects, all the files, plugins and other infrastructure would have to be converted for the new application. Even the data seems to indicate that people tend to stick to video editors that have been developed for decades. [2] [3]

What however is possible is that the program get used in parallel with existing solutions. As this kind of decentralized system would allow better collaboration with other members in a post-production team. As a standalone program it is unlikely to be used by professionals due to latency, that pre-rendered clips cause.

7.4.4 Summary

The answer to **research question 3 (RQ3)**: "How to make a user interface, that allows the end user to use different modules in one program?" is that the program should be as similar to existing solutions as possible. It should also provide new features as without them there is no reason to switch. The reason why the modular video editor should be similar to existing solutions for the end user is simple. People do not want to utilize their existing knowledge and using something completely different can be frustrating. Different users have different requirements for the user interface as was pointed out with different user groups.

7.5 Collaboration

This chapter answers to **research question 4 (RQ4)**: "How to integrate collaboration with different people to the user interface?" Possibility to extend the collaboration features of video editing are discussed here.

Video editor, that has its front- and backend separated, can be easily turned into project that allows collaboration between different members of the team. This means that one central computer acts as a server, that renders the project and there are multiple people as a client editing the video to its final format.

In Section 4.4, there is Figure 4.2 where collaboration features of Runway ML-video editor get demonstrated. Similar collaboration features could be implemented to this project. The difference between our project and Runway ML is that Runway ML is run in a centralized server, but this project has both client and server in the same package, so no internet is required, you can host the server yourself. The collaboration features require local area network, but access to internet is not necessary. The updates to the project get shown in real time for both users. They can see what each user has done for the project.

Another possibility could also be that one person can disconnect from the project and lock part of it, so the user can later return to the project and merge their progress to the video editing project. The project could use version control (such as Git) to properly merge two versions of the same project.

The answer to the **research question 4 (RQ4)**: "How to integrate collaboration with different people to the user interface?" is that, real time updates and different users seeing playhead position in the timeline could be implemented as part of the user interface. Also integration with version control could improve collaboration features.

8 Conclusion

The video editor described in this thesis is surely modular and stable. It is however much harder to evaluate is it possible to construct a video editor that would satisfy the needs of different end users in different user groups. The video editor constructed in modular way would most likely not replace existing non-linear video editing solutions. This kind of video editor would however most likely have its use in collaboration, Linux and other free open source software users, and for those who are willing to use more experimental video editors.

Linux-users and users of other open source operating systems would most likely use this kind of video editor as there is not much competition in there and most video editors do not have support Linux or other free open source operating systems. Another audience would be other early adopter, who are dissatisfied with the current state of video editors. There could be also a market for users that are not professionals, but not really beginners either. Most likely people who make videos to YouTube, could potentially adopt more experimental video editors.

The implementation currently has a somewhat complete front-end (User Interface) and back-end (management and processing itself). Connecting those two is the next milestone. There are multiple opportunities and ways to extend the functionality of the video editing system. Proper remote session implementation, support for VST- and OpenFX-effects, Multi-seat editing, user-authentication, Git-based version control, and other collaboration features could be the next steps to extend the

functionality of the video editor.

Collaboration is most likely the direction this kind of video editor would be taken towards. Two or more people could edit the video at the same time, which would allow video editing that none of the mainstream video editors currently are able to provide.

References

[1] J. Roizen, "Quadruplex video-tape editing — an introduction," *Journal of the SMPTE*, vol. 79, no. 3, pp. 177–182, 1970. DOI: 10.5594/J13603.

[2] *Industrial video editor market share*, Survey done at www.learningdslrvideo.com. [Online]. Available: https://www.youtube.com/watch?v=5zgdYW6rh3o&t=123s.

[3] *The big 3 nles and their place in today's film industry*, [fetched 2023/04/05]. [Online]. Available: https://www.premiumbeat.com/blog/big-3-nles-place-todays-film-industry/.

[4] *Blackmagic & davinci, and what it means*, [Archived interview; fetched 2023/04/05]. [Online]. Available: https://web.archive.org/web/20190426195124/https://library.creativecow.net/article.php?author_folder=petty_grant&article_folder=grant_petty-blackmagic-davinci&page=1.

[5] R. Evans, *Practical DV Filmmaking*. Taylor & Francis, 2013, ISBN: 9781136067976. [Online]. Available: https://books.google.fi/books?id=r%5C_FvKJV7oygC.

[6] R. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design* (Martin, Robert C). Prentice Hall, 2018, ISBN: 9780134494166. [Online]. Available: https://books.google.fi/books?id=8ngAkAEACAAJ.

[7] M. McIlroy, E. Pinson, and B. Tague, "Unix time-sharing system," *The Bell system technical journal*, vol. 57, no. 6, pp. 1899–1904, 1978.

[8] E. Raymond and B. Young, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* (O'Reilly Series). O'Reilly, 1999, ISBN: 9781565927247. [Online]. Available: https://books.google.fi/books?id=o40ZAQAAIAAJ.

[9] G. Weikum and G. Vossen, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery* (The Morgan Kaufmann Series in Data Management Systems). Elsevier Science, 2001, ISBN: 9780080519562. [Online]. Available: https://books.google.fi/books?id=2sFRpAmNnoMC.

[10] *Make man-page*, [Unix Manual page; fetched 2022/04/20]. [Online]. Available: https://linux.die.net/man/1/make.

[11] *Yuv4mpeg man page*, [Unix Manual page; fetched 2023/04/03]. [Online]. Available: https://www.systutorials.com/docs/linux/man/5-yuv4mpeg/.

[12] *Open effects association web page*, [fetched 2023/04/05]. [Online]. Available: https://openeffects.org/.

[13] *Vst 3 api reference*, [fetched 2023/04/05]. [Online]. Available: https://steinbergmedia.github.io/vst3_doc/vstsdk/index.html.

[14] *Renderman git repository*, [fetched 2023/04/14]. [Online]. Available: https://github.com/fedden/RenderMan.

[15] *Dawdreamer git repository*, [fetched 2023/04/14]. [Online]. Available: https://github.com/DBraun/DawDreamer.

[16] *Cython-vst git repository*, [fetched 2023/04/14]. [Online]. Available: https://github.com/hq9000/cython-vst-loader.

[17] *Pyvst git repository*, [fetched 2023/04/14]. [Online]. Available: https://github.com/simlmx/pyvst.

[18] *Nvidia studio driver*, [fetched 2023/05/23]. [Online]. Available: https://www.nvidia.com/download/driverResults.aspx/189618/en-us/.

[19] *Msdn — access violation*, [fetched 2023/05/03]. [Online]. Available: https://learn.microsoft.com/en-us/shows/inside/c0000005.

[20] *Sigsegv: Linux segmentation fault*, [fetched 2023/05/03]. [Online]. Available: https://komodor.com/learn/sigsegv-segmentation-faults-signal-11-exit-code-139/.

[21] *Starvation and deadlock*, [fetched 2023/05/03]. [Online]. Available: https://www.tutorialspoint.com/starvation-and-deadlock.

[22] *Ffmpeg webpage*, [FFmpeg webpage; fetched 2022/05/29]. [Online]. Available: https://ffmpeg.org/about.html.

[23] *Mlt framework source repository*, Github repository; fetched 2022/07/15. [Online]. Available: https://github.com/mltframework/mlt.

[24] *Mlt multimedia framework webpage*, Mlt Webpage; fetched 2023/05/10. [Online]. Available: https://www.mltframework.org/.

[25] *Gstreamer webpage*, [Gstreamer webpage; fetched 2022/09/01]. [Online]. Available: https://gstreamer.freedesktop.org/.

[26] *Ffmpeg webpage — license and legal considerations*, [FFmpeg webpage; fetched 2023/05/01]. [Online]. Available: https://www.ffmpeg.org/legal.html.

[27] *Tuttle ofx webpage*, [TuttleOFX webpage; fetched 2022/09/01]. [Online]. Available: https://sites.google.com/site/tuttleofx/.

[28] *Vapoursynth webpage*, [Vapoursynth webpage; fetched 2022/09/01]. [Online]. Available: https://www.vapoursynth.com/category/vapoursynth/.

[29] *Glaxnimate webpage*, [Glaxnimate webpage; fetched 2022/09/01]. [Online]. Available: https://glaxnimate.mattbas.org/.

[30] *Operating system market share*, Archived, accessible via Web Archives. [Online]. Available: https://web.archive.org/web/20120909203552/http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=11&qpcustomb=0.

[31] *Davinci resolve — fusion*, [DaVinci Resolve documentation; fetched 2023/02/02]. [Online]. Available: https://www.blackmagicdesign.com/products/davinciresolve/fusion.

[32] *Blender — shader editor*, [Blender documentation; fetched 2023/02/02]. [Online]. Available: https://docs.blender.org/manual/en/latest/editors/shader_editor.html.

[33] *Houndini layouts*, [Houndini documentation; fetched 2023/02/02]. [Online]. Available: https://www.sidefx.com/docs/houdini/network/layout.html.

[34] *Qtwebengine/chromiumversions*, [Qt documentation; fetched 2023/03/15]. [Online]. Available: https://wiki.qt.io/QtWebEngine/ChromiumVersions.

[35] *Google chrome cve security vulnerabilities*, [Chromium CVE details; fetched 2023/03/15]. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-15031/year-2021/opec-1/Google-Chrome.html.

[36] *Windows desktop version market share*, [Windows Desktop Version Market Share; fetched 2023/03/15]. [Online]. Available: https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide.

[37] S. M. Bellovin, "Security problems in the tcp/ip protocol suite," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 2, pp. 32–48, Apr. 1989, ISSN: 0146-4833. DOI: 10.1145/378444.378449. [Online]. Available: https://doi.org/10.1145/378444.378449.

[38] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 78–81, 2009. DOI: 10.1109/MSP.2009.12.

[39] *Colorspace in y4m*, [fetched 2023/05/16]. [Online]. Available: https://docs.rs/y4m/latest/y4m/enum.Colorspace.html.

[40] *Kdenlive — speech to text*, [Kdenlive documentation; fetched 2023/03/14]. [Online]. Available: https://docs.kdenlive.org/en/effects_and_compositions/speech_to_text.html.

[41] *Openshot github page — blender rendering*, [fetched 2023/05/16]. [Online]. Available: https://github.com/OpenShot/openshot-qt/tree/develop/src/blender.