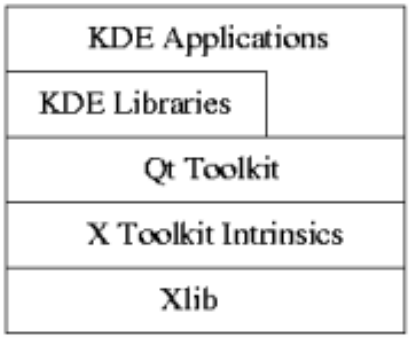

Figure 1: The KDE architecture

0.1 Choice of libraries and standards

0.1.1 GUI

As our GUI library we choose the Qt[**?**] from the start. This choice was made mainly because it is a class library that are written in the same language as we use (C++) and because KDE, which is the desktop Skolelinux use, is based on it.

As shown in figure 1 KDE applications are written using Qt and the KDE libraries. Although Skolelinux uses KDE we didn't want to tie our application to closely to this graphical environment, and therefore we haven't used the KDE Libraries although these had several interesting classes such as the KDirWatch which would have aided us. The KDE libraries would have been an dependency which would have made it harder to spread our application to the wider community, which in the end is the goal of most open source projects.

Although we selected Qt early we decided to factor out all the Qt GUI code so that the software isn't completely dependant on it. The way this is done was explained in section **??** "Architecture" on page **??** and because of it we were able to write a NonGUI frontend where the user can use the application from a shell. This frontend can be used for scripting automating tasks or for a script testing framework at a later stage.

0.1.2 XML parser

When saving the project to persistent storage we had to choose a format for serializing the data structure. A natural choice for this is XML and as this seemed to be more than sufficient for our needs, along with being an increasingly used standard we decided to use this format.

The next decision we had to make was on the type of parser to use for reading and writing the XML file. We identified two alternatives: DOM and SAX. The advantage with a DOM parser is in flexibility and ease of use, while a SAX parser is faster and more efficient. We choose a DOM parser because the speed and low memory consump-

tion of a SAX parser didn't have much impact on the relative small amounts of data we are saving to XML. The ease of use in the DOM parsers was more important to us.

When the choice to go for a DOM parser was made a natural choice of library was libXML2 which is widely used, well documented and available on most platforms. The latter means it will be easier if someone decides to port the application to another operating system later.

To make our XML files as standard as possible we chose to use the SMIL standard[**?**] which is a W3C standard.

0.1.3 Threads

During this project we have focused on minimizing the number of threads in the software as many threads usually leads to quirks and unpredictable bugs in the software when the utility threads work on the same data as the main thread. Our software still has to be able to do several things at the same time on many occasions or else the GUI would lock/lag and lead to annoyed users. This is done in three different ways depending on the situation: Timers(playback and animation running), Registering code in the main loop(file monitoring) and through code which allows the event loop to perform its processing with regular intervals(frame adding).

Some places however threads either required or more practical than the other alternatives. For thread support in Stopmotion we have used two different libraries depending on where the code which needs threading is located. In the presentation and application layers where we can use Qt we have used the QThread classes to create and run threads. These classes are very handy as well as object oriented. In the other layers we couldn't use Qt because this would lead to a Qt dependency and therefore tie the whole application to this library. A user who wants to use an another frontend such as the non-GUI frontend or a new GTK fronted he or she has written, would then still need to compile and link the application with Qt. This is not acceptable.

Therefore, for the other layers needing support for threads, we have chosen to use Pthread, a POSIX thread library specified by the IEEE POSIX 1003.1c standard (1995). Pthread has good support for creating and manipulating threads and is not a critical dependency as it's POSIX-compliant.

0.1.4 Datastructures

In our software we also needed some library for handling storage in the main memory, with lists, vectors, etc. We had several choices here and chose to use Qt's lists and vectors in the presentation and application layers where Qt were already an dependency. These are easy to use and very handy.

In the lower layers we couldn't use Qt so we chose to go with STL here as these libraries are very flexible as well as being a part of the Standard C++ libraries.

0.1.5 Graphical manipulation

Some of the most important features in our application are the viewing modes for helping the user stage the next snapshot.

This was at first meant to be handled by GStreamer plugins. A plugin called ImageMixer for mixing images on top of video already existed but wasn't included in the GStreamer code yet so we planned on shipping this plugin with our program. There was no plugins to cover the image differentiation and playback functionalities so we would have had to write these ourself, and we have spent some time on investigating the GStreamer plugin system.

Later when we moved away from GStreamer to a system where we import snapshots from other programs (see section **??**), we needed some library to efficiently display these images and add effects to them. We could have used Qt's for displaying these images the way we did in the framebar, but we wanted/needed hardware acceleration for displaying and manipulating the pictures as this is a potential bottleneck. We therefore choose to go with the SDL(Simple Directmedia Layer) which allows us to view and manipulate images and video. As quoted from the libSDL webpage[**?**]:

> Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer.

SDL gives us a lot of speed and flexibility in displaying and manipulating image surfaces and also allows for the program to be ported to other platforms than GNU/Linux if this should become desirable at a later stage.