Chapter 1

Infrastructure, tools and packaging

In this chapter we have collected some sections that are mostly technical, but which we didn't think fitted in the implementation chapter. We wanted to keep the Implementation chapter cohesive and only write about issues related to the implementation of the actual application.

This chapter explains how we have managed documents and how the infrastructure we have created for distributing these documents work. We will also describe how the Stopmotion packages are built and the various tools we have used during our project.

1.1 Tools

Stopmotion has become an application with a relatively large amount of classes and allot of code lines. To navigate through these classes effectively and recompile repeatedly without having to stare at the roof for minutes at a time, it's necessary to use different tools speeding this up. We have used the following tools helping us with that:

Make

Make is an intelligent tool which controls the generation of executables of a program from its source files. It builds the executable based on information described in a makefile following a strict syntax. The makefile should therefore be well written and has to be error free for make to interpret it.

The advantage of using make is that it automatically figures out which files it need to recompile, and which ones it doesn't need to recompile. It determines if a source file is dependant on an another source file and recompiles them if one of them has changed. Make uses the timestamp for the file to figure out when it was last modified and uses this to find out if it need to be recompiled. That means it doesn't actually check the contents of the file for differences.

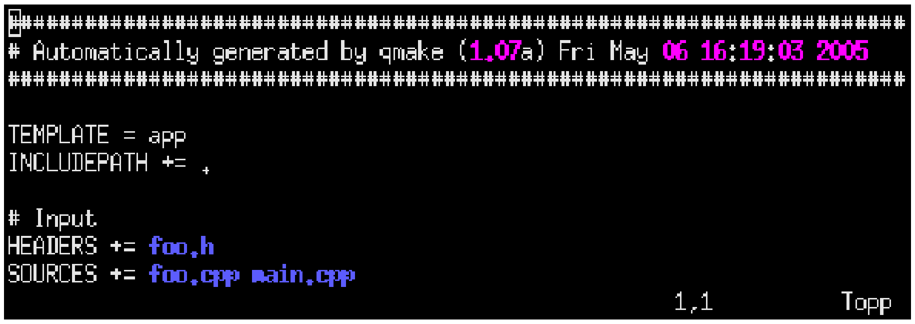
Ccache

As described above, make only checks the timestamp for a file to find out if it needs to be rebuilt. That means a file will be recompiled if you opens and saves it without making any changes. If the file also has dependencies to other files, make will decide to recompile them too. It would be really nice if we only could recompiled the files which actually has its contents changed. Ccache can help us with that.

Ccache uses wrappers for both gcc and g++ and has a cache containing the contents of previously compiled files. The wrapper compiler acts exactly like its "original" compiler, so the source code is interpreted and compiled the same way. The only difference is that Ccache compares the content of a file with the content of the same file found in cache before it compiles it. Recompiling only happens if the file is different from the file found in cache. This often results in a five to ten times speedup on recompile. In our case we have measured it to be 11 times faster on its maximum.

QMake

Writing makefiles by hand is time consuming, boring and very error prone. QMake is a tool created by Trolltech which auto-generates the makefiles and takes care of getting the right dependencies for the running platform and compiler. Since it doesn't have super cow powers it needs to be fed with some input from a project file (.pro). This file can be auto-generated by qmake itself by running *qmake -project* in the top-level directory containing the source code. QMake will then scan through all of the files in the running directory and sub-directories and generate the file. It's also possible to edit the file by hand if you need to add additional information. The project file format is simple and human readable and a very basic example file can be found in section 1.1 on



```
######################################################################
# Automatically generated by qmake (1.07a) Fri May 06 16:19:03 2005
######################################################################

TEMPLATE = app
INCLUDEPATH += .

# Input
HEADERS += foo.h
SOURCES += foo.cpp main.cpp
                                                  1,1          Topp
```

Figure 1.1: A very basic qmake project file

page 3. When the project file satisfies all of your needs, the makefile can be generated with executing *qmake <filename>.pro.*

Qmake will auto-generate makefiles and do a lot of job for us, but in our case we still have to do some work on the .pro file. It's not sufficient to let qmake auto-generate the .pro file because of the dependencies tied to Stopmotion. It's necessary to add some include paths and linking parameters which are specific for these libraries. By the way, this isn't good supported by the format if you use *pkg-config* or similar tools. In our case, when using *sdl-config*, we had to use *sed* and *grep* like this to get it work: *INCLUDEPATH += $$system(sdl-config –cflags | sed -e 's/-I//g')*. We also had to list some other files to get them included with the tarball generated with make dist.

In addition the .pro file is also used when generating the translation files, but this is explained in section 1.2.

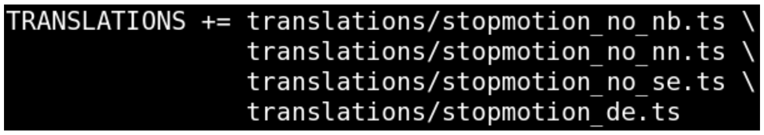
KDevelop

KDevelop has a separate QMake template which is excellent to use when programming Qt applications. KDevelop takes care of editing the .pro file with a tool called QMake Manager. This tool will automatically add information needed in the .pro file when adding new header files or source files to the project, and it will not override the manual settings such as the include path mentioned above.

In addition to this there's a lot of short cut keys such as F8 for compiling, Shift-F9 for executing the program, Ctrl-Shift-s for getting a Doxygen template for documentation etc. It's easy and quick to move between header files and theirs source files, and a lot of external programs such as debugging tools are available from the menus, if they're installed on the system.

1.2 Internationalization

Internationalization is important for any software, perhaps even more so for an open source project. We aim at having users in a variety of countries in all parts of the world and internationalization and translation of the application is therefore very important.



```
TRANSLATIONS += translations/stopmotion_no_nb.ts \
                translations/stopmotion_no_nn.ts \
                translations/stopmotion_no_se.ts \
                translations/stopmotion_de.ts
```

Figure 1.2: Translation fields in a .pro file.

We can't expect everyone to read English, especially when our software is primarly aimed at students in primary and secondary schools.

Qt supports internationalization and provides some functions and tools which eases this work. All text in a Qt application which should be translated to various languages have to use QStrings and has to be processed with the `tr()` function.

Here is an example of how some user visible text (a tooltip using HTML formating), which needs to be translated are entered using the tr function:

```
infoText =
        tr("<h4>Remove Selection (Delete)</h4> "
        "<p>Click this button to <em>remove</em> "
        "the selected frames from the animation.</p>");
```

As mentioned above Qt has several tools to ease the translation process.

Ts files

The *lupdate* program takes the stopmotion.pro file and the source-code as input. It then scans through the source code looking for instances of the tr() function. All of these instances are added to XML based files with the extension .ts (eg. stopmotion_no_nb.ts, stopmotion_gr.ts, ...). This file contains the English text, from the tr() functions as well as the translation in the language for that file. The translators can then use a tool called *Qt linguist* to translate the .ts files.

Before using *lupdate* on the .pro file however, some fields have to be added to this describing which translation files to make as shown in figure .

When some or all of the text have been translated one can create a .qm file using lupdate program, again with the stopmotion.pro file as input. The .qm files can then be loaded in the source code using the QTranslator class and used to translate strings.

Po files

The system with the .ts files is all one should need for translating the application, but as we are working on an open source project we can't tell people to do things our way. If we want people to use their spare time for free to help us translate our application it has to happen on their terms.

The translators in the Skolelinux didn't like the .ts files as they were accustomed to a file type called .po. To recruit translators and to be able to upload our files to the Skolelinux i18n repository we needed .po files of the strings to be translated.

Qt used to use this format in older versions so we were able to find some tools called *findtr*, *msg2qm* and *mergetr*. findtr will scan through the inputed source files and produce a .po file. msg2qm produces .qm files from the .po files and mergetr will merge to .po files.

One problem we had with the findtr command was that it was less intelligent than the equivalent lupdate command. This meant that we had to change some parts of the source code in order to allow it to find the strings to translate. We also made some small scripts to ease the use of the findtr command to create po files for many languages. These scripts can be found in section **??** on page **??**.

To translate the .po files there are several tools out there. They are in ASCII format so one could use any text editor, but if one want a graphical environment for the translation one can for example use the KBabel program, which is very popular in the KDE world.

Implementation

Blanchette and Summerfield points out in chapter 15 of their excellent book on Qt programming[**?**] that for most application it is enough to set the language at startup. We wanted the user to be able to switch language at runtime so we had to go through some extra effort to do this. When the user selects a new language we have to retranslate all the strings in the program so that Qt updates them.

The user can change language by going to the Settings->Languages menu. Stopmotion dynamically detects and loads all the translation .qm files which have been translated or partly translated and adds them to this menu. With Qt program it is customary to place the translation files in the catalogue /usr/share/<programname>/-translations and the translation files for Stopmotion is thus placed in the catalogue /usr/share/stopmotion/translations when one installs it from the Debian package.

Our module for creating language menus and loading translators (LanguageHandler) was another piece of code which proved to be very general and as such our sister Skolelinux project, the gnup project, was able to use the class directly in their AdminWorm application[**?**].

Stopmotion has currently only been translated to English and Norwegian no_NB, but now that we have .po files and now that the strings have stabilized we think we should get some speed in this process.

1.3 Packaging the program for distribution

Gunzipped tarball

This is the most simple format for a package. It only contains source code and files needed to build the program. Dependencies are often checked with a configure script which gives an error if unmet dependencies are found. The user is responsible for getting the libraries needed to build or run the program. In other words, the user needs to do everything by hand for getting it up and running. However, many peoples still prefer to do it this way, especially Slackware users.

In our case, this kind of package isn't difficult to build since Qmake already has added a section in the Makefile taking care of this. Running *make dist* will give us the gunzipped tarball, named *stopmotion.tar.gz*, which is ready to be shipped out to the people. To make it easier for the people to see which version the new package reflects, we repacks it with the name *stopmotion-x.y.z.tar.gz*, where *x*, *y* and *z* are the current version (e.g. 0.3pre.2). All this is done automatically with a script which will be further described below.

Debian package

One of the most fabulous things with Debian is the package management system. In contradiction to the simple tarball described above, everything is done automatically when installing a Debian package. Each and one of the dependencies to a package are downloaded and installed when installing the package itself. It can do a lot of more things too. However, to get such a complex package management system working well, packages have to follow strict rules; they have to be fully in line with the Debian policies.

When the 2.0 version of Stopmotion was finished in late of march, we decided to try getting this excellent software into Debian. The first step was to report a "Request For Package" (RFP) into the Debian Bug Tracking System (BTS). RFP means that someone has found and an interesting piece of software and would like someone else to maintain it for Debian. The bug report which was sent to the BTS can be read in appendix G on page **??**. We had to do this because no others than Debian developers can upload packages to the repository.

After two weeks nobody had responded to our RFP, but we could still see the light in the tunnel. We knew that it would be at least three official Debian developers at the upcoming gathering in Greece. The chance for catching one of them was therefore good. We succeeded with that. Andreas Schuldei, a German software developer, was willing to be a *sponsor* for the package. A sponsor is a Debian developer who acts as a mentor; they checks the package to see if it's packed correctly and uploads it to the Debian archive when they're satisfied with it. One of the group members therefore needed to be a maintainer for the package.

Building a Debian package is not a very complicated task if you have some experience with GNU/Linux and Unix programming. But, if you're a real novice, it's hard. An experienced Debian developer has stated the following [**?**]:

> One thing is certain, though: to properly create and maintain Debian packages you need man hours. Make no mistake, for our system to work the maintainers need to be both technically competent and diligent.

Since one of the project members already had builded few Stopmotion packages and became familiar with the most important Debian policies, it wasn't that hard getting the package correct. So, a few days after we had arrived Norway and was back in business, we had our first package into the Debian upload queue. After a while the package move on to the Debian repository[1] and now everybody who uses Debian can easily install our

[1] http://packages.debian.org/stopmotion

program by typing the following command in their consoles*:*

```
apt-get install stopmotion.
```

In other words: Our dream came true.

After the package had been built twice by hand, a bash script (see appendix **??** on page **??**) was created to automate the process as much as possible. The script creates a gunzipped tarball which contains all of the necessary files needed to build the application. The .pro file is also changed so compiler flags are switched from "debug mode" to "release mode", which means a more optimized executable. The tarball is then used to build the Debian package. When the building is finished and the output from lintian – a Debian package checker – is ok, everything is uploaded to one of the group members private Debian repository. Andreas is then given a hint about that a new package is available. He then gets the necessary files from the private repository and checks them. Everything is then signed with his GPG key and uploaded to the official Debian repository if the package is ok.

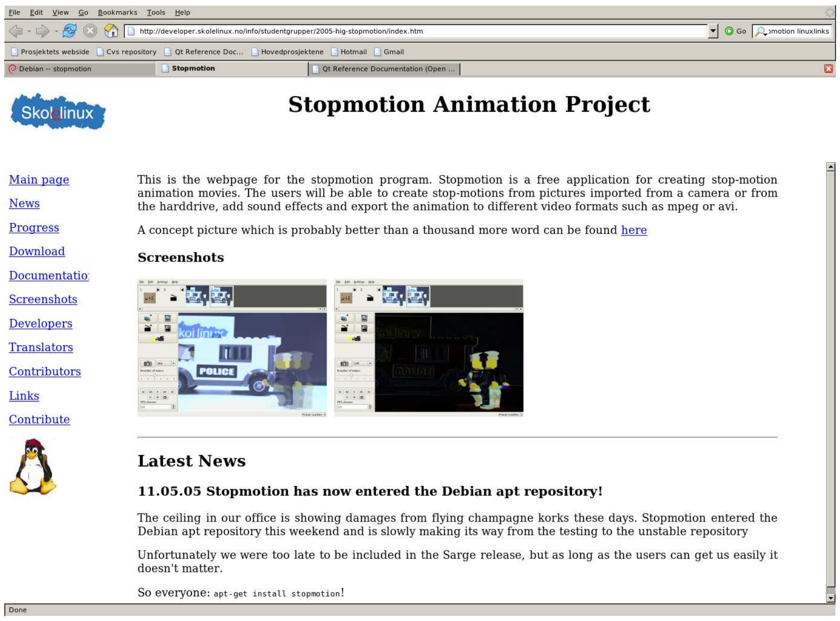
1.4 The Stopmotion webpage

Being an open source project it was important to have a proper webpage[**?**]. The webpage is usually the first thing a potential user or contributor sees and if it doesn't give them the right information they will move on. Therefore we have spent allot of time and effort on creating a webpage which is informative and easy to navigate.

Our webpage literally contains all the information there is about the project. It isn't very fancy with flash menus, etc, but it is visually tasteful and it serves its purpose as a portal for the our project.

The first thing a potential user or contributor wants to see when they enter a open source project website is screenshots, a link to a download section and some information about what they can use the application for. All of this is provided on our front page as shown in figure 1.3. In addition we also have the following sections worth noting:

- The *News* page where we make announcements.
- The *Download* page one can download the packages for Stopmotion, both as Debian packages (i386 and powerpc), as tar.gz files and as rpm packages. There is also information there on how one can download the entire CVS branch, a link to the ViewCVS as well as some example animations and this report.
- The *Screenshots* page with more screenshots presenting the various aspects of stopmotion.
- The *Documentation* page where all the documentation for this project is, including the user manual, the design and requirement documentation, the API documentation and the general project documentation.
- The *Translators* page where we have information for potential translators including translation files in both .ts and .po format (see section 1.2 on page 3).



http://developer.skolelinux.no/info/studentgrupper/2005-hig-stopmotion/index.htm

Stopmotion Animation Project

This is the webpage for the stopmotion program. Stopmotion is a free application for creating stop-motion animation movies. The users will be able to create stop-motions from pictures imported from a camera or from the harddrive, add sound effects and export the animation to different video formats such as mpeg or avi.

A concept picture which is probably better than a thousand more word can be found here

Screenshots

Latest News

11.05.05 Stopmotion has now entered the Debian apt repository!

The ceiling in our office is showing damages from flying champagne korks these days. Stopmotion entered the Debian apt repository this weekend and is slowly making its way from the testing to the unstable repository

Unfortunately we were too late to be included in the Sarge release, but as long as the users can get us easily it doesn't matter.

So everyone: `apt-get install stopmotion`!

Figure 1.3: Stopmotion webpage

Our webpage has been quite popular and some of our users and testers originally found us through it. It is ranked among the top sites on google for the search string "stop-motion", sometimes being number one, and we have had more than 1100 unique hits when writing this.

In addition to our main webpage we also have a section on freshmeat[2] where we have 650 hits and 7 subscribers as well as sections on other webpages which we never even registered at such as linuxlinks[3] and directory.fsf.org[4].

1.4.1 Maintenance

The webpage contains a lot of text which isn't changed over time, but we also have sections with highly dynamic content. Especially the sections containing downloadable packages, API doc and general documentation such as this project report and iteration plans can change on a daily basis.

From the beginning of we wanted the newest versions of all our docs to be easily available at all times for people interested in the project. A script responsible for creating PDF documents from all of our LyX files was therefore early created. This script was set to run in a cron job every second hour. It updates a pool with the newest files from the CVS server – where everything is continuously checked in by us – and creates PDF documents from all the LyX files in this pool. All of the PDF files are then

[2] www.freshmeat.net
[3] http://www.linuxlinks.com/portal/cgi-bin/search.cgi?query=stopmotion
[4] http://directory.fsf.org/graphics/anim/Stopmotion.html

automatically moved over to the webpage, where they become www readable when the export process is finished. The PDF files on the web site will have the same directory structure as the LYX "source" files have on the CVS server and since this directory structure is well organized it's easy to navigate. It also makes it usable for us when we have to read the docs ourself. The group members has actually used the webpage consequently for reading the docs as it is faster to navigate than our own directories and already have the files in PDF format.

The Stopmotion manuals are also updated by a script which builds the PDF and HTML versions using docbook2PDF and docbook2html and this script also runs as a cron job. This way we never have think of it and more important, the users always have the latest version of the documentation.

A similar script was created for generating the API doc in HTML format with Doxygen. This one wasn't set to run in a cron job since it sometimes needs manual editing. However, the API doc can easily be updated and uploaded to the webpage by running the script allowing us to do this on a daily basis.

The webpage is updated by a script with new packages immediately after they are builded and ensured to be error free. This makes the packages available for the hole world as soon as possible.