Custom Media Player with Integrated Feedback

Semester Thesis

Rowan Klöti

14th January 2011

Advisors: **Sacha Trifunovic**
Supervisor: **Prof. Dr. Bernhard Plattner**

Computer Engineering and Networks Laboratory, ETH Zurich

Abstract

In this work, a media player is designed that delivers feedback on user behaviour to a host application and allows user ratings and tags. It can be used in the context of an opportunistic networking system to deliver an implicit rating of distributed materials to control quality and prevent abuse. This media player is implemented on the Android mobile operating system. Moreover, to validate this approach, an analysis is performed on data obtained from Google's video platform YouTube.

Contents

Chapter 1

Introduction

In order to make use of an opportunistic networking system for media distribution, it is necessary to differentiate between desired and undesired content. A simple solution would be to make use of a a numerical rating system, such as the one employed by YouTube. However, most users do not go to the trouble to explictly rate items, regardless of how simple the rating system is made (e.g. Youtube's simple binary rating system). So rather than attempting to change user behaviour, which often proves unproductive, we may instead take advantage of information that can be obtained without any action initiated by the user himself (or herself). It may be assumed, for instance, that if a video contain's unwanted advertising (spam), malicious content or is simply not interesting that the user will not wish to watch the video in its entirety but rather will quickly cancel the playback. On the other hand, if the user finds the content especially interesting, he or she may watch it a second time or a third time. We may also infer the user's interest from more elaborate statistics, taking into account the user's seeking behaviour inside the content. This system of deriving a rating from user behaviour rather than an explicit valuation will be referred to use an *implicit rating*. In order to derive such an implicit rating, we must first collect information about user behaviour, in manner that does not obstruct normal usage and is ideally entirely transparent to user, otherwise we may encounter issues with user compliance. Therefore, a media player application is developed which automatically and transparently collects user data. It connects with a controlling application and forwards collected data to it.

To validate this approach, I have done a study on the popularity of YouTube videos, with the intention of linking viewing duration and popularity. I have examined both data obtained from crawling Google and data obtained by surveying channel owners (so-called *Insight* data).

A brief outlook: In chapter 2, I will examine existing papers and the media capabilities of mobile platforms, as well as providing a brief introduction to PodNet. In chapters 3 and 4, I will explain how I designed and implemented the media player solution. In chapter 5, I will briefly discuss how collected data may be used to derive a rating and review data collected on YouTube. In chapter 6, I will discuss what functionality may still be implemented as part of the media player as well as how further studies on YouTube may be conducted. I will also discuss an alternative option for collecting data from YouTube. Finally, chapter 7 briefly outlines what I have done.

Chapter 2

Related Work

In this chapter, I will briefly review two papers already written on the topic of popularity of online media sites, specifically YouTube. I will give an overview of the media capabilities of modern mobile media platforms, with a special emphasis on Android. I will also give a short introduction to PodNet.

2.1 Online media popularity studies

Chatzopoulou et al. [1] collected a large sample of data (~37 million video entries) on videos on the YouTube platform, recursively utilising the YouTube public API's "related video" feature and accounting for approximately a quarter of all videos on YouTube at the time. They correlate common metrics publicly available on the YouTube website, namely the number of viewings, the number of times a video is made somebody's favourite, the number of comments, the number of ratings and what the average rating is. They claim that

- The average rating does not correlate with other metrics
- All the other metrics correlate with each other
- One in 400 user views results in an action of some sort
- The network of related videos is a small world graph

It must be said that the available data do not allow any statement to be made about the number of users viewing a video, only about the amount of times a video is viewed[1].

Cha et al. [2] performed an investigation of sites containing "user generated content" (also including YouTube) and compared user behaviour to sites containing videos not supplied by the user (that is, containing commercial content). They compared popularity distributions, as well as investigating the efficiency of caching and the quantity of material in violation of copyright laws. In particular they

- Claim that the popularity of videos follows a power law distribution
- Derive similar numbers for user participation as were obtained by the other study

[1]See Evaluation (5.3)

2.2 The media capabilities of mobile platforms

Before designing any media player, it is essential to know what it should be able to playback. Modern mobile devices have integrated media platforms capable of playing and recording in a variety of formats, often rivaling the capabilities of a PC. Most phones use the 3GP format[2] to store media data, with some also supporting additional formats. I will briefly summarise support for video codecs in this section. Audio codec support is only listed for the Android platform. Most platforms support at least the MP3 format as it is the most prevalent way to distribute audio files on the Internet.

The Android media platform supports the MPEG-4 Simple Profile, H.263 and H.264 AVC codecs, the latter two also with the .mp4 file format[3]. Audio support includes AAC including HE-AAC, AMR (optimised for speech) as well as the familiar PC audio formats MP3, Ogg Vorbis as well as PCM/Wave. Android also supports playback of MIDI files.

Codec name	Support recording?	File formats
H.264	Yes	.3gp or .mp4
H.264 AVC	No	.3gp or .mp4
MPEG-4 SP	No	.3gp

Table 2.1: Supported video formats on Android platform.

Codec name	Support recording?	File formats
AAC LC/LTP	Yes	.3gp or .mp4/.mp4a
HE-AAC v1 and v2	No	.3gp or .mp4/.mp4a
AMR-NB and WB	Yes	.3gp
MP3	No	.mp3
MIDI	No	.mid and others
Ogg Vorbis	No	.ogg
PCM/Wave	No	.wav

Table 2.2: Supported audio formats on Android platform

The Windows 7 Phone platform supports the WMV9 (VC-1) codec in its Simple, Advanced and Main profiles, MPEG-4 Part 2 Simple Profile and Advanced Simple Profile, H.263 and H.264 (MPEG-4 Part 10 AVC) in its Baseline, Main and High profiles. The 3GP, MP4 and WMV containers are supported, depending on the codec used[4]. The Apple iPhone supports H.264 Baseline profile and MPEG-4 video formats in MP4, M4V and MOV containers[5]. The Palm WebOS supports [6] H.263, H.264 Baseline Profile and MPEG-4. The Blackberry

[2]See: `http://tools.ietf.org/html/rfc3839`

[3]See: `http://developer.android.com/guide/appendix/media-formats.html`

[4]See: `http://msdn.microsoft.com/en-us/library/ff462087`

[5]See: `http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/MediaLayer/MediaLayer.html`

[6]See: `http://developer.palm.com/index.php?option=com_content&view=article&id=1981`

supports WMV in its Simple Profile, H.263 and MPEG-4 Part 2 in its Simple Profile[7].

Operating system	H.263	H.264	MPEG-4	Other
Windows 7 Phone	Yes	Baseline, Main, High	Parts 2 & 10	WMV
Apple iOS (iPhone)	-	Baseline	Yes	-
Palm WebOS	Yes	Baseline	Yes	-
Blackberry	Yes	-	Part 2 only	WMV

Table 2.3: Overview of format support on other platforms

It is also worth noting that it may not be necessary to limit playback to formats supported by a built-in media platform. There are also third party media platforms, often open source, such as FFMPEG. There are third party players which utilise ffmpeg instead of the default media platform, offering a much larger set of supported formats. An example of such a player is the RockPlayer application for Android[8].

2.3 PodNet

The original motivation for the development of this media player was the extraction of implicit ratings for use on *opportunistic networking systems*. It would therefore be worthwhile to examine such a system. The PodNet content distribution system, developed at the Computer Engineering and Networks laboratory of the ETH Zürich[9], leverages the WiFi connections available on mobile devices to distribute user-created content (i.e. podcasts) to other interested users. The user simply subscribes to a podcast and it will be obtained from peers as soon as it becomes available. This spares users from often expensive and slow 3G connections, using free and plentiful WLAN bandwidth instead. It is also envisioned that PodNet will provide opportunities for spontaneous interaction of geograpically close users as well as allowing research into social behaviour of platform users.

[7]See: `http://www.blackberryfaq.com/index.php/What_video_and_audio_formats_are_supported%3F%3F`
[8]See: `http://rockplayer.freecoder.org/index_en.html`
[9]See: `http://podnet.ee.ethz.ch/`

Chapter 3

Design

As stated in the introduction, the aim is to develop a media player application which collects usage data transparently. In order to allow the media player to be integrated into existing plaformts, I have elected to implement it as an external application. Even though the media player is an external application, it is not intended to be run independently of a host application and this fact is reflected in design decisions that I have made.

The media player itself makes use of the built-in framework for playing media back on Android. The existing media player functionality is simply extended, the media player does not actually implement any kind of decoding or rendering on its own, limiting it to playing back those formats supported by the Android platform.

In order to collect that statistics, I have implemented a straightforward extension of the built-in video player, which introduces callback functions to supplement those already available. Making use of these functions, I can implement an object of my own that listens to events relating to playback and keeps track of user actions. This object is only dependant on my extended video player class, it can therefore also be integrated into other media player software, if need be.

I wish to maintain maximum flexibility by not integrating the media player into an existing application, while ensuring that it is capable of being integrated easily. Therefore I will also implement a simple control application. This application will pass necessary parameters to our media player, then receive the resulting data and display it to the user, acting in lieu of a host application. There are several options available to integrate the media player and the control application together: Here I will make use of the inter-process communication system offered by the Android platform. I will avoid the usage of a file or a database for communication so that a host application can manage the data as it sees fit. This also allows real-time updates of viewing information without superfluous file I/O.

The media player should provide cumulative statistics, so it will receive the stored set of statistical values when it is launched. It will then return the values added to the newly collected data, either when it is closed or otherwise interrupted. This process is idempotent, i.e. there is no harm in values being returned multiple times as we always return the running totals. It is also relatively resilient, there is nothing a user could do to prevent the correct operation of the data collection, aside from terminating the host application.

I could implement an IPC system that allows single values to be passed between processes, however that would require a large number of comparatively expensive cross-process function calls. Instead I will forward an entire table of name-value pairs to the media player at run time, then in return receive a table when the media player finishes or is interrupted (for instance by another process). This mechanism also provides a way by which the media player may be supplied with a file name or other arguments, especially the current position and state. This can be used to seamlessly continue playback if my application is terminated by the Android operating system due to a screen reorientation or the like.

Finally, I will provide the user with the ability to give media files a rating and assign tags to them. The tags are simply text strings (separated by semicolons), while the ratings are numerical (whole numbers from one to five). This information will be stored and forwarded in the same manner as the statistics described above. Furthermore, it is straightforward and possibly useful to forward information about video resolution and duration to the host application.

3.1 Data collected

It is necessary to decide which data should be collected. This is obviously limited by the implementation, specifically by the type of events that we can process. I have decided to collect data on the following quantities, all of which are passed to the host application as integers:

- The total playback time, which should be the central indicator of the user's preferences. The total is cumulative, so that if we play a ten second video twice, the result will be twenty seconds of playback time.

- The total rewind time. This is the total amount of time seeked backwards - the total of the differences between the place we have seeked from to the place we have seeked to, counting only seeks where the destination of the seek is *before* the source.

- The total time fast forwarded. This is as above, but now only counting seeks where the destination of the seek is *after* the source. These two statistics together give the total amount seeked, which I have decided not to collect separately — it can easily be derived from these two statistics.

- The total number of times a certain event occurs. The events supported include:

  - Playback is started
  - Playback is paused or suspended
  - Playback finishes normally
  - Seeking occurs. We also count rewinds and fast forwards separately, as defined above.

Chapter 4

Implementation

In order to fulfill the design specifications, I have implemented two Android applications, which are called CustomMediaPlayer and CustomControlApp. The former application is the actual media player and collects the statistics, the latter application allows the user to select a file name, launches the media player, then shows the results when the media player exits. These two applications need to communicate, so I have implemented a simple IPC interface called MediaInterface. The media player functionality is implemented in the MVideoView class while the statistics collection is implemented in the MVideoViewMonitor class. I will outline the functionality of these components in the following sections.

4.1 MVideoView class

The class MVideoView inherits from the class VideoView, which is supplied by the Android platform. This class makes use of the MediaPlayer system in Android, in particular, it can be connected to a MediaController, which provides a simple GUI for controlling media playback (play/pause, fast forward and rewind controls as well as a seek bar). The VideoView class is also capable of playing audio files, in which case the display is simply blank.

The MVideoView class is a lightweight extension to the VideoView class that provides a callback on invocation of all of VideoView's public methods[1], which effect a change in the player's state. The method setVideoViewListeners(VideoViewListeners) allows me to pass a reference to a VideoViewListeners interface to the MVideoView object. This interface can be implemented by the class that wishes to receive callbacks when MVideoView's public methods are invoked. If the method setVideoViewListeners is never invoked, MVideoView behaves exactly like its parent class.

4.2 MVideoViewMonitor class

The MVideoViewMonitor class implements the VideoViewListeners interface. The contructor of MVideoViewMonitor receives a reference to a MVideoView

[1]These are pause(), resume(), seekTo(int), start(), stopPlayback() and suspend()

object, it then calls the setVideoViewListeners method with itself as the argument. The class can then keep track of MVideoView's internal state. In particular, we store the position that the video is started from in SAVEDTIME, then when playback stops, we calculate the difference and add it to CUMULATIVEPLAYTIME. When seeking, we compare the value of the destination (the parameter of the SEEKTO(INT) function) to the current position. If the destination is larger (later), we increment the value of FFTIME by the difference. Otherwise, the value of RWTIME is incremented instead. Moreover, counters are maintained for each of MVideoViews public methods, and separate counters are maintained for fast forwards and rewinds[2].

All of the collected values can be obtained through access functions and the values can be reset to zero by the RESET() method. The class VideoView (and therefore MVideoView) provides an ONCOMPLETED() callback which we make use of. I therfore provide a callback ONFINISHED() with its own interface ONFINISHEDLISTENER, so any class making use of MVideoView and MVideoViewMonitor can itself receive notice when the video playback is finished without interfering with data collection.

4.3 CustomMediaPlayer class

The class CustomMediaPlayer is the actual media player application itself. Specifically, it is an Activity, that is, it inherits from the Android API's ACTIVITY class. The Activity class provides a framework for implementing the part of an application that actually interacts with a user, i.e. its GUI. In general, the layout of the Activity is defined in an XML file. The Activity can then load this layout with the SETCONTENTVIEW(INT) method. It is then possible to obtain references to individual UI elements (called *views*) with the method FINDVIEWBYID(INT). It is also possible to instantiate views programmatically, in which case they can be added to a layout with the ADDVIEW(VIEW) method.

The Activity class provides several methods which are invoked when the state of an Activity changes and which can be overriden by a child class[3]. The ONCREATE() method is used to setup the layout, create necessary objects and change settings as needed. Here I do not make full use of XML based layouts, as there is no way to create an MVideoView object from the XML interface. Instead, the objects are created and added to the existing layouts. The RATINGBAR view, used to provide the user a possibility to give videos a rating, is created and added to the layout in a similiar manner, in this case I only add the rating bar if the orientation is vertical. In this case, I also add a TEXTVIEW to display any tags which have been assigned to the media files, and a BUTTON which shows a dialog box so that the user can enter new tags.

The final act of the onCreate method is to attempt to connect to the service provided by the controlling application and start a timeout counter. The ONDESTROY method is used to ensure than the Activity cleans up after itself

[2]The access methods available are GETCUMULATIVEPLAYTIME(), GETFFTIME() and GETRWTIME(), which return values in milliseconds, and GETPAUSECOUNT(), GETRESUMECOUNT(), GETSEEKCOUNT(), GETSTARTCOUNT(), GETSTOPCOUNT(), GETSUSPENDCOUNT(), GETFINISHEDCOUNT(), GETRWCOUNT(), and GETFFCOUNT(), which return the number of times that the event referred to has occured. All returns are of type integer.

[3]See appendix A.2 on page 18 for list of method an when they are invoked.

correctly. If the application manages to connect to the host application successfully, the onServiceConnected(ComponentName, IBinder) callback is invoked, which cancels the timeout counter then receives data through the established service connection. The data includes the file name to be opened as well as any statistics we collected in a previous session, so we can take care of the accumulation and not leave this to the host application. If the service connection is not established within the specified timeout, the media player exits and launches the control app, under the assumption that the failure to connect to the service means that the control app is not running.

When the MVideoView object is finished loading the video, it calls the onPrepared(MediaPlayer arg0) callback, where we restore the state of the media player if we previously stored it due to the player being destroyed during a screen reorientation. At this stage, the user can interact normally with the activity. When the method onPause() invoked, the collected data is obtained from the MVideoViewMonitor, added to the existing data and sent through the IPC connection.

4.4 CustomControlApp and MediaInterface classes

In order to implement an IPC receiver on the Android it is necessary to implement a service by inheriting from the Service class. Like Activities, Services constitue part of an Android application. They are intended to perform background operations and receive IPC connections by implementing onBind callback. This callback provides the necessary IBinder interface to establish the connection and also allows the service to maintain multiple connections. We use two connections, a local connection from the CustomControlApp activity and a remote connection from the media player. To implement a remote connection, it is necessary to design an interface using AIDL[4](here: MediaInterface) then implement the interface in the service as an inner class (here: rBinder). The service is connected to with the bindService(Intent) method, and the application containing the service must publish the Intent[5] via its Manifest in order for external applications to be able to connect to it.

MediaInterace implements two methods, allowing the media player to receive a set of values from the service, in the form of a Bundle object, as well as to forward a set of values back to the service. The Bundle object is a container provided by the Android API specifically for use in IPC; it uses a HashMap internally.

The local API is built up in a similiar manner, so that the CustomControlApp can push its stored set of statistics, as well as the filename which the user has entered, to the service, from which the media player obtains those values, and then when media player exits, it sends the values to the service, then when the control app resumes, it gets the values from the service, parses them and displays them.

[4]Android Interface Description Language

[5]An Intent represents a sort of URL: An application to run plus associated data (if applicable). This is necessary as the Android platform does not have a way of passing arguments to applications.

Chapter 5

Evaluation

Once sufficiently detailed statistics on user behaviour have been obtained, it is possible to derive some sort of rating from them. A rating should take into account the number of times a user watches the content. This could be done by counting the number of times playback is started. A more subtle method might transform this number into a continuous quantity by dividing the total playing time by the actual length of the video. Furthermore, any such algorithm should take into account that user behaviour is likely to vary with the length of the video, and watching a long video entirely or even multiple times might be interpreted as being more strongly approving than with a shorter one. Seeking behaviour could also be taken into account - for instance users may fast forward because they are bored or rewind because a certain section of the video was of particular interest. In the absence of a field test to validate an implicit rating, I have performed an analysis on YouTube data instead. The aim is to attempt to validate the approach of collecting data to assign implicit ratings.

5.1 Methodology

YouTube provides a public API called “Data API”[1], allowing automated interaction with the YouTube platform. This API also allows some statistical information to be gathered - the number of views, the number of users who made a video a favourite, the number of “likes” and “dislikes”. However, more detailed information is available via the Insight feature, intended for marketing analysis[2], which provides more detailed statistics, such as the number of *unique* views per video, as well as demographic data. Unfortunately, this data is only available to the video owner. In order to obtain some data, a survey was performed and YouTube users with widely subscribed channels were asked to provide Insight data. In addition, a program was developed to crawl YouTube, gathering information on popular videos. This program takes an initial feed[3] and then iterates through all the entries, adding the “Related videos” feed for each entry, then proceeding to the next feed. The program ensures that video entries appear only once in the output.

[1]See: `http://code.google.com/apis/youtube/overview.html`

[2]See: `http://www.youtube.com/t/advertising_insight`

[3]The feed used in this case was the standard “Most Viewed” feed

5.2 Data gathered from YouTube crawling

The YouTube application made 10,000 requests for information from YouTube, yielding 3433 video entries[4], each containing the number of “like” and “dislike” ratings, the number of times a video is made favourite and total number of views. I have obtained a figure of 486 views per favourite and 611 views per rating, which agrees well with the claims from the reviewed papers. Comparing views and ratings in Figure 5.1, it is evident that the two are not correlated in any meaningful way, that is, popular videos are not necessarily highly rated. There is a cluster of videos with a high rating, tending to suggest that users tend to give relatively generous ratings - 87% of all ratings given are a “like”. The mean rating was 4.42 with a median of 4.70 and a standard deviation of 0.71.

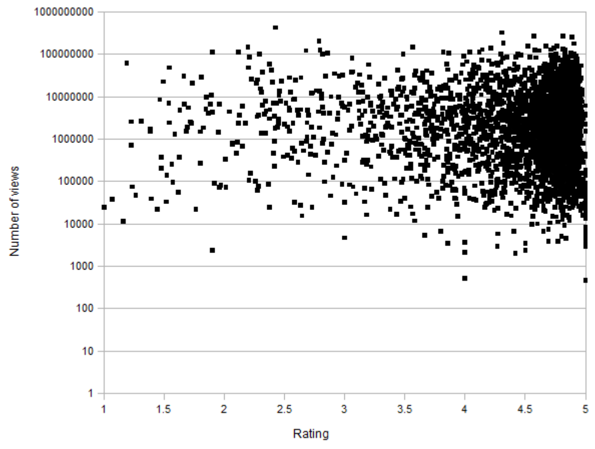

Figure 5.1: Number of views vs. rating

I have also compared the number of ratings given with the rating in Figure 5.2. It is once again apparent that the average rating is quite high. It might be assumed that the number of ratings have some correlation on the average rating - videos which get rated a lot are probably popular and may get higher ratings on average. The data suggests that no such connection exists, that content that is rated more often is not rated any different from content rated less often.

[4]Performed at 18:06 on 08.01.2010

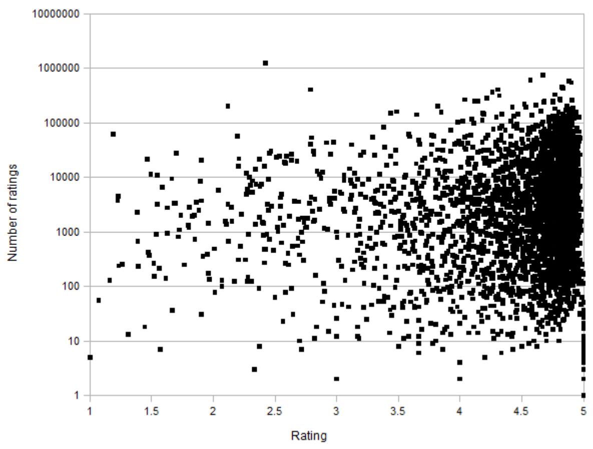

Figure 5.2: Number of ratings vs rating given

In Figure 5.3, I have compared views and favourites. The nearly linear form suggests that favourites are mostly proportional to the number of views. There are numerous outliers above the main line, while the form of the lower boundry is very sharp. Clearly there is a relatively sharp upper limit on the number of favourites a video gets (which is much lower than the number of views). On the other hand, the lower limit is much more diffuse.

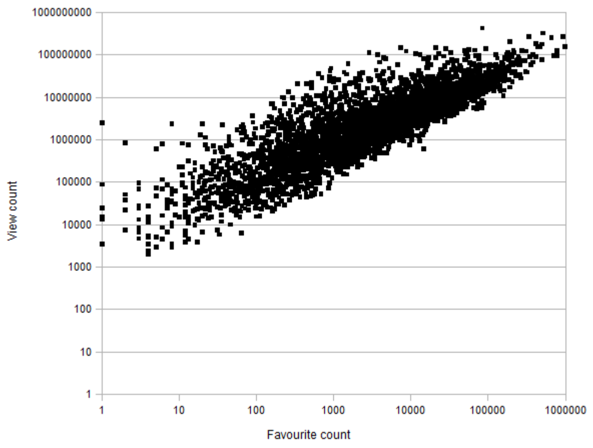

Figure 5.3: Favourite count to views count

Finally, I have compared the favourites-to-views ratio with the rating in Figure 5.4, under the assumption that the favourites-to-views ratio represents a sort of implicit rating. Interestingly, it is possible to observe a sort of one way relationship here. Lowly rated videos rarely become users' favourites - unsurprisingly - but highly rated videos show a great deal of variation. It may be that the much larger number of highly rated videos allows for a wider distribution of values.

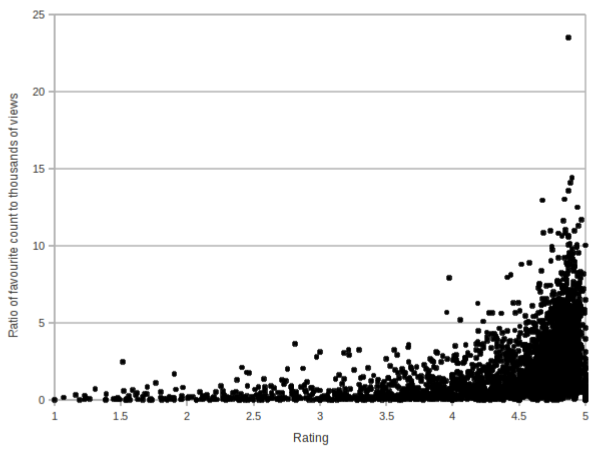

Figure 5.4: Ratio of number of favourites to view count vs. rating

5.3 Data gathered from user survey

Unfortunately, there were not enough responses to the survey in order to provide statistically significant data (2 responses for 102 requests - 1.96%). Based on the limited data available, one point can still be made. One user had 82 unique views and 257 total views (just under 1 unique view for 3 total views), while the other had 845 unique views and 5423 total views (a ratio of less than 1 unique view to 6 total views). These figures are for a period of 2 weeks and per channel (not per video).

5.4 Discussion

Although the results obtained here do not contradict the statements made in [2, 1], there is a large discrepancy between the number of views a video has and number of users who have viewed a video. The discrepancy may be a result of the YouTube page being loaded multiple times, perhaps because the page is reloaded. It may also be a result of session restore reloading many pages, without the user viewing them. In any case, it would tend to suggest that the publicly available data are not adequate for detailed popularity analysis. Based on the figure of around 500 views per rating, claimed by both papers and also reflected here, content viewed by a small number of people may never even get a rating. This certainly makes some kind of automatic system sensible. Also, for future YouTube studies - given that neither approach used has yielded data in sufficient quantity and quality - it would be worth trying an alternative, client side solution. Such a solution could built on the CustomMediaPlayer developed in this thesis. It could be distributed as an application on mobile systems. It would monitor user behaviour on YouTube (or another video site) and automatically report to a remote client, or store the data and allow it to be collected cumulatively at the end of the examined period.

Chapter 6

Future Work

6.1 Media player

The media player application must be integrated into a host application, which would have to manage the data collected by the application as well as derive actual implicit ratings from the collected data, which should be trivial, although the rating system should be validated in some way, e.g. by correlating the ratings with explicit ones. It could also communicate the relevant data to a remote server, which would allow transparent collection of data pertaining to user behaviour. It is even possible that the application could learn user preferences, assigning implicit ratings based on previous explicit ratings, although this would probably require extensive data mining.

It would be preferable if the data collected here were also available for web-based video playback. It is thinkable that a site such as YouTube (or a comparable site providing user-created videos) would add functionality to its video player so that playback time could be exactly recorded. This would also aid in the creation of implicit ratings. It could make use of plugin-based players, such as the Flash based player used by YouTube (although Silverlight would work just as well, albeit limiting the video playback to systems which support Silverlight), but it could also make use of the new functionality added by HTML5, in particular the VIDEO element, which can interact with JavaScript code.

6.2 YouTube study

The scope of YouTube study was limited by the methodology and might be seen as a sort of proof-of-concept for a much larger and more extensive study to be performed at a later date. In particular, obtaining YouTube's cooperation would greatly increase the quantity of the data collected and remove the necessity of obtaining user cooperation - always difficult on a large, relatively impersonal website. It would be necessary to ensure the anonymity of YouTube users, and YouTube would be more likely to cooperate if it had some sort of incentive. Obtaining YouTube's cooperation would have been well beyond the scope of the small study done here. A client based solution, as discussed in the Evaluation (5.4 on the previous page), is an alternative that should be evaluated.

Chapter 7

Conclusion

In this semester thesis, I have designed and implemented a media player application for the Android platform which collects data on user behaviour. I have developed a backend application, which launches the media player and acts in lieu of an integration into a larger software project. I have also done a investigation into videos on YouTube to validate the use of user behaviour to rate video content.

The media player application integrates a modified media player component utilising the Android platform, as well as the component that uses the modified media player component to derive statistics on user behaviour. In hindsight, this functionality might have been merged into a single component, which could also be used as a drop-in replacement of the default video widget in other projects, although the solution chosen here can also be integrated into other projects without great difficulty. The media player uses IPC to communicate with a backend, which also allows the backend to be replaced without modifying the source code of the media player. This solution was chosen for maximum flexibility. The data is passed as a table, which allows easy extensions without modifying the API. It also offers good performance, keeping the number of cross-process function calls to a minimum. Finally, the media player itself has some workarounds to adapt to changes in screen orientation. Here performance improvements could make the process faster and more seamless, although the benefits seam small compared with the additional complexity.

The YouTube study was composed of two parts. Firstly, data was gathered automatically by crawling. Secondly, a small survey was performed. Unfortunately, the results of the survey were disappointing, if not entirely surprising, and it was difficult to make inferences from such a small data set. The data gathered by crawling provides only rudimentary statistics and does not add anything substantial to results already obtained by others[1, 2]. A case was made for the development of a client side solution. The approach of using an application such as CustomMediaPlayer was validated, given the relative scarcity of user ratings and the minimal accuracy of simple view counting, such as YouTube provides.

Appendix A

Further information about design and implementation

A.1 Running the applications

Anyone who wishes to modify and/or compile the Android applications should observe the following:

- I developed the applications for Android 2.2, with Eclipse 3.5. It may be possible to get them to work with other versions.
- Obtain the Android SDK at `http://developer.android.com/sdk/index.html` and the appropriate version of Eclipse (Galileo) at `http://www.eclipse.org/downloads/packages/release/galileo/sr2`.
- It is best to follow the instructions at `http://developer.android.com/sdk/installing.html`.
- Once the two projects have been imported, it is necessary to manually link or copy the AIDL file, situated in CustomControlApp/src/android/CustomControlApp to the directory CustomMediaPlayer/src/android/CustomControlApp. It is important not to copy it to the directory of the rest of the CustomMediaPlayer source files - it must be in the same package as CustomControlApp or CustomControlApp will not be allowed to connect to it.

Anyone who wishes to modify and/or compile the Java application used to crawl YouTube should observe the following:

- It is necessary to obtain the GData API from Google. Please see `http://code.google.com/apis/gdata/articles/java_client_lib.html`. Further packages are required - servlet-api.jar, mail.jar and activation.jar. These must be obtained separately, as detailed on the above. It is necessary to include them in Eclipse.
- It is necessary modify the source code to change operating parameters, specifically the number of queries to make (not too large) and the feed to start off with.

A.2 The Activity class

Method	Invoked when...
ONCREATE	The Activity is first created, essentially like a constructor
ONSTART	When the Activity becomes visible
ONRESUME	When the Activity comes to the foreground
ONPAUSE	When the activity is no longer in the foreground
ONSTOP	When the Activity is no longer visible
ONDESTROY	When the Activity in shut down, essentially like a destructor

Table A.1: The various methods that are invoked when the state of an Activity changes. Source http://developer.android.com/reference/android/app/Activity.html

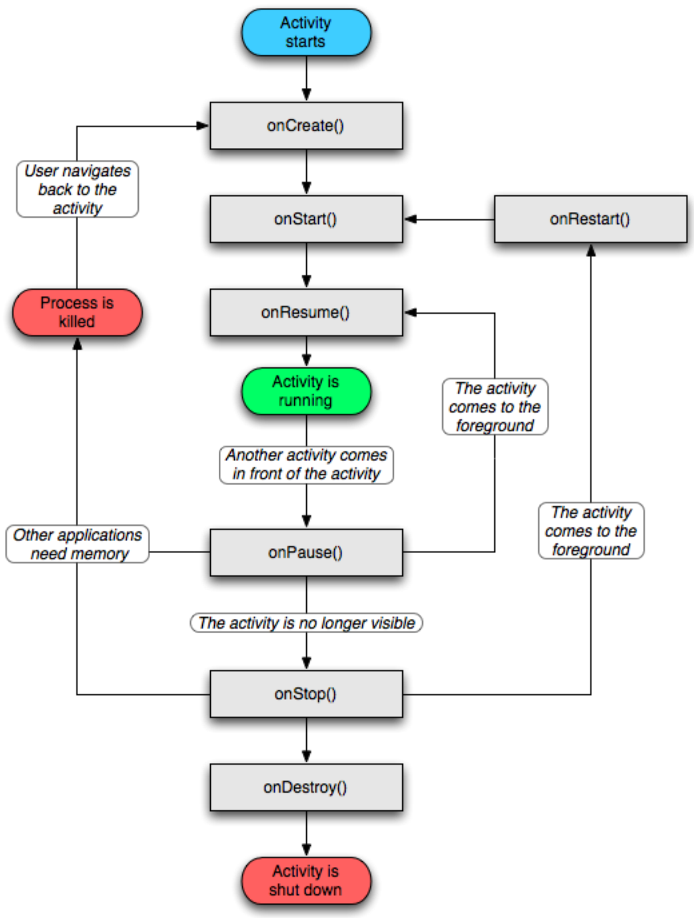
Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure A.1: This diagram from the Android developer's guide show's the lifecycle of an Android Activity as a flow chart. Source `http://developer.android.com/guide/topics/fundamentals.html`.

Bibliography

[1] Gloria Chatzopoulou, Cheng Sheng, and Michalis Faloutsos. A First Step Towards Understanding Popularity in YouTube. In *IEEE NetSciCom Infocom Workshop*, 2010.

[2] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong yeol Ahn, and Sue Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *IMC*, 2007.