Automatic Mobile Video Director

Alexander Egurnov
University of Mannheim
aegurnov@mail.uni-mannheim.de

Thilo Weigold
University of Mannheim
tweigold@mail.uni-mannheim.de

Jon Pettersen
University of Oslo
jonup@student.matnat.uio.no

Alf-André Walla
University of Oslo
alfandrw@ifi.uio.no

***Abstract*—The abstract goes here.**

I. Introduction

In the last years the Internet has changed rapidly. Users not only retrieve information of simple websites anymore, rather they are sharing information, videos and pictures all over through networks. With the "Web 2.0", user-generated content has become an important part of the Internet we know nowadays. Well-known platforms like, YouTube, Facebook and Twitter make it easy for regular people to distribute data and media.

With the changing role of mobile phones from simple communication devices to devices with a multiplicity of functionalities, user-generated content has even seen a faster growth in recent years. Due to more precise sensors, integrated high-resolution cameras and of course faster mobile network technologies, it has never been so simple to contribute content when ever and where ever you want. The ubiquitous use of mobile phones leads to complete new opportunities for web services using data and user information, in order to create dynamic content.

Let us imagine an interesting, public event like a political speech, concert or any kind of sport event, which is not filmed and streamed professionally by TV channels. Fortunately it has become common of spectators nowadays to capture parts of ongoing events by their phone camera. The potentials are tremendous. Why not using these user-generated media to provide a real-time stream or list of videos of the ongoing event, which not present people could therefore follow anyhow. A high quality mash up of an event requires an complex selection of all potential videos available, in order to cover the whole event with the best quality provided and being aware of bandwidth constraints of the mobile network.

In the following paper we want to deal with this kind of situations and introduce our implemented approach of an Automatic Mobile Video Director. Thereby we focus on the interaction between multi clients and one central server, our automatic video director, while addressing bandwidth and complexity problems, already mentioned above. Furthermore we do not focus on retrieving sensor information, which is done only in a rudimentary and simple way.

II. Related work

As mentioned above the interest of the public to capture and share videos from public events has increased with the introduction of mobile devices with high resolution cameras and wireless internet connection. As such there has also been a huge increase in the number of research projects trying to make this task easier and to reduce the strain represented by the uploading of such videos. (The Mobile Vision Mixer) proposes a manual way of making these videos where five people are able to set up a connection between their mobile devices, assign roles to each device, one is assigned the role of director and the others are cameramen, and capture an event taking place where they are. Others (On Demand Retrieval of CrowdSourced Mobile Video), (Automatic Mashup Generation) uses an algorithm to make the director automatic. With this solution you have to find a way of deciding which of the videos received, is the one covering the event in the best way and with the best picture quality. They propose using metadata from the devices sensors to evaluate the quality of the videos instead of uploading all videos from an event to the server and going through the video frame by frame. This also enables us to just send the best videos to the server, discarding the rest which saves a lot of bandwidth. This is pointed out by (On Demand Retrieval of CrowdSourced Mobile Video) as a serious concern at big events like the SuperBowl.

Another area that has been investigated is the task of identifying what is called region of interest or point of interest. These terms describe areas of the event that are more interesting than others. We probably will find a video of a concert more interesting if the camera is pointing towards the stage than one pointing somewhere else. Several solutions have been proposed for solving this problem. You can go through frames of the video to see if they resemble the main attraction of the event, like a stage on a concert, or the singer on that stage. This is used in (). However the reliance on the sensors in the mobile devices for discovering such regions seems to be a much more preferred method. (Sensor-based Analysis of User Generated Video for Multi-camera Video Remixing) uses the built-in compass found on mobile devices to make sure the camera is pointing in the right direction. This system also uses the compass to detect if the camera changes direction. They make the assumption that if the camera is suddenly turned to face a different way, the user has probable spotted something interesting. Detecting this change in direction allows for a more interesting viewing experience. The detection of something interesting also goes as far as detecting which activity is being filmed. This is used to relive the managers of such systems from the trouble of marking each video with appropriate tags. (Sport Type Classification of Mobile Videos) has created a system that is able to detect which of several sports is being filmed while (MoVi) is able to detect when a social happen is taking place in from of it and start recording whenever it detects one. (MoVi) analyses sound to detect, among other things, laughter.

III. Methodology

A. System Overview

In this section we briefly want to give an overview of our system throughout. Detailed discussions of our implementation will follow in the subsections afterwards. To implement our idea of an Automatic Mobile Video Director it was necessary to find the right structural organization for our system. Restricted requirements and the logical distribution of tasks leads us to the decision to realize our service in an conventional client-server architecture. Figure 1 shows our general system architecture, consisting of multiple clients and one central server part. This architectural set up entails some important advantages. Client-Server gives us the possibility to strictly separated tasks and work between clients and server. Compared to other possible architectures, like peer to peer, we are able to centralize all resources and important computation in one place now. Resources, like battery and computation power can be saved on client side, while the hard work has to be taken on server side. In peer to peer networks clients would have to communicate with each others, which would produce big communication overhead to weight of important resources. However these resources are reserved and needed for recording videos and sending them, if necessary to our central Video Director. Therefore we think an client-server architecture fits the requirements of the Automatic Mobile Video Director in the best way.

B. Client application

In a nutshell, the client side application “”Automatic Video Director“” consists of 4 main logical components: The user interface UI, the HTTP client connection part, Camera and Sensor retrieving and a persistent database. The application is implemented using the official Android SDK, provided by Google Inc..

Great effort of our client development was raised on the client and server communication part. As an communication protocol between client and server we decided to us exclusively Http. For realization, Android offers two main solutions: Apache Http Client or HttpURLConnection.(reference Android) Since Google is not supporting Apache Http Clients anymore, we decided to use the HttpURLConnection Client(Reference: Android DEV). In order to ensure a smooth work flow of the android application, we make use of specific thread techniques. First it was necessary to prevent network operations to block application’s user interface, which would later result in poor user experience. Therefore all network operations need to be run as an instance of the specified HttpAsnycTask class, a heredity of the AsyncTask class. This means that all HTTP requests run asynchronous in the background of the application’s main thread. Another important task of our application, is the automatic checkup and upload of selected videos, even when users leave the application and multitasking is done virtually. Such an feature can be realized in Android by implementing a service class, which handles background computations in the main thread. All HTTP_GET requests for retrieving server notifications are called within a separate thread of the service, as well has the HTTP_PUT method for uploading the video file to the server. By doing so we make sure, that all network operations run independently of our main thread, while fulfilling their tasks.

Further core function of the application is the recording of videos. Using Android’s “”MediaRecorder“” class we are able to handle access to the phone’s build-in camera as well as the recording and storage of video files. Simultaneously sensor data is tracked as well. Thereby we use the observer pattern. Sensor detection classes (ShakeDetection.class Tilt-Detetcion.class) are implemented as Observables, while the CameraActivity class registers as a single observer. All in all we have decided not to focus on sensor detection and retrieving. A lot of research has been already done in those parts. Due to the shortage of available time we only retrieve some basic accelerometer data, in order to determine the amount of shaking and weight of tilt, represented by simple integer values. The sensor data is part of the meta data which is sent to the server for directing the video upload process. Hereby the amount of shaking is a counter which is increased by one, when ever shaking is detected. We assume to be in a shaking motion, when at least two of three accelerometer values have changed to their previous values by a specific threshold. The threshold value is initializes as a float literal with the value 0.7f; This value is exclusively based on experimenting.

In order to keep track and coupling of stored video files and obtained meta data, we make use of a persistent database, implemented with the lightweight SQLite library, which has support from Android side as well. The local database set up would not have been entirely necessary. But due to considerations for future extensions, we decided to have all data in persistence. For Future work check out section “Future Work”; Each meta data entry has its unique id and each video file which is stored has its unique filename. Therefore we use the database column filename as a unambiguous reference to a specific video file. When the client gets notified by the server to upload a video, it is necessary to check the filename.

C. Server application

Server general description goes here. Video storage, database connection, server framework description. RESTful Web Service

D. Client-server interaction

BRAINSTROMING: HTTP protocol,

1) Protocols: Our Automatic Mobile Video Director server implementation provides a general interface to applications which wish to interact with it. It is implemented through HTTP requests to certain server locations result.

GET /events

Lists all events (including videos) in JSON.

GET /event/*id*
: Returns Event (including videos) in JSON.

POST /event/new
: Create new event from JSON. Expects request body to be a JSON string containing attribute *name*.

POST /event/*id*
: Upload JSON metadata about a video for Event with given *id*.

PUT /video/*video_id*
: Upload video *video_id* from Event *id*. Expects request body to be a file stream containing a full video file.

GET /video/*video_id*
: Retrieve video *video_id* from Event *id*.

GET /selected
: Retrieve a list of selected but not yet uploaded videos in JSON.

E. Metadata format

JSON vs XML arguments here As a final result we should state that metadata is transferred in JSON format.

id
: Client-side unique identification of the video.

filename
: File name in client's local file system.

timestamp
: Video creation time.

duration
: Video duration in frames.

width
: Video frame width in pixels.

height
: Video frame height in pixels.

shaking
: Amount of shaking detected by sensors.

status
: Video status. Indicates video life cycle phase.

serverId
: Server-side unique identification of the video. Needed for coordination of all clients.

IV. Video Director Algorithm

A. Video life cycle

B. Selection algorithm

V. Evaluation

How good/bad it is.

A. Data Traffic

Thilo can test it. I will have a look to this. If someone has experience please let me know.

B. Battry consumptio

Who wants to test it?

C. Selection criteria

VI. Future Work

Put down all the awesome ideas we have.

VII. Conclusion

The conclusion goes here.

References

TABLE I
TASK DISTRIBUTION

Part	Task	Subtask	Responsible
Android application	Video Capture		Thilo Weigold
	Sensor data collection		Thilo Weigold
	Metadata class		Thilo Weigold
	Http Client	Post Method	Thilo Weigold
		Cookies, Callbacks	Alexander Egurnov
		Get & Update methods	Alexander Egurnov
	Background upload service		Thilo Weigold
	SQLite database connection		Thilo Weigold
	Preferences		Alexander Egurnov
	GUI		Thilo Weigold, Alexander Egurnov
	Client-server data exchange		Alexander Egurnov
Server application	RESTful Server application	Client authorization	Alf-André Walla
		Request processing	Alf-André Walla
		Video and Event logic	Alf-André Walla
	MySQL database connection		Jon Pettersen
	Video Director		Alf-André Walla
	Client-server data exchange debug		Alexander Egurnov
	Video upload		Alf-André Walla, Alexander Egurnov
Web Server	MySQL Administration		Alexander Egurnov
	Nginx setup for streaming video		Alexander Egurnov
Documentation	Basic template formatting		Alexander Egurnov
	Introduction		Thilo Weigold
	Related Work		Jon Pettersen
	System Overview		Thilo Weigold