First version of UGC capture tools

Deliverable D3.3.1

ICoSOLE identifier:	ICoSOLE-D3.3.1-BIT-FirstVersionOfUGCCaptureTools_v1.doc
Deliverable number:	D3.3.1
Main author of Deliverable:	Reinhard Grandl, Christian Timmerer (BIT), Marcus Thaler, Werner Bailer (JRS), Rik Bauwens (VRT)
Internal reviewer:	Mario Sieck, Uwe Riemann (DTO)
Work package / task:	WP3 / T 3.3
Document status:	Final
Confidentiality:	Public

Version	Date	Reason of change
0.1	2015-03-02	Document created
0.2	2015-03-31	Content Analysis
0.3	2015-03-31	Marconi Moments Capture App
0.4	2015-04-01	Capture on Devices
0.5	2015-04-09	Uploading and Streaming
0.6	2015-04-20	Structural Overview of the App
0.7	2015-05-05	Updates Sensor Capture/Metadata
0.8	2015-05-08	Version for internal review
1	2015-05-26	Final version

The work presented in this document was partially supported by the European Community under the 7th framework programme for R&D.

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

Table of Contents

1 Introduction

ICoSOLE will develop an approach enabling an immersive experience of live events, which are spatially spread out. The approach is scalable in terms of content capture. In order to make the approach applicable for events where not all venues of interest can be covered with professional equipment, user generated content (UGC) captured with mobile devices (e.g., built-in cameras of smart phones, action cameras) will be supported. The content will be streamed from the mobile devices to the production system adopting network-adaptive (trans-)coding and streaming techniques.

The above is an excerpt from the ICoSOLE project's Description of Work (DoW) that highlights the significance of user-generated content. Figure 1 shows the conceptual diagram of the ICoSOLE system architecture – with the framed UGC capture part on the left.

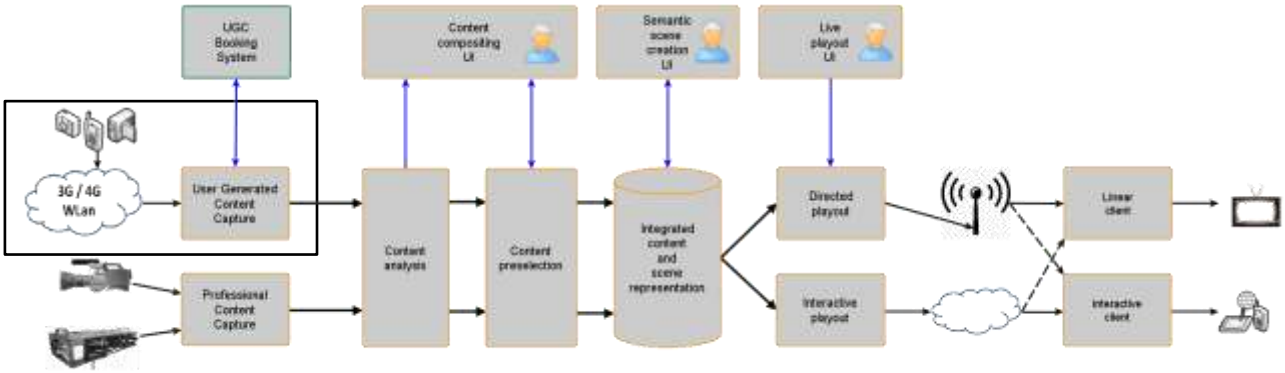


Figure 1 Conceptual diagram of the ICoSOLE system architecture, with the framed UCG capture part on the left.

In context of ICoSOLE, UGC capture covers the generation of audio and video, as well as metadata on consumer/prosumer devices (smartphones and tablets). To meet the quality constraints of the project, on-device analysis of capture essence, based on metadata generated by in-build sensors of the device, will be done.

The developed capture tools will also deal with the communication with production management, as well as transport of captured essence (audio, video and metadata) to a central storage and processing unit on site. Considering mobile devices, like smartphones, one has to deal with tremendous bandwidth fluctuations within mobile networks. Especially in the case of live streaming, this will be challenging, as the multimedia data must be delivered in time. If the requirement of real time up-streaming can not be met (e.g. in cases where the uplink channel has not enough capacity to deliver even a low-quality representation), the captured essence should be stored on the device for later transmission to the system.

List of Figures

List of Tables

2 Executive Summary

In this document we describe the current status, as well as an outlook of UGC capture tools developed within the project. To meet the requirements, as stated in the DoW, an Android based application, with focus on high quality content generation (ensured by on-device analysis of captured essence) will be implemented. Also low latency, especially for live scenarios, and an easy to use and intuitive user interface is of high interest.

We split the application design in different parts, (a) input/capture, (b) quality analysis and (c) upload/streaming. Different approaches for these parts have been developed and evaluated by partners since the start of this task.

In terms of audio/video capture, we evaluated two different ways: either using Androids *MediaRecorder* API, which is easy to use, or the *MediaCodec* API provided by newer Android devices, which enables greater flexibility, when it comes to content analysis. We choose to pursue the second approach, having in mind that real-time quality analysis on the captured essence will be done.

For quality analysis, we make use of the available sensors of the mobile device, like accelerometer to detect fast and shaky movements of the mobile device and, thus, unstable image sequences. Furthermore, algorithms for sharpness, noise and over-/underexposure detection have been implemented. If the recorded content does not reach the desired quality levels it gets discarded. In order to encourage users to optimize the quality of their content, they receive immediate feedback on the quality of their recordings.

For uploading/streaming of captured essence we choose to evaluate session based streaming libraries for RTMP and RTSP in terms of their practicability for the project. Due to incompatibilities and a lack of support for our use-case, we developed another approach, which is session free and based on the HTTP protocol, with great similarities to MPEG-DASH, which we will pursue further.

Based on the findings of our evaluations, and user feedback of a first test shoot (the *Marconi Moments* in October 2014), we will improve the application and work on the integration of the different parts, implemented by partners.

3 Introduction

3.1 Purpose of this Document

This document presents concepts and first implementations of user generated content tools, related to audio and video capture. Focus is given on ensuring quality, synchronization and on-device analysis of captured essence. Furthermore, insertion of user metadata and communication with production management, as well as essence and metadata transport to a central storage and processing unit on site, is covered.

A basic set of functionalities is presented and an outlook on planned activities is given, which will be covered within the final deliverable.

3.2 Scope of this Document

This document covers a description of user generated capturing tools in the context of capturing, quality analysis, and streaming. Functionalities as well as considerations about different approaches for specific parts are presented. Improvements and future work in the context of UGC capturing tools are prospected.

3.3 Status of this Document

This is the final version before review.

4 Structure of the Capture Application

In context of UGC capture tools, an Android based capture application will be implemented. The main features are: (a) audio and video recording, via the in-build microphone and camera respectively, (b) metadata capturing from different sensors available on the device, (c) on-device analysis of captured essence to meet quality constrains, (d) en-/transcoding and packaging of recorded content and (e) the up-streaming functionality to servers for processing. The high-level architecture of the capture application, as shown in Figure 2, is divided in three main parts: Capture, Processing and Streaming.

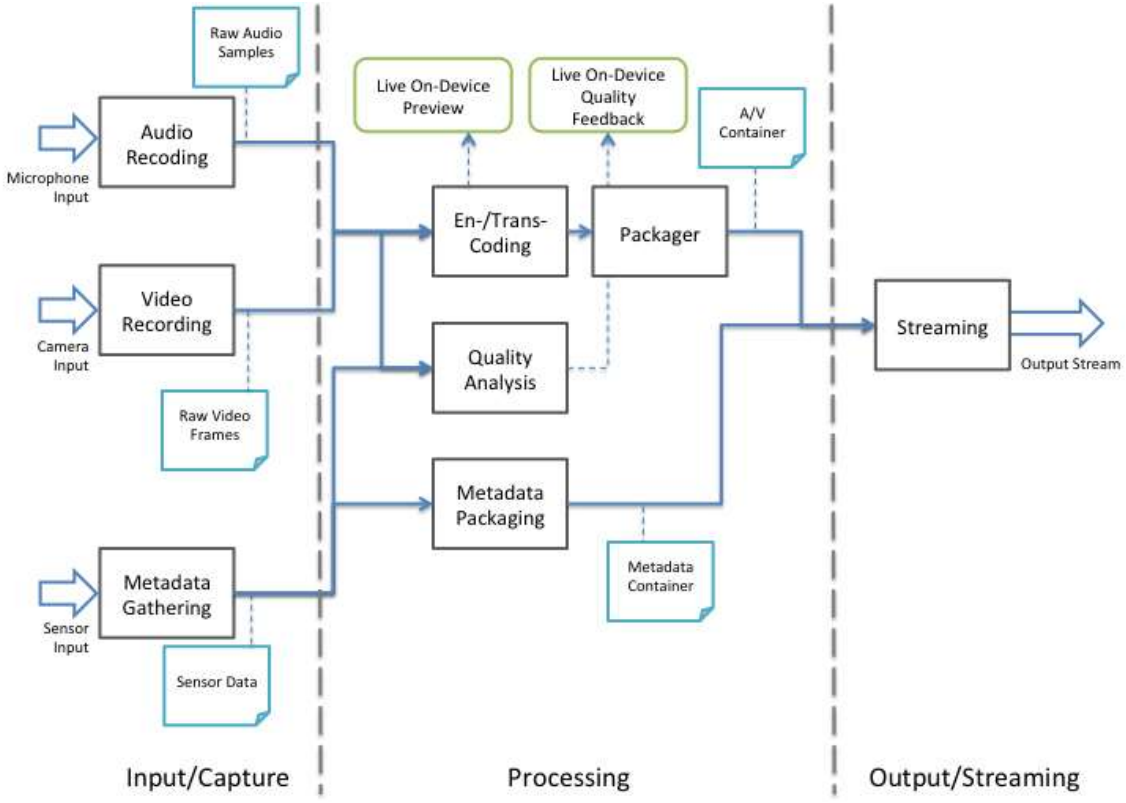


Figure 2 High-level overview of the capture application.

The input or capture part is further divided into audio/video recoding and metadata gathering. Raw audio- and video samples are forwarded to the encoding stage and a live preview of the recorded essence is shown to the user. The packager further processes the encoded frames and an audio/video container is created. Also raw sensor data is written to a container-file. We describe the input and audio/video processing component in more detail in Section 5.

The processing part performs also quality analysis of the captured essence, as described in Section 6.1. The user gets noticed about potential quality issues, using a basic live feedback, which allows the user to react on the issues (e.g. someone stands in front of the camera) immediately, as stated in Section 6.2.

Finally, the app provides streaming functionalities for captured clips together with the metadata. Different approaches have been implemented and evaluated using session based streaming protocols, as well as HTTP uploading, as described in Section 7.2.

So far different components of the application have been implemented and evaluated by partners separately and the integration is left for future work within the project, as we address in Section 9.

5 Capture on Consumer/Prosumer Devices

In order to encourage users to contribute content, the capture has to be as simple and intuitive as possible. Thus we have implemented a capture app that records not only the audiovisual content, but also sensor metadata, which helps aligning the captured content with the scene.

5.1 Audio/Video Content Capture

On Android-based devices, audio/video capturing can be accomplished in two different ways. Either using the *MediaRecorder* API, which allows for media recording to a container format such as MP4 in a simple and straightforward approach by specifying capturing sources and the output format. Alternatively, one may use the *MediaCodec* API, which was introduced within API level 16 (Android 4.1 Jelly Bean) and offers low-level access to media codecs. In general, *MediaCodec* objects are associated with a number of input and output buffers, which are used to feed raw media data to the codec and access the encoded data respectively. Therefore, the raw media data must be captured separately and also the multiplexing of the encoded audio and video streams must be taken care of, unlike when using *MediaRecorder*.

There are good reasons for choosing the *MediaCodec* over *MediaRecorder*. In particular, using the *MediaCodec* enables the possibility to perform processing on the captured raw data before the actual encoding. For example, this is necessary for instantaneous quality feedback to users, which record a video and producing too much motion by moving the device too fast or holding it unstable. More information about quality analysis will be described in Section 6.1.

5.1.1 Main Components of the Implementation:

Figure 3 shows a graphical overview of the main components and their interaction which are introduced as follows:

- Audio Recorder Thread

  Records raw audio data from the microphone of the device.
- Audio Encoder Thread

  Encodes raw audio data provided by the Audio Recorder Thread and stores it into a buffer that is later accessed by the multiplexer.
- Frame to Surface Renderer

  Renders an image frame from the camera onto a surface that can be used as input for the video encoder and camera preview.
- Video Encoder Thread

  Encodes raw video data drawn into the input surface of the codec and stores it into a buffer that is later accessed by the multiplexer.
- Circular Byte Buffer

  Stores the encoded data coming from both encoders and notifies consumers of the buffer, if enough data is available and ready to be processed further (i.e., muxed).
- Multiplexer Thread

  Multiplexes elementary streams from the Circular Byte Buffer to container formats.

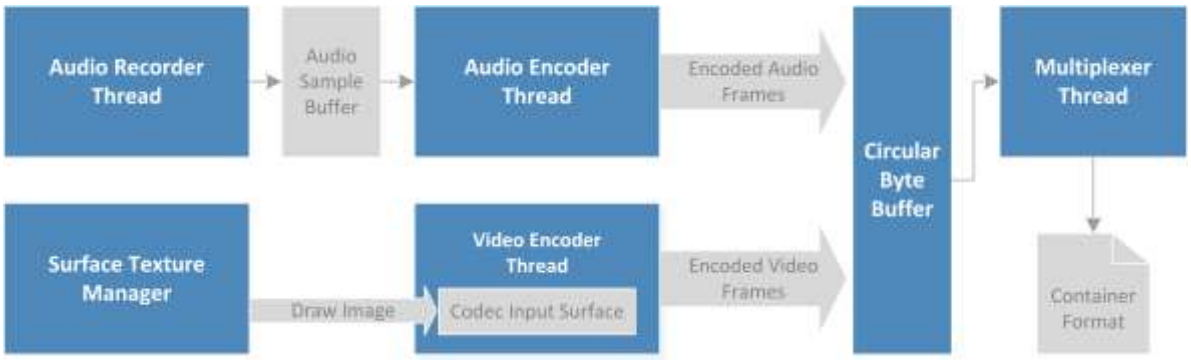


Figure 3 Overview Diagram of main components and their Interaction.

5.1.2 Communication Mechanism between Threads

We split up the processing bits into separate tasks, which are executed by individual threads concurrently. As the separate threads should be cooperating, we need to communicate between them, which is done via message passing.

5.2 Sensor Capture

The capture application enables real-time quality analysis including capture of the available sensors of the mobile device to support quality analysis. For example, the accelerometer readings are used to detect fast and shaky movements of the mobile device and, thus, unstable image sequences. Detailed sensor information of the following sensors is captured: location, accelerometer, gyroscope, magnetic field, orientation, rotation vector, ambient light, proximity and pressure.

Sensor	**ID**	**Parameters [units]**
Location	0	Latitude [deg], longitude [deg],altitude [m], accuracy [m]
Accelerometer	1	Acceleration x [m s^{-2}],acceleration y [m s^{-2}],acceleration z [m s^{-2}]
Gyroscope	2	Rotation x [rad s_{-1}], rotation y [rad s^{-1}], rotation z [rad s^{-1}]
Magnetic Field	3	Geomagnetic Field x [µT], geomagnetic Field y [µT], geomagnetic Field z [µT]
Orientation[1]	4	Azimuth [deg], pitch [deg], roll [deg]
Rotation vector	5	Rotation vector x [deg], rotation vector y [deg], rotation vector z [deg], scalar component [0]
Ambient Light	6	Ambient light level [lx]
Proximity	7	Object distance [cm]
Pressure	8	Air pressure [hPa or mbar]

Table 1 Summary of captured sensor data.

Table 1 provides an overview of the data captured from sensors of mobile devices. The sensor metadata is generated using the Android sensor framework[2] and currently accumulated in a local CSV file, with records indexed by time and the type of sensor (see Table 2 **Extract of a sensor data file.**)

[1] using magnetic field and accelerometer

[2] http://developer.android.com/guide/topics/sensors/sensors_overview.html

Timestamp	ID	SensorEvent.values[0]	SensorEvent.value[1]	SensorEvent.values[2]
1413744540606	1	9.69532	0.40761283	1.2018893
1413744540609	2	0.23116238	0.19281286	0.0077231675
1413744540611	3	-33.0	-6.4	10.3
1413744540613	4	109.58795	2.389158	82.93332
1413744540616	5	0.025789792	0.24839695	0.8855456
1413744540619	6	0.0		
1413744540622	7	8.0		

Table 2 Extract of a sensor data file.

6 Content Analysis and User Feedback

In order to ensure the quality of the output, incoming user generated content streams must also be checked in order to discard content that does not reach the desired quality levels. In order to encourage users to optimize the quality of their content, they shall receive immediate feedback on the quality of their recordings, rather than frustrating them by discarding their contribution later. Also, bandwidth and processing resources are scarce in such a live scenario, so that content with severe quality problems should not be uploaded or streamed to the server. Thus, the capture app performs quality analysis by using both sensor metadata (e.g., to detect shaky recordings from accelerometer data) and analysis of the content (e.g., detect out of focus issues), which are further detailed in the following subsections.

6.1 Quality Detection based on Basic Image Processing

Algorithms for sharpness, noise and over-/underexposure detection have been implemented in the app. The implementation of the quality analysis algorithms uses basic image processing functionalities of the *OpenCV*[3] library for Android. Therefore, the Android app *OpenCV Manager* has to be first installed on the mobile device to provide the *OpenCV* version that fits best for the current hardware. The sharpness algorithm computes slopes of edges by applying the Laplace operator. By subsampling those slope responses to a lower block level (e.g., 32x32 pixels) and by selecting 5-25% of the blocks with the strongest slopes, a reliable global sharpness value is obtained. The noise estimation algorithm is built on block scores from sharpness estimation. For noise level estimation a median filter is applied to the most homogeneous blocks and the average absolute differences between the original and the filtered image are computed for each block. Both, the sharpness and noise detection algorithms, work by considering blocks of a Laplacian filtered input image. The partial similarity of the two quality measures was utilized for the implementation and, thus, the computational effort can be optimized if both quality measures are calculated. If the average luminance exceeds predefined thresholds, the average brightness progression of images within a certain time frame is approximated where positive and negative brightness correction values are summed up individually. If one of these sums exceeds a certain threshold, the algorithm returns either an overexposure or underexposure event, respectively. A detailed description and evaluation of each quality analysis algorithm can be found in Deliverable D4.1. The runtime for analyzing one frame of a HD image sequence is about 200ms (Samsung Galaxy S5). Due to gradual temporal changes of sharpness/noise properties it is sufficient to process the visual quality analysis algorithms for every sixth frame.

6.2 User Feedback and User Interface

As illustrated in Figure 4, the proposed application provides a live preview to the user and using the “Start/Stop recording” button a video stream can be recorded. The application continuously measures sharpness, noise, and exposure. It detects the use of brightness compensation in real-time. The results of the quality analysis and the accelerometer sensor data are displayed on the left hand side of the preview. If a calculated value of one algorithm exceeds a predefined threshold, the corresponding text is highlighted in red as immediate feedback to the user about visual quality problems. Therefore the users is informed about which metric is affected (that one highlighted in red) and can react specifically.

The proposed application has a configuration functionality (see Figure 5) for (de)activating the analysis algorithms and sensors, and setting thresholds and parameters. For each quality analysis algorithm, for sharpness, noise and over-/underexposure detection, a predefined threshold can be adjusted. For example, if the estimated sharpness value falls below a predefined threshold, the image is considered as blurry and the user is immediately informed about the visual quality problems.

[3] http://opencv.org/

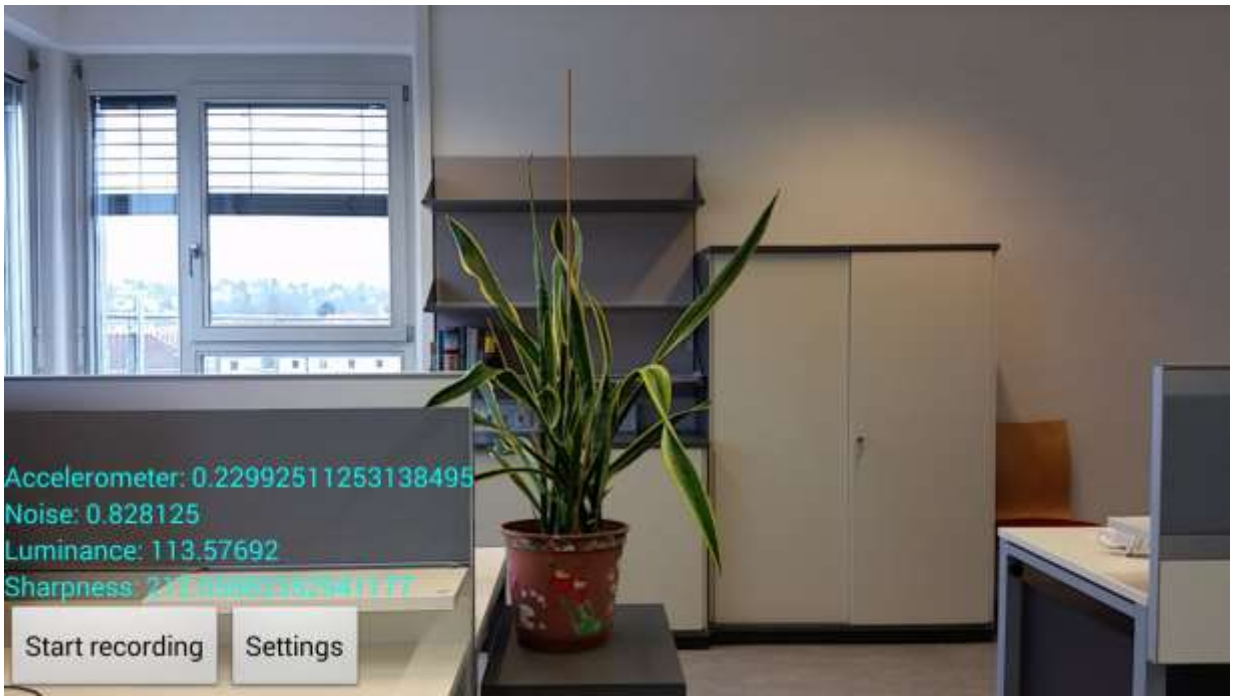


Figure 4 Screenshot of the Android capture application.

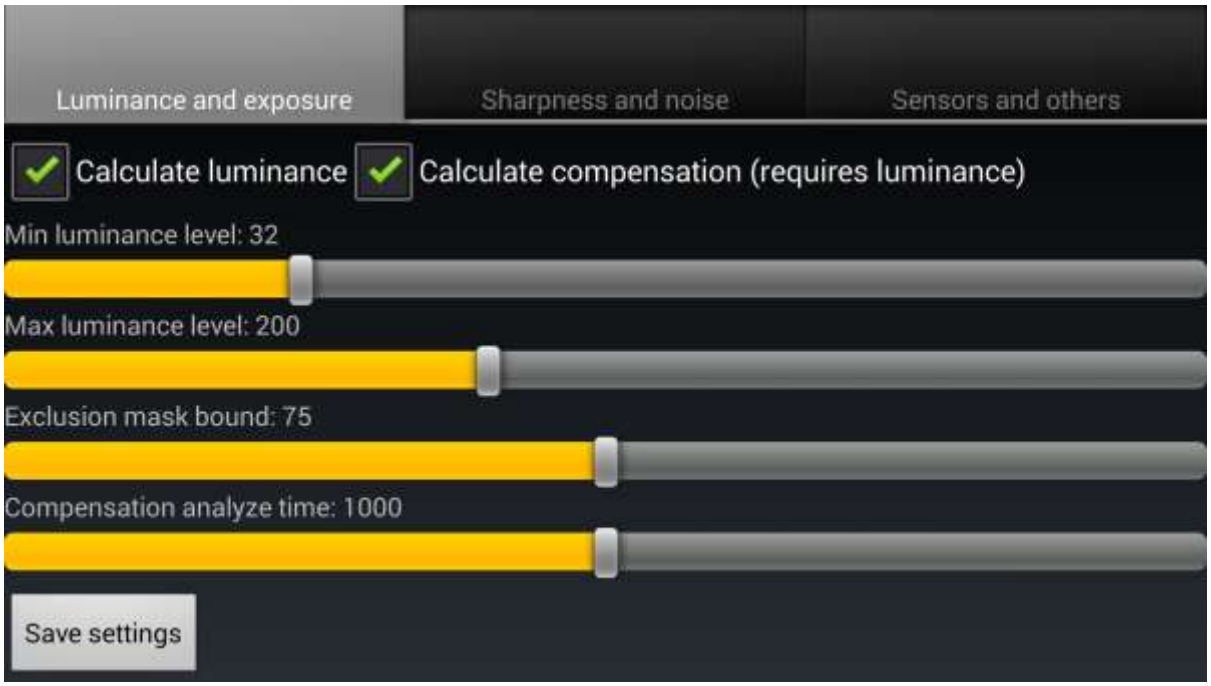


Figure 5 Settings of the capture application.

7 Upload and Streaming

Finally, the app has to provide means to upload captured clips together with the metadata immediately after capturing has finished or to stream the captured content and metadata in live scenarios. The latter case introduces difficulties like bottlenecks due to probably congested or low capacity (access) networks, which makes real-time upload of high-quality content difficult or even impossible. Furthermore, a trade-off between the number of simultaneously uploading users, connected to the same base station (in cellular networks) or access point (in WiFi networks), and streamed video quality, has to be made.

7.1 Session based Streaming and FTP Upload

In our first approaches, we evaluated session based streaming technologies, like RTSP (Real-Time Streaming Protocol) and RTMP (Real-Time Messaging Protocol) for live scenarios, as well as FTP (File Transfer Protocol) upload for non-live use cases, where uploading of captured essence after the recorded is finished is sufficient.

7.1.1 RTSP using libstreaming[4]

The *libstreaming* library is an open-source library and allows for streaming from the camera and the microphone on Android-based devices. To implement a simple stream using the library, only a few lines of code and almost no knowledge regarding audio/video encoding or the offered protocols is needed.

There are basically two common ways to use the library: (a) having the device acting as a streaming client by capturing audio and video and streaming it to a remote server; (b) letting the device itself be a server publishing the audio and video content that it captures.

In the latter case one can simply use a conventional RTSP receiver, like the VLC media player[5] for playout. Figure 6 shows a screenshot from VLC media player, playing a live RTSP stream, recoded by an Android-based device.

Figure 6 RTSP Live Playout of capture essence from a Prosumer device, on VLC Media Player.

The documentation claims that the library supports RTSP and RTMP protocols along with the video codecs H.263, MPEG-AVC/H.246 as well as the audio codecs AAC and AMR. Regarding RTMP, this was not found to be true, at least not in the version that was current at the time of writing this deliverable. Codec support and supported resolutions seem to be very device dependent as well.

[4] https://github.com/fyhertz/libstreaming

[5] http://www.videolan.org/vlc/

Using our implementation, we made the following observations while running the application on different devices:

- RTSP Server using H.263 video codec works well with a resolution of 1280x720 when streaming via VLC media player or ffmpeg. Even a resolution of 1920x1080 is possible; however the stream does not look very smooth anymore, due to a reduction of the frame rate.
- RTSP server using MPEG-AVC/H.264 video codec could not be made to work for any resolution.
- RTSP Client streaming on a Wowza Media Server worked using H.263 with a resolution of 1280x720, but not for 1920x1080. When using MPEG-AVC/H.264 we found that it only worked with a resolution of 352x288.

It seems like that the support for codec and resolution, using *libstreaming* highly depends on hardware specifications. Furthermore, since the outdated H.263 video codec seems to be the only one working properly, we came to the conclusion to not further develop this approach.

7.1.2 RTMP

For our test with RTMP, we used *android-rtmp-client*[6], which is for the most part based on the well-known open source media server *Red5*[7]. The library makes it possible to stream a video file from the local file system, after setting parameters, like the address and port of the remote server, the stream path, the file path. In general, for any file to be pushed to the server a *FileProvider* object is created that is associated with that file. This class provides functionality to repeatedly create messages, which can be pushed via RTMP until the end of the file. In a live streaming scenario, small video files are continuously recorded, stored, and pushed to a server. In this case, we repeatedly create a *FileProvider* object for the latest file and push it. The content of the first file of the stream is received correctly at the server, but all following data seems to be corrupted.

We change the implementation so that the timestamps of the files are added when changing the file to be pushed. This resulted in the video being okay except for the transition between two files, where the image freezes for about a second, and the audio being completely unusable.

Because of the fact of the library being based on *Red5*, which has a rather large codebase, and having not the capacity to overcome the described problems in a reasonable amount of time, we decided to choose a different approach, as described in Section 7.2.

7.1.3 FTP Upload

For non-live use-cases, the proposed application allows uploading the captured video with the associated sensor data to an FTP server after recoding is finished. Therefore, an FTP server connection for uploading captured video and sensor data files can be configured based on either the provided configuration functionality or an upload dialog that appears immediately after capturing has finished. As a conventional file upload, like FTP, is not applicable for live-scenarios this approach will eventually be used as backup mechanism for scenarios where the network connection was not accessible during the event.

7.2 Streaming/Uploading using HTTP

In contrast to conventional, session based, streaming, we developed a scalable upload approach, which runs on top of the well-known HTTP protocol.

After the user starts recoding, the captured essence is encoded, split and packaged into media containers, like we describe in Section 5.1. The container-files, containing media of a few seconds length, are stored on the local storage of the device. The device will maintain a storage of segments, based on its memory capabilities, where older segments will be removed first.

The device runs a webserver, where each segment is accessible via a unique URL, for a certain amount of time. A client, which can be another user or are processing unit on site, will be able to fetch the segments via HTTP standard compliant GET requests, as shown in Figure 7.

[6] https://code.google.com/p/android-rtmp-client

[7] https://github.com/Red5/red5-server

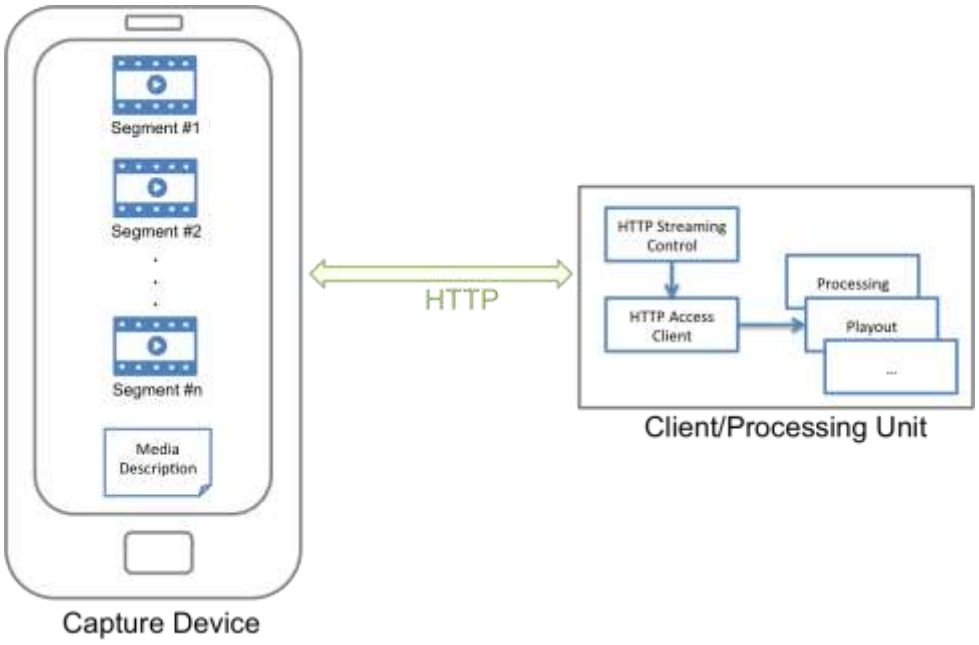


Figure 7 Streaming/Uploading Captured Essence via HTTP Overview.

In order to describe the temporal and structural relationships between segments, the application generates an XML file (Media Description), based on the formal description of MPEG-DASH Manifests[8], that represents the individual segments with HTTP URLs. This structure provides the binding of the segments to parameters, like the duration of segments or the time they will be available on the device. As a consequence each client will first request this description of the media content. Based on this information, it will request the individual segments.

The described concept shows several similarities to the MPEG-DASH (Dynamic Adaptive Streaming over HTTP) standard, like the content is stored in segments, the usage of HTTP, and the media description. Due to these commonalities, we call this approach R-DASH (Reverse-Dynamic Adaptive Streaming over HTTP).

Because of its scalability and its possibility for continuous adaptation to the current upload channel conditions, as we describe in Section 9.1, we prefer this approach over conventional session based streaming protocols and will develop R-DASH further.

7.3 Metadata Handling

The captured sensor metadata is currently accumulated in a local CSV file, with records indexed by time and the type of sensor. The file can be sent via email or is uploaded to a FTP server. Using the approach described in Section 7.2, incremental metadata files will be stored locally and are made available as segments of the metadata stream. The segments will be available in JSON format. It is envisaged to use a DASH-like mechanism for metadata upload. The details of this method are currently being developed.

[8] http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=65274

8 Marconi Moments Capture App

8.1 Capture App

During a test shoot in October, named the *Marconi Moments*, the audience was able to contribute UGC during the concerts using a simple web application. We set up a private WiFi network (without Internet access due to bandwidth issues) with which the audience could connect using their smartphone or tablet. We hosted a small Web application 'Moments', which was accessible via http://marconimoments.be. There was only one button: 'Select Moment'. When a user pressed this button, the (native) camera app would pop up and s/he could start recording.

Figure 8 Marconi Moments Capture App

When the user stopped the recording, the video file was automatically uploaded, via FTP to a backend system, where it was placed in a queue. The backend then converted this UGC to a Web-supported format (a low-resolution WEBM proxy without audio in this case) so it could be used in another Web application, 'The Wall'. The Wall is a light version of 'The Wall of Moments' (described in Section 8.2), which displayed submitted Moments on a screen next to the stage.

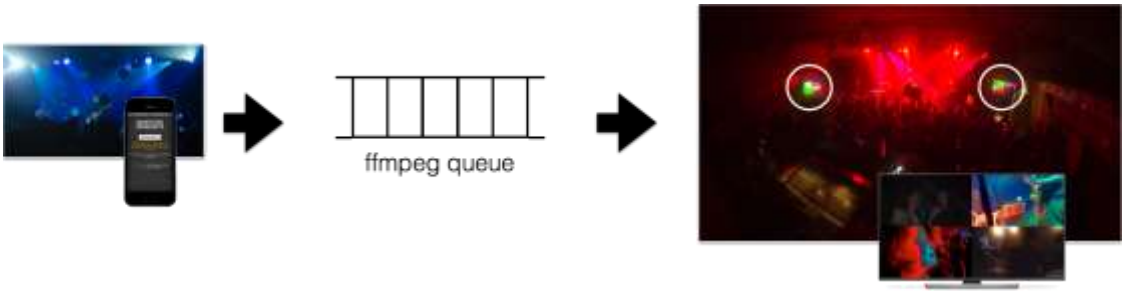


Figure 9 Upload to The Wall

8.2 The Wall of Moments

The Wall of Moments (shown in Figure 10) is a Web application, optimized for tablet devices. It delivers a unique festival experience for remote users by using 'Moments' captured by peers attending the festival. These moments are arranged in a moving mosaic representation (The Wall) on the remote users' device (Moments are arranged in a scrollable matrix, with dimensions depending on the device). The selection of shown Moments is based on popularity, friends, music preference, and custom filters.

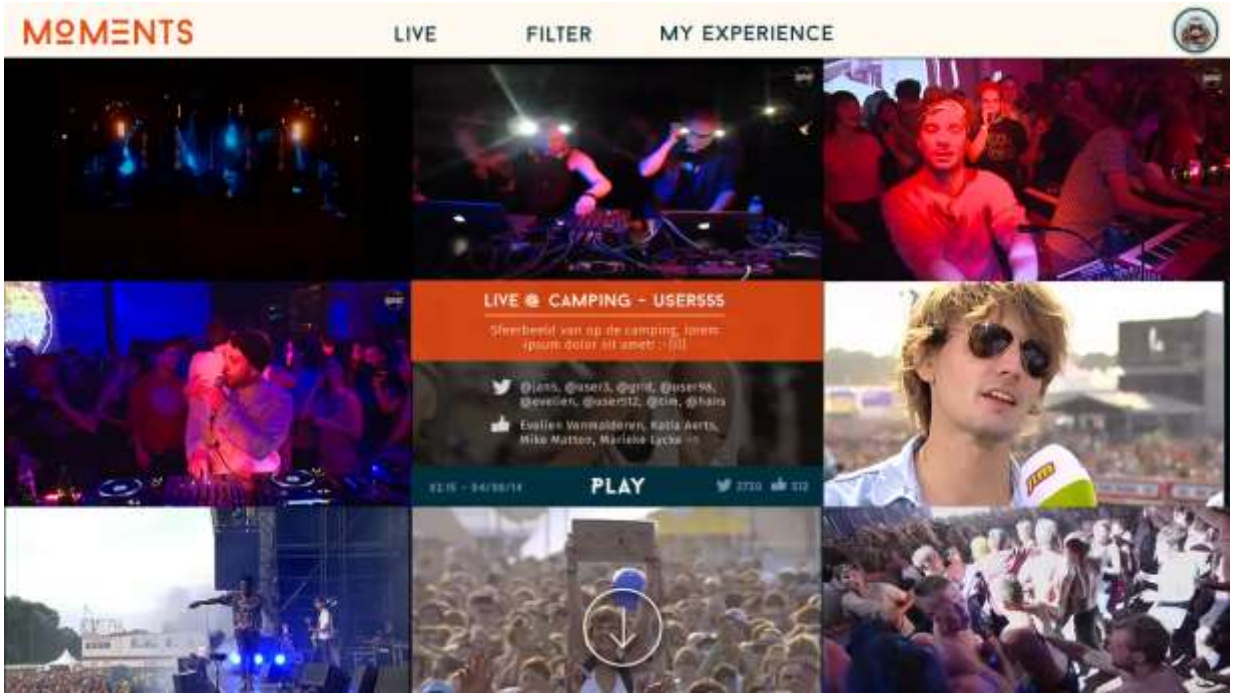


Figure 10 The Wall of Moments Screenshot

The aim is to create a social, interactive portal to the festival for users at home and on the way during the festival; and a best of after-movie experience thereafter.

8.3 Marconi Moments

Because the Marconi Moments were organized in a controlled environment, only a small group of people attended the Marconi Moments (an average of about 50 people per night, we had a concert on Saturday and Sunday). Nevertheless, the results were promising. We received a lot of videos (736 in total, accounts for about 4 hours of raw video content), and the audience was enthusiastic. Afterwards, we sent out a small survey to allow people to share their thoughts on the test shoot. Some quick insights on the survey results are presented here:

- 73% of the attendees used the Moments app to upload videos to our system.
- 51% of the attendees were of the opinion that the presence of the two screens (displaying the Wall) was an added value to the festival experience.
- 66% of the attendees consider using such an application on a real festival.
- 45% of the people who filmed would share their videos on social media as well.
- 72% of the people who filmed would share their videos with the festival production for use elsewhere.
- On the second night, 89% of the concert was covered by videos uploaded by the audience.

8.4 Dranouter Moments Capture App

People attending the Dranouter festival[9] will be able to install an updated, native version of the Moments capture app. With this app, people can record and upload video while getting instant (quality) feedback, such as:

- The video is too shaky, please steady your device (using accelerometer).
- The video is too light/dark (using algorithms).
- This video is not recorded on the event site (using GPS).
- This video is not recorded during the event (using timestamps).

Based on this quality assessment, users will be asked to change their camera position. If there is no improvement after a certain amount of time, uploading to the ICoSOLE core system will be halted, as the captured video will be useless anyway.

If the upload speed is too slow or if the connection is lost, uploading will automatically resume on a later time. On the field test, we will provide a (private) WiFi network for fast upload performance, but users will be able to use their 3G/4G connection as well.

The app itself will be as simple as possible to provide users at the festival with a seamless experience, which does not interrupt the festival experience itself. Screenshots of the recoding-views can be seen in Figure 11, and upload views in Figure 12. Users will only be able to use videos captured with the app and will not be allowed to select videos from their camera roll to make sure only festival content can be sent in. There will be one main button: 'Record Moment', which starts a new recording.

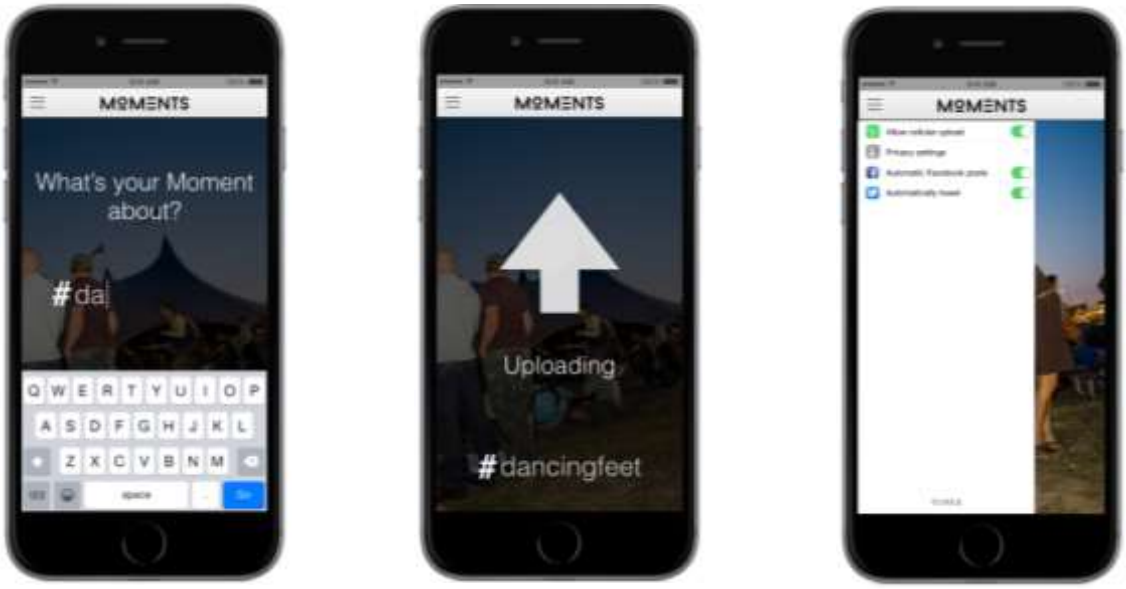
Main screen: one record button | During recording | Real-time quality assessment

Figure 11 Dranouter Capture App Recoding Views

When a recording is finished and if it meets the (quality) requirements, a user can annotate it quickly by typing what is it about in a couple of words. Afterwards, it will automatically be uploaded to the ICoSOLE backend, along with metadata such as GPS coordinates, timestamps, orientation and so on. If the user enabled social integration, it will also be posted to her/his Facebook wall and Twitter feed.

Later on, the Moment will be used on the Wall. Depending on the user's privacy settings, it may or may not be used on the public Wall installed at the event site, the public wall online, or his/her friends Wall.

[9] http://www.festivaldranouter.be/



Quickly annotate your Moment Automatic uploading Privacy settings

Figure 12 Dranouter Capture App Upload Views

9 Outlook and Planned Activities

In the next months the focus will be given to the integration of the described parts for user tests at live events during summer time. Based on the findings, we will improve existing components or implement new ones if necessary. In the context of planned activities, we highlight three functionalities we want to implement next in line.

9.1 Efficient Bandwidth Usage for Content Upload/Streaming

We will address possible bandwidth issues, for uploading captured content on site, due to congested networks. A trade-off between the quality of the captured essence and upload-delay (especially important for live scenarios) has to be made.

Our approach will include on-device transcoding depending on the current upload channel capacity. In times of high network usage the device will produce a lower quality representation, in terms of bitrate and resolution. Therefore the media segments will be smaller in size and real-time streaming will be ensured to some extent. In addition to that lower quality representation, a second representation containing the source quality will be generated and stored – depending on the device capabilities.

The stored high quality representation will be available for later upload/streaming, once the channel capacity is sufficient.

9.2 Time Synchronization

As clocks of mobile devices are not precise, a synchronization mechanism will be implemented. GPS time will be used as a reference, as it is assumed to be available even if the GPS signal is not good enough to determine the position or if it has not been updated recently. The offset of the system time to the GPS time can also be determined easily at the server side. In case GPS is not available at the client device, the device can be synchronized with the server via timestamp round-trip packages set between the server and the app.

9.3 Reverse Dynamic Adaptive Over HTTP (R-DASH)

The method described in Section 7.2 presents a first step towards R-DASH.

The application will be extended to generate a valid DASH manifest file describing the temporal and structural relationships between the generated segments. In combination with the media segments, DASH player implementations, like bitdash[10] or dash.js[11] will be able to playback a stream directly from the user device itself. Processing devices on site will be able to fetch the generated content in parallel.

[10] http://www.dash-player.com/

[11] https://github.com/Dash-Industry-Forum/dash.js/wiki

10 Conclusions

This document presents concepts and first implementations of user generated content tools, related to audio and video capture. To meet the requirements of the project, and Android based application was implemented, where focus is given on ensuring quality and on-device analysis of captured essence. Furthermore, insertion of user metadata and communication with production management, as well as essence and metadata transport to a central storage and processing unit, is addressed.

We divided the structure of the application in three different parts: (a) capture, (b) processing and (c) streaming, as described in Section 4.

For recording of audiovisual content, we make us of Android's *MediaCodec* API, due to its flexibility. As stated in Section 5.1.1, the recorders for audio and video inject the captured essence to encoder threads, which generate encoded audio and video frames respectively and forward them to a multiplexer instance, where the elementary streams are written into a container format. Also metadata, on basis of available sensors of the device are recoded and currently accumulated in a local CSV file, with records indexed by time and the type of sensor.

The proposed application continuously performs quality analysis of the captured essence on the device and beside the live preview also a textual output is displayed, to inform the user about the potential visual quality issues, as immediate feedback. As described in Section 6, algorithms for sharpness, noise and over-/underexposure detection have been implemented. The device's accelerometer is used to detect fast and shaky movements of the mobile device and, thus, unstable image sequences. If the user is not able to meet the desired quality restrictions, the content gets discarded.

Based on our evaluation of different uploading/streaming approaches, we conclude to use a novel upload approach, running on top of the well-known HTTP protocol, as discussed in Section 7. As this session free and scalable method of uploading media (as well as metadata) as small chunks of few seconds' length shows great similarity to MPEG-DASH, we call it reverse-DASH. In combination with on-device transcoding, as described in Section 9.1, adaptations to the currently available bandwidth, and therefore to varying channel conditions, will be possible.

Our next steps will be towards the integration of the different parts developed by partners for user tests during summer time, as well as time synchronization, using GPS, as described in Section 9.2. We will also improve the application based on the findings of our evaluations and user feedback of the first test shoot.

11 Glossary

Terms used within the ICoSOLE project, sorted alphabetically.

Partner Acronyms

BBC	British Broadcasting Corporation, UK
BIT	bitmovin GmbH, AT
DTO	Deutsche Thomson OHG (Technicolor), DE
iMinds	iMinds Vzw, BE
JRS	JOANNEUM RESEARCH Forschungsgesellschaft mbH, AT
TaW	Tools at Work Hard+Soft Vertriebsgmbh, AT
VRT	De Vlaamse Radio en Televisieomroeporganisatie NV, BE

Abbreviations

API	Application Programming Interface,
CSV	Comma-separated values
DASH	Dynamic Adaptive Streaming over HTTP
DoW	Description of Work
FTP	File Transfer Protocol
GPS	Global Positioning System
HD	High Definition
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MPEG	Moving Picture Experts Group
R-DASH	Reverse-Dynamic Adaptive Streaming over HTTP
RTMP	Real-Time Messaging Protocol
RTSP	Real-Time Streaming Protocol
UGC	User Generated Content
URL	Uniform Resource Locators
XML	Extensible Markup Language

Acknowledgement: The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 610370.

