Design Description
LiveTV for Mobile Applications

Version 1.1

Revision History

Date	Version	Description	Author
2009-10-05	0.1	Initial Draft	Darko Ronić
2009-10-07	0.7	Architecture description	Darko Ronić
2009-10-08	0.9	Error handling	Darko Ronić
2009-11-07	1.0	Classes and some fixes	Darko Ronić
2009-11-22	1.1	Additional fixes	Darko Ronić

Table of Contents

1. Introduction....4
- 1.1 Purpose of this document....4
- 1.2 Intended Audience....4
- 1.3 Scope....4
- 1.4 Definitions and acronyms....4
- 1.5 References....4

2. External interfaces....4

3. Software architecture....4
- 3.1 Conceptual design....4
- 3.2 System specification....4
- 3.3 Error handling....5

4. Detailed software design....5

5. Approvals....5

1. Introduction

1.1 Purpose of this document

The purpose of this document is to present the project design to all parties involved in the project. Among other things, system architecture will be described, as will the interfaces to the the system itself. As the project depends heavily on video and audio codecs, they will be described as well as will the network communcation used between parts of the system.

1.2 Intended Audience

This document is intended for flowing parties:

- project team; to use it as guidance during the project
- project supervisor; to see if the project is on the right path
- customers; to see if they are satisfied with the system and its functionality

1.3 Scope

This document will not deal with implementations of every part of the system. It will more be focused on a higher level of implementation, describing what functionalities will each part have and how will they be implemented. Network and codecs used will be described in more detail than the rest of the system as will the user interfaces of each part of the system.

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
RTSP	Real Time Streaming Protocol
FER	Faculty of electrical engineering and computing, Zagreb
MdH	Mälardalen University, Vasteras, Sweden
SVN	Subversion revision control software
API	Application Programming Interface
S60	Symbian S60 operating system
VLC	VideoLAN server used for streaming
DSS	Darwin Streaming Server

1.5 References

2. External interfaces

The system itself consists of three separate interfaces. Two of those three interfaces are private and one is public.
The public interface is the one on the client mobile phone. This interface will be developed in Java so that it can be played on the majority of mobile phones. It will provide the client with interface to watch the video stream of an event.
The first private interface is the studio application. It is a standalone Windows application and as such will provide a graphical user interface (done in C#) to a person to create a event stream from several camera streams with added support for commercials.
The second private interface is the one at the recorder mobile phones. Mobile phones used for recording of an event are Symbian S60 5th edition based Nokia phones (for example Nokia XM5800). Therefore, an user interface will be developed in Symbian C++.
The purpose of the private/public division is because some of the interfaces won't be accessible by public individuals but only by trusted employees. A director will be responsible for creation of event stream, while several “cameramen” will record an event.

3. Software architecture

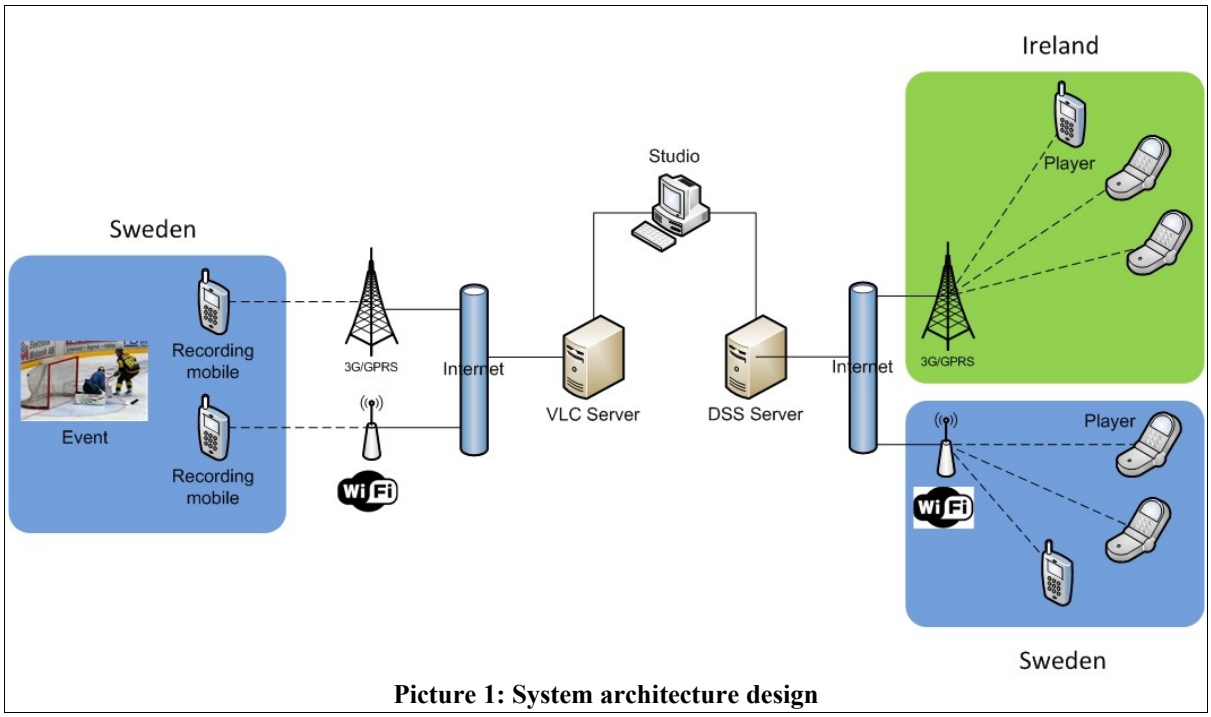
3.1 Conceptual design

The whole system is divided into three modules.
First is the recorder modules which is used to record an event via mobile phones, encode video and audio in h.263 and AAC audio with and 3GP container and send it via RTP protocol to the server module.
The server module is in fact made of VLC server integrated in the studio application via libVLC for C#, the stream selection application itself and a Linux server running DSS. The video is streamed to the client using RTSP protocol.
The client module is an Java application for mobile phones which receives a stream via RTSP protocol and plays it. The player also provides the feedback functionality. User are given a choice to use an video application of choice as long as it supports RTSP streaming but the feedback functionality is then lost.

3.2 System specification

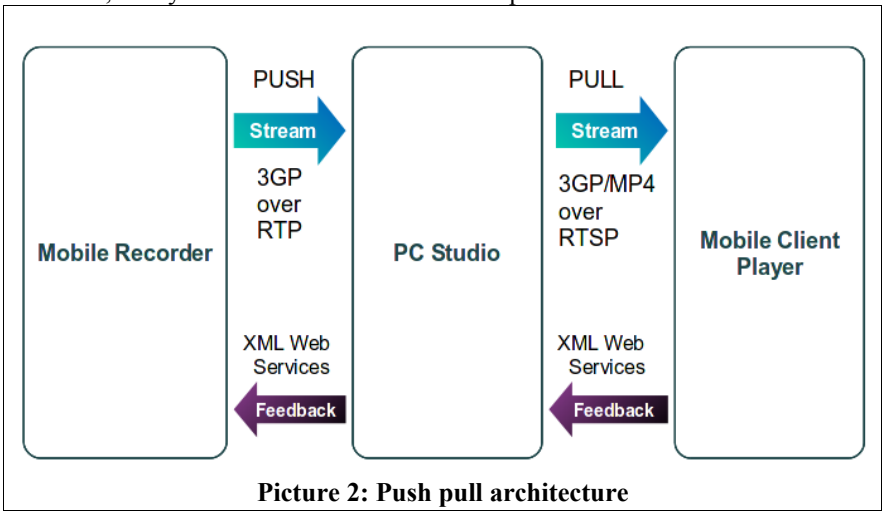
On the recorder module, Symbian C++ will be used to develop an application for recording. This will be used together with a number of Symbian APIs that are used to access various functions of the mobile phone, such as the camera, the microphone or the WLAN/3G interface. There are number of APIs that can be used to record video and audio, each with it's own advantages so which API is used will be decided during development.
Server module will be developed in C# using DirectDraw to display the video streams. VLC implementation for C# will be used to receive streams from the recorders using RTP protocol. A total of six streams will be supported together with a local stream with commercials. The combined stream will than be transferred to client via DSS on a separate server. DSS is used because it supports load balancing which will be required because the scalability should be good (large number of viewers).

Player module will be developed in Java using MMAPI framework. This framework is used for various multimedia features such as video and audio playback or integrated video and audio streaming. A different protocol shall be used for feedback. This protocol will send the feedback directly to the studio application without the need for additional server.



Picture 1: System architecture design

Design of system architecture from a higher level can be seen on Picture 1. As noted already, the system is divided into three modules. Each of this modules will be described individually in this chapter. The purpose of the picture above is to show how the system will work from a higher level without specifying network communication, protocols or codecs. Even this picture shows three modules that are separated from each other.
On a slightly lower level, the system can be shown like on the picture below.



Picture 2: Push pull architecture

From this level, the system can be described with a simple PUSH-PULL architecture. The Mobile Recorder module pushes the recorded stream in 3GP via RTP protocol to the PC Studio. Picture 1 showed that the server consists of the PC Studio and VLC/DSS servers. The VLC server is included in the Project Studio. In that way, the Mobile Recorder pushes the video stream to the VLC server.
The Mobile Client Player pulls the video stream for the PC Studio. The PC Studio's purpose is to combine streams from multiple recorders and create a TV stream. The stream is relayed through a DSS server and is then pulled by the mobile client.

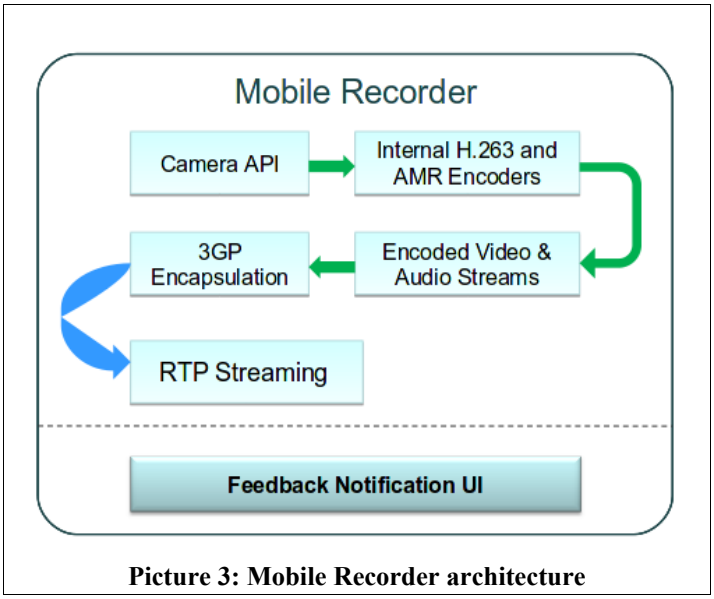
3.3 Error handling

Error	Action

Single recorder stream fails	Automatically switch to another stream and display message in the PC Studio.
All recorder streams fail	Display an error in the PC Studio and automatically start streaming the commercials.
Connection fails on the client	Display error message and add an option to reconnect to the stream.
DSS server crashes	Display error in the PC Studio informing the operator that the video isn't streamed correctly. On the player, same as above.
Connection fails on the recorder	Stop recording and streaming, return to menu and add option to restart streaming.
PC Studio crashes	Same as above for the recorder. Same as third from top for the client.

4. Detailed software design

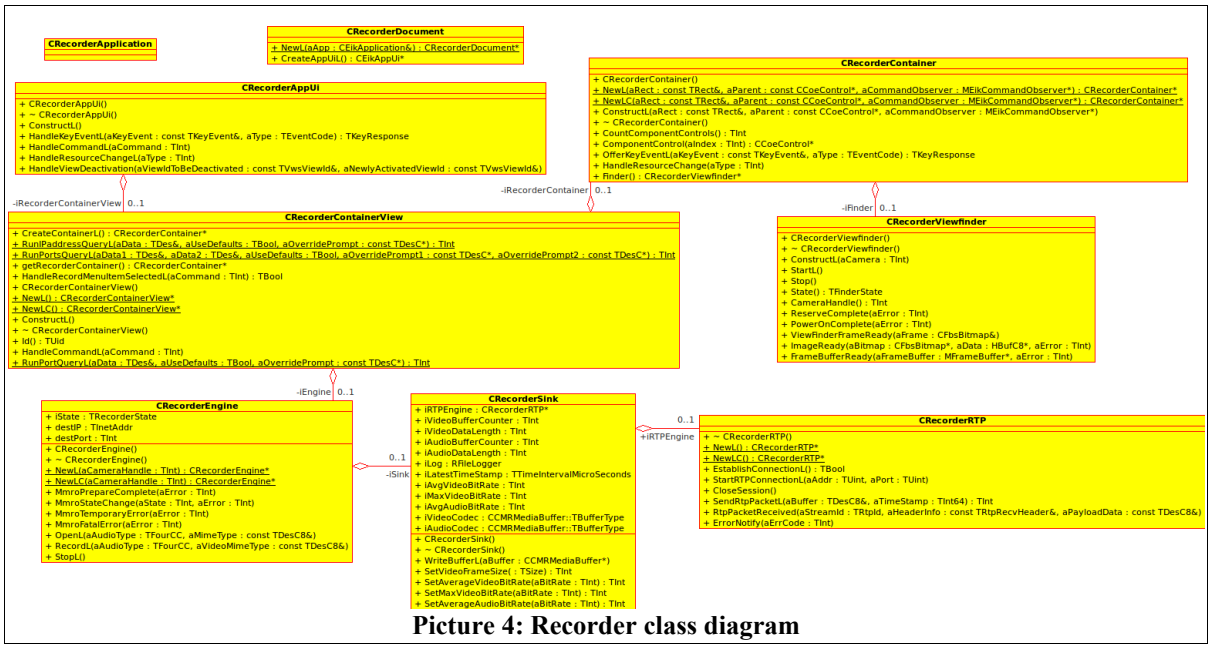
4.1 Mobile Recorder



Picture 3: Mobile Recorder architecture

Picture 3 shows the detailed architecture and process of the Mobile Recorder.
The video is recorded via Camera API provided by Symbian OS. This video stream is internally encoded in H.263 and AMR formats before it is stored in a memory buffer. Since Symbian doesn't provide 3GP API, manual 3GP encapsulation has to be done. 3GP is used just as an container which is then transferred via RTP protocol to the PC Studio application.

On the picture below we can see a class diagram for the Mobile Recorder application:



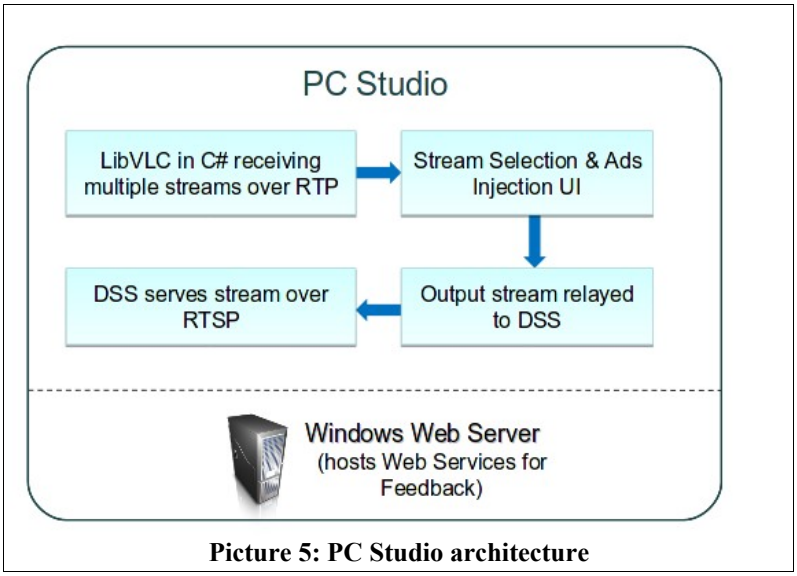
Picture 4: Recorder class diagram

The applications follows the standard template for Symbian applications in C++. This template requests that the application is started in a specific way. Classes *CRecorderApplication* and *CRecorderDocument* are part of this template and are generated automatically by the Carbide.c++ IDE for Symbian applications. These classes are not modified.

Next is the *CRecorderAppUI* class which is the basis of whole application. It creates new containers which are displayed (GUI) and captures keys that the user presses. The next two classes are tightly connected: *CRecorderContainerView* and *CRecorderContainer*. These two classes are visual containers that are shown to the user (GUI). When a key is pressed, *CRecorderContainerView* handles it and calls the appropriate methods. There is also another class, *CRecorderViewfinder* which displays the camera viewfinder on the container.

These classes are GUI based, others are used for recording and communication. The *CRecorderEngine* class is the basis for recording. It inherits the Media Recorder API observer class and is, therefore, used to start the recording process. The *CRecorderSink* class functions as a sink for recorded frames. The recording operation in Symbian is implemented as source-sink operation where the Media Recorder API is the source and *CRecorderSink* is the sink. The sink receives buffers containing audio and video in regular intervals and then uses the *CRecorderRTP* class to send it via RTP. The RTP class creates a connection on application startup while the RTP connection is started at the beginning of each recording cycle.

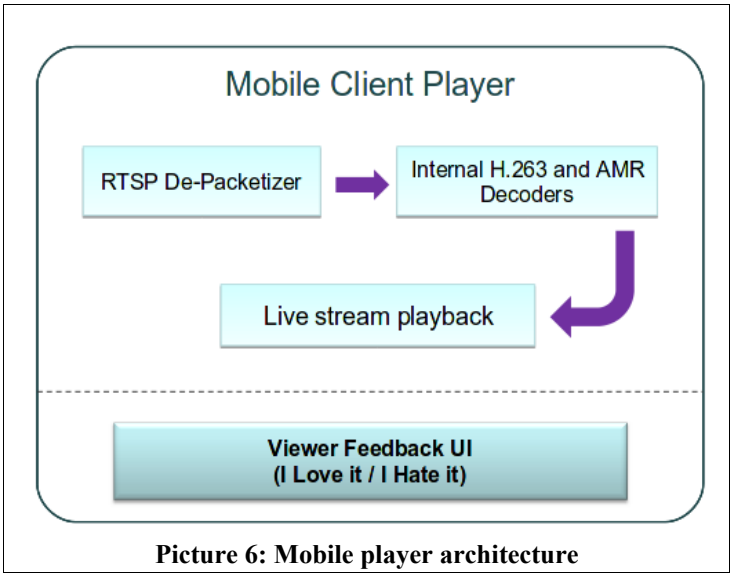
4.2 PC Studio



Picture 5: PC Studio architecture

The PC Studio application uses libVLC for C# to listen on a port for incoming video streams from mobile recorders via RTP protocol. It can support up to 6 parallel streams. These streams are then showed to the operator who chooses which stream will be forwarded to the client and maybe insert some commercials. The output stream is then relayed to the DSS server. The DSS is used because it supports load balancing and there will always be more clients than recorders so future development is considered. DSS server then serves the output streams via RTSP protocol to the viewers.

4.3 Mobile Client Player



Picture 6: Mobile player architecture

The Mobile Client Player is written in Java and MMAPI so it can support as many mobile phones as possible. For mobile phones that do not support Java or the MMAPI, a viewer can use some other video player suitable for his phone. It should only support RTSP streaming. Mobile client player uses internal Java API to stream from DSS server and decode the video and audio in the stream. The video is then played on the screen.

4.4 Client feedback

Among other functionalities, the system would be able to support feedback from the client to the server and back to the recorder. XML Web Services will be used to support feedback. Client will rate the stream in the player which will then send his feedback back to the PC Studio. Since this is an optional requirement, it isn't as important as the other functionalities.

5. Approvals

Name	Title	Date yyyy-mm-dd	**Signature**