WP9 - Validation on Platform

Deliverable D9mm.4 - Multimedia Application V2

WP9 - Validation on Platform: Deliverable D9mm.4 - Multimedia Application V2
by Cristina Sandoval

Table of Contents

List of Figures

Chapter 1. Introduction

1.1. Document Purpose

In this document we will describe the implementation of the Multimedia Application in its version 2 developed in the context of the OCERA project and the scenario in which we will test the enhancements that OCERA provides to the whole digital video system.

We refer to documents [OCERA D9mm1], [OCERA D9mm2] and [OCERA D9mm3] for information about requirements, architecture and methodology not present in this document.

After a brief description of the multimedia application and details of implementation, to make this document as self-contained as possible, we will present the "Quality of Service" requirements and some of the results of the OCERA project, particularly the OCERA scheduler, in its use on a popular consumer electronic device, a hand-held computer.

Chapter 2. Application Description

In this chapter, we will present a brief description of the application for document completeness. Further information on requirements and concrete architecture are described in [OCERA D9mm1], [OCERA D9mm2] and [OCERA D9mm3].

2.1. Overview

Visual Tools has developed a digital video viewer for XScale and StrongArm based handheld platforms that will be integrated with its range of Digital Video Recorders and Transmitters (DVRT) using Wireless Communications. We will call this application Visual Tools Wireless Viewer, VTWV from now on.

As already stated in previous documentation, this application is both, high resource consuming and running on a low performance platform (XScale and StrongARM based devices) and thus the OCERA Resource Reservation components are extensely used.

In what respects communications, the nature of the transmission media (radio) implies the necessity of controlling resources since, depending on particular conditions, the scenario is prone to interferences and the resource consumption can increase when transmitting the video stream.

OCERA Resource Reservation Scheduler running in the handheld device assures that it will be able to show the whole incoming video stream on the client side of the communication. On the server side, DVRTs, could also be difficult to guarantee a constant outcoming video stream as, on one hand, many applications are always running, i.e. grabbing, compressing, storing and transmitting video, and on the other hand there are some very high CPU consumming applications that start on specific events, for example when an alarm is fired. In this way OCERA components will play an important role on both sides of communication.

The VTWV is a lightweight X Window application thought to receive wirelessly and present video streams transmitted by one (or several) Visual Tools DVRT. We are able to find out which DVRTs are present in the LAN and request them to stream video from any of its cameras.

The VTWV was first designed and thought to work on a PDA Compaq iPAQ 3850, that is a StrongArm based device, as this family was present in most of the more powerful PDAs.

By the end of 2003, Intel put on the market the XScale ARM processor, it was designed to optimize low power consumption and high performance processing. From that moment a long list of companies have integrated any processor of this family on their latest mobile handheld devices. This state-of-the-art situation has made Visual Tools be replacing the old StrongArm based handhelds devices for XScale based new ones.

On the other hand, due to commercial reasons we have developed the VTWV in a programming language suitable for an easy port to PocketPC, though the reference development environment will, of course, be done on a Linux platform using the OCERA components. This porting will, eventually, lack the benefits of the OCERA components wich are developed in and for a Linux environment.

2.2. Quality of Service in Digital Video Surveillance Systems

We have applied Resource Reservation Scheduling to a digital video surveillance system. Before getting into details, let's briefly explain the main characteristics of these sort of systems.

Digital video for security markets differ from digital video created for entertainment and other professional uses in several aspects. First, quality and bitrate must be traded-off.

While bitrate should be kept low, for recording to save storage space and for transmission to save bandwidth, the video quality should be enough as to serve its purposes. Typically, video quality in surveillance systems is much lower than in broadcast systems, wich make the desired low bitrates possible.

Second, framerate should not be very high. On one hand, to save bandwidth and storage, and on the other hand, to save digitized images on the video acquisition side. Only a small portion of security market is interested in high-framerate videosurveillance systems, e.g., casinos.

Finally, digital video surveillance systems should provide constant framerate. Being already not very high, the user should be guaranteed with constant framerate that permits good analysis of the scene in case of alarms, fires, etc... It is not acceptable that high variation in framerates might occur in those situations.

2.3. Architecture

The development of VTWV will be done on a Compaq iPAQ 3850 with a Symbol 802.11b CompactFlash Wireless LAN card and an HP iPAQ h5550.

2.3.1. Linux Distribution

In the typical scenario, see Figure 2-1, we can distinguish, at least, two kinds of Linux devices:

- DVRTs: x86 based platforms in charge of grabbing, compressing, storing and transmitting video.
- PDAs: Handhelds devices that work as clients in the communication. In our implementation these devices are XScale or StrongArm based.

On each one of these platforms there will be a different configuration of Linux and OCERA components, that are related below.

Both handhelds will have the Linux Familiar distribution installed. See [OCERA D9mm1] for details.

The Familiar distribution will provide all external software required for the application. In particular, it will provide the different drivers needed for the CF Wireless LAN card, the iPAQ screen, etc... as well as specific software developed for this kind of platform, as the handwriting recognition software, orientation display manager, etc...

On Compaq iPAQ 3850, instead of using the off-the-shelf kernel in the Familiar distribution, we will use the OCERA kernel, which is a vanilla kernel plus several patches that enhance its soft real-time capabilities, check deliverable D2.1 for further information about the OCERA kernel.

But we cannot make use of the standard OCERA kernel. In order to run a Linux kernel in an iPAQ we need to apply some patches to it that provide the specific drivers for this platform. Some examples of the particularities these patches deal with are the touchscreen, battery, etc... Once we apply those patches, we can boot the iPAQ and run the OCERA kernel with soft real-time capabilities.

On HP iPAQ h5550 it was not possible to follow the above described steps. As this device is not supported by Linux kernel version 2.4.18, that is indeed OCERA kernel version, we have installed a Linux kernel 2.4.19 with extra patches to support the handheld device. To benefit from OCERA kernel, in particular from the OCERA scheduler, the OCERA generic scheduler patch has been ported to version 2.4.19.

On the DVRT, it is used a 2.4.20 version of Linux kernel with some additional patches for our architecture. Although it was not possible to make use of OCERA kernel, we were very inerested in, at least, having the OCERA scheduler available in the video server, so we have ported the OCERA generic scheduler patch to 2.4.20 kernel version.

2.3.2. Ewe

For the development of VTWV we will use the Ewe programming system (`http://www.ewesoft.com`).

Ewe allows to write Java programs for desktop computers as well as for mobile handhelds. It is particularly suitable for the development on handhelds platforms and will allow us not only to use it under the Linux kernel for the OCERA project but to port the application (though, unfortunately, making no use of the OCERA components) to the PocketPC OS, which is of particular importance from the commercial point of view.

VTWV have no hard real-time constraints, but in case we add, in future releases, some feature requiring hard real-time capabilities, we can still make use of Ewe since it provides an API for C extensions.

There are other options for Java development on mobile devices but we have chosen Ewe for its (rather) small footprint and its API completeness and its maturity. For a comparison between Ewe and PersonalJava see [Ewe vs PersonalJava].

2.3.3. Security

VTWV does not provide any security mechanism. We rely on the WLAN security mechanisms setup by the network administrator.

It is true that the current 802.11b wireless LANs are at risk of compromise. Since it is not easy to provide robust security mechanisms on the wireless LAN, to make the transmission as secure as possible, we will just encourage the use of WEP and, eventually, other proprietary security mechanisms depending on the particular vendor that provides the wireless access points and stations. See [Arbaugh et al. 2001], [Walker2000] and [Borisov et al.] for details.

2.4. Application scenario

The aim of this demonstrator is to get the OCERA components running on all the Linux devices present on the scenario, to ensure a minimum quality of service on video streaming and displaying and to test the OCERA resource reservation components on different platforms.

We will run the application within a network configured according to what we called *Basic Topology*, see Figure 2-1. For details about this topology check [OCERA D9mm1].

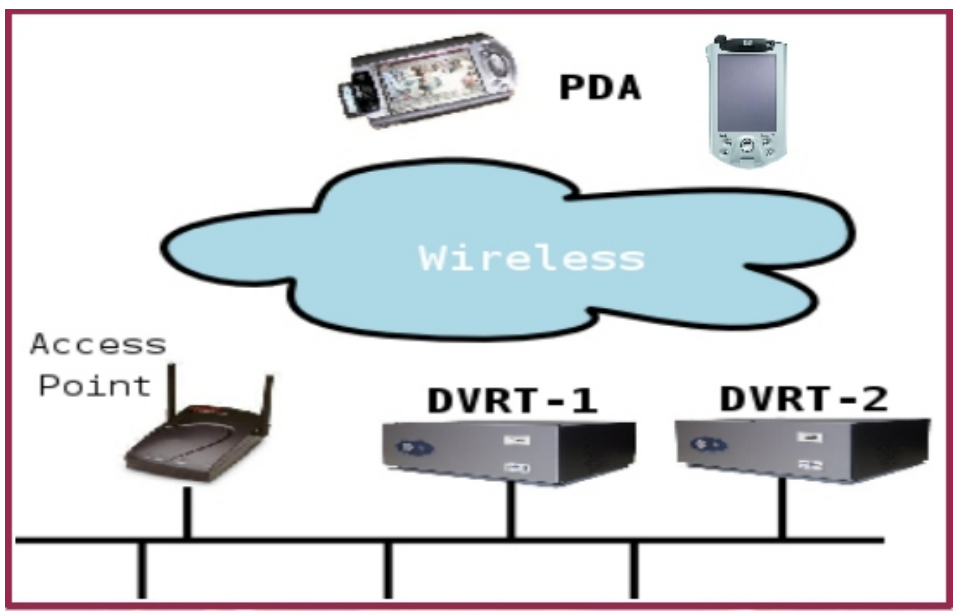

Figure 2-1. Basic Network Topology.

Chapter 3. Implementation

3.1. Methodology

Let's comment, in a few lines, the methodology we are following for the development of VTWV: Extreme Programming (or XP for short).

These are the guidelines of Extreme Programming:

- **Iterations**
  An XP project work in short iterations of one or two weeks.

- **Pair Programming**
  In an XP project all code is written in pairs. There are always two programmers working on the same computer, yes, computer!

- **Testing**
  In an XP project, the testing is done automatically. The team code the testing software the application has to pass. There are both **unit testing** and **acceptance testing**. Moreover, the testing software is written *before* the software itself.
  Unit testing validates each piece of software individually.
  Acceptance Testing validate the whole application. Since the development is **testing-driven**, the software is finished when the acceptance testing is passed. The software is developed to pass the tests!

- **Refactorization**
  An XP project work refactoring continuously the code.

- **Simple Design**
  The design has to be as simple as possible. You never do or design something for tomorrow. If tomorrow your needs change, "refactorize" the code.

- **Everybody owns the code**
  In an XP project everybody owns the code and then everybody can change any line!

- **Continuous Integration**
  Integration of the different modules is done continuously. There is always a machine with the latest software running and being tested. Since all code is released when it passes the tests, the integrated system always works. This way we avoid undefined integration periods.

3.2. Linux Kernel

The patches applied to the standard OCERA kernel to run on the Compaq iPAQ 3850 are `patch-2.4.18-rmk3` and `patch-2.4.18-rmk3-hh10` wich can be downloaded from the Linux Familiar Distribution website, `http://familiar.handhelds.org`.

As the Linux support of HP iPAQ 5550 is continously evolving, we have used the CVS kernel version of handhelds project locate at `http://www.handhelds.org`. On this kernel we have applied generic scheduler patch ported to this kernel version, that we have named `linux-2.4.19-rmk6-pxa1-hh36_gensched.patch`.

On the DVRT is running a Linux Debian Stable kernel 2.4.20 with OCERA generic scheduler patch `linux-2.4.20_gensched.patch` and some other extra patches.

3.3. Classes Structure

VTWV is written in Java. Let's present now the classes structure.

3.3.1. `KoalaRequest`

This class is in charge of making requests to the video server. The protocol to command the video server is implemented in this class. Using this protocol, we can start and stop playing from the specified camera and video server.

The connection established to command the DVRT is done through a TCP/IP socket to the request port of the DVRT and there is one TCP connection per request, i.e., we do open a connection with the server, send the command, wait, synchronously, for the response and close the connection. When there is need for another request we open a new connection with the DVRT.

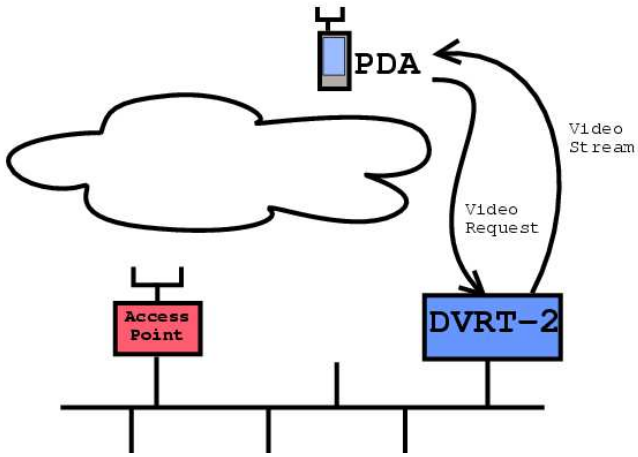

Figure 3-1. The handheld requests a video stream

3.3.2. `ImageReceiver`

This class is an abstract class that provides the common functionality needed to receive a video stream, independently of the compression codec used.

This class is in charge of connecting to the video server and receive the video stream.

3.3.2.1. `JPEGReceiver`

This class is in charge of receiving the JPEG frames sent from the server. We support a particular stream of JPEG images composed of a header containing image size, resolution, etc... and the proper JPEG buffer.

We use this class to ensure that a the whole image have been received and prepare it to be decompressed and displayed.

3.3.2.2. `MPEGReceiver`

MPEGReceiver is the corresponding class to receive the MPEG stream from the network. Using this class we get the MPEG buffer ready for decompression and display. See below how we deal with decompression and display.

3.3.3. `Buffer`

Class to manage byte buffers. Basically it is a wrapper of a byte array that provides extra information about it.

3.3.3.1. `DecodedBuffer`

Derived class of the Buffer class to hold the buffers of the decoded stream.

3.3.3.2. `EncodedBuffer`

Derived class of the Buffer class to hold the MPEG encoded buffers.

3.3.4. `FFMPEGDecoder`

This class is responsible of MPEG stream decompression. To decompress a MPEG stream we rely on native methods that in the end access the `ffmpeg` open source library.

The native methods are implemented using ENI, the Ewe Native Interface, with wich we are able to call the decompression routines of the open source library `ffmpeg`. In order to facilitate the development of these native methods, the Ewe SDK provides the Jewel tool that creates, from the `FFMPEGDecoder` Java class, the C++ source code skeleton which implements the native code.

Right now we have just implemented the decompression into RGB image format since that is what we actually need in order to render the image. In fact, `FFMPEGDecoder` native methods access other dynamic link libraries for YUV to RGB conversion, filtering, and MPEG user data extraction.

3.3.5. `DecoderException`

Class defining the exceptions produced during the MPEG stream decompression.

3.3.6. `VSFinder`

VSFinder is the Java class that finds out the Visual Tools DVRT present in the network. Once the detection process is finished, the discovered DVRTs are presented in a browsable list in the GUI. See below.

3.3.7. `VTWV`

VTWV is the class holding the GUI. It contains the design and behaviour of the graphical user interface. We have, however, separated as much as possible the engine from the user interface. This has been specially useful, other considerations apart, to be able to build the automatic tests corresponding to each class.

The GUI is composed of one panel that contains the image to be displayed and two browsable lists that contain the DVRT present in the local area network and the corresponding cameras. These lists allow the user to select which server and camera she wants to play video from. See Figure 3-2.

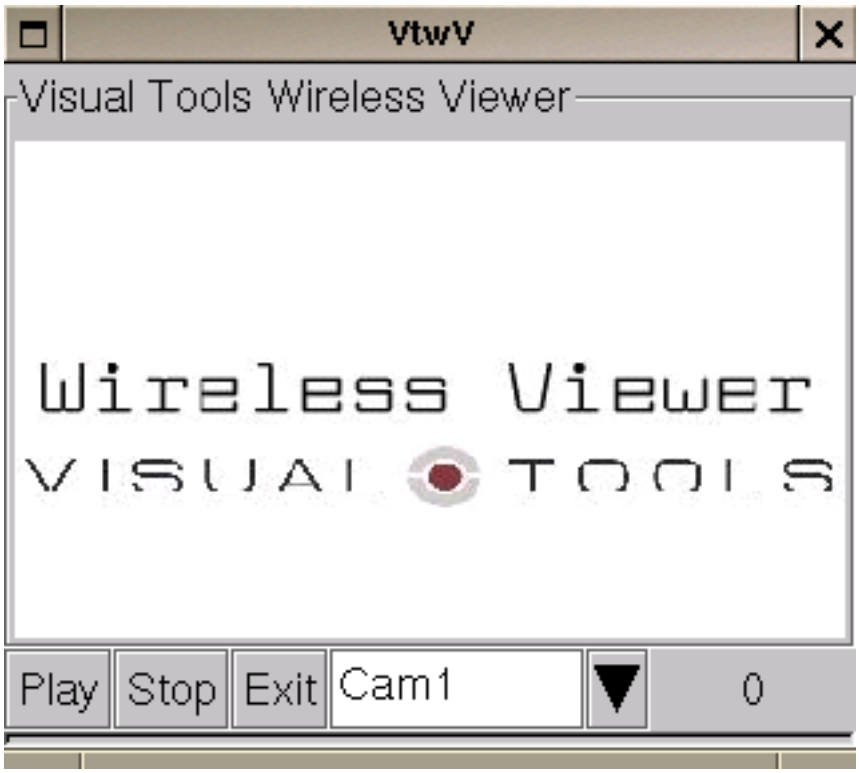

Figure 3-2. VTWV screenshot with debug play/stop buttons.

Details about the expected behaviour of the application are given in the Acceptance Testing section.

Chapter 4. Testing

As already stated in the Methodology section above, there are two main steps in the testing of VTWV, the unit testing and the acceptance testing.

4.1. Unit Testing

For each class presented in the Classes Structure section there is a testing class. Therefore we have testing classes to validate:

- The DVRT Request protocol
  This class, TestKoalaRequest will validate the correctness of the protocol to be used to command with the video server.

- The streaming of video over the WLAN
  There are two classes that take care of the streaming: `TestJPEGReceiver` and `TestMPEGReceiver`. These classes will validate, respectively, that VTWV receive the streams sent from the server through the network.

- The decompression of the video stream
  The class to test the decompression of the video stream received from the server is, TestMPEGDecoder. Notice that there is no need for a decompression class for JPEG since it is built-in into Ewe.

- The discovery of Visual Tools video servers on the LAN
  This class, TestVSFinder will test that all Visual Tools video servers present in the network are found by VTWV.

4.2. Acceptance Testing

The acceptance testing will indicate whether we provide the functionalities needed or not and if they work according to specification. Aside from the fact that the streaming application "runs okay", we have done testing to verify that the appropriate use of the OCERA resource reservation components give the desired results.

The basic points in the acceptance tests are the following:

1. Discovery of the DVRT present in the LAN.
   VTWV will find all the Visual Tools DVRT present in the LAN. To make the GUI as simple as possible this is done only at the application start. If the user plugs another DVRT into the local area network and wants the application to detect it, she must reboot the application. This decision has been taken because it is not likely that new DVRTs appear on the LAN very often and we can thus save resources: CPU used to discover the new ones while probably we are receiving a video stream and network bandwidth. It is also important to keep the GUI as simple as possible since we are targeting small devices.

2. Selection of a video server and camera.
   All discovered video servers will be presented in a browsable list. The user can select one of those servers at a time. Once the user has selected one server, the desired camera from that server can also be selected from a browsable list.

3. Video playing.

   So far the application has detected the DVRT present in the network and the user has selected the desired video server and camera.

   From the very first moment of camera selection, the application connects to the video server and request a video stream.

   The DVRT requested proceeds to stream live video of the selected camera until the user selects whether other camera or another video sever. In each case the application request the corresponding video stream to the appropriate server.

   In order to stop streaming the user have to stop the application, again, to facilitate the use and save GUI we have avoided the inclusion of a STOP button or any other mean to tell the application to stop the video playing.

Chapter 5. Benchmarking

Several experiments have been carried on. Most of them address the problem of showing if the OCERA resource reservation components preserve the application behaviour under different load conditions.

Now we will briefly explain the experiments regarding OCERA QoS mechanisms.

In the first series of experiments, VTWV was run with OCERA QoS components disabled. Once VTWV was receiving, decompressing and displaying video, some synthetic load software was launched. We observed then, the framerate varitation and GUI responsiveness. As shown in next picture, framerate was dramatically affected going down to less than 50%. GUI responsiveness was also affected, getting as high GUI latencies as 1000ms when pressing a button.

In the second series of experiments, VTWV was run with OCERA QoS components enabled. We made several tests tith resource reservation enabled, varying CPU budget and period. When reserving an apropriate amount of CPU budget, measured to be around 75%, we observed constant framerate display and GUI responsiveness similar to that of an idle state. We also observed that, once budget was fixed, the assigned period to the VTWV task did not modified the results.

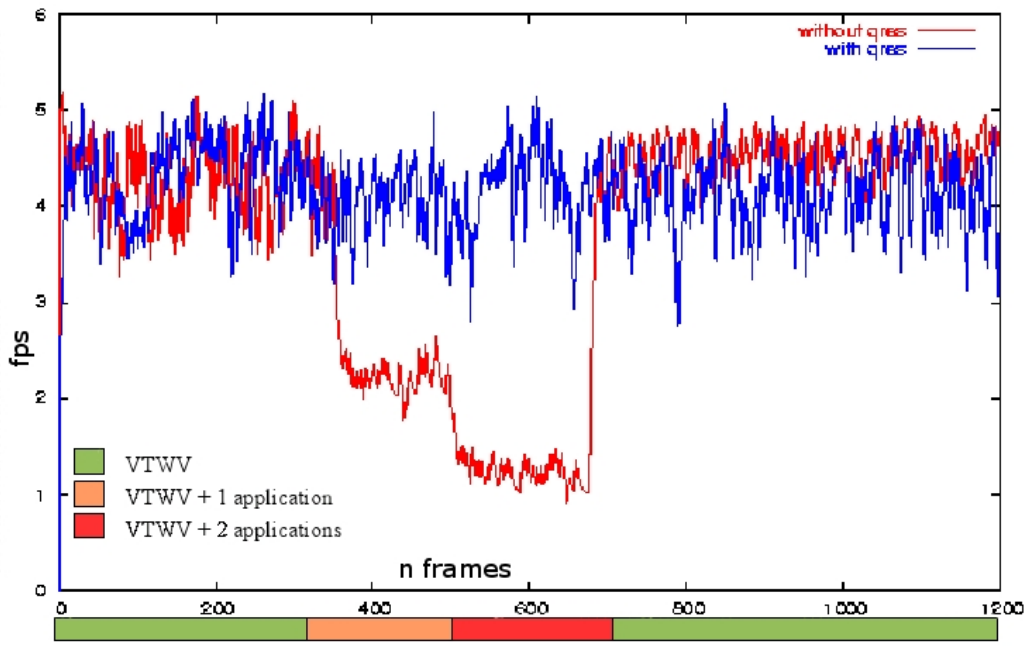

Figure 5-1. VTWV Frame Rate depending on the use of OCERA Scheduler.

We can observe this same behaviour in the following figure, now it is represented the percentage of CPU consumed by VTWV. It can be seen that in the test realized without OCERA Scheduler, there is an important decrease of %CPU when some other applications are launched. Looking at the results from many other tests, we realized that VTWV needs about the 55% of CPU to run smoothly, so configuring properly the scheduler we can guarantee the percentage during VTWV life time.

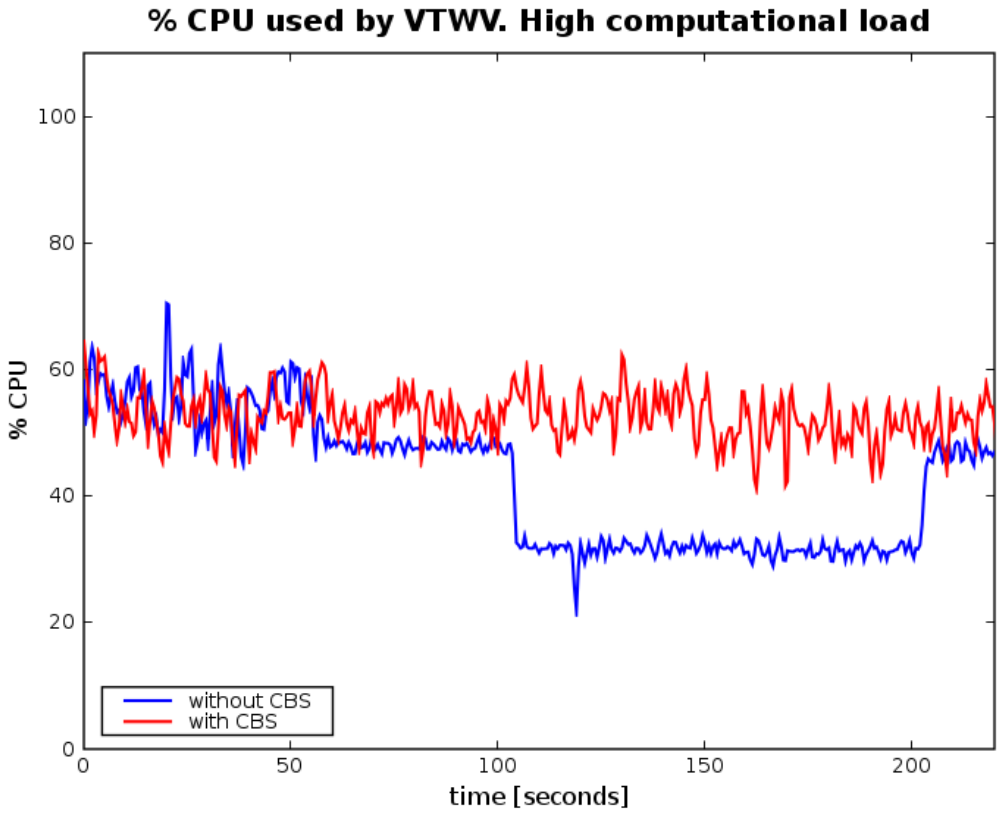

Figure 5-2. Behaviour of VTWV depending on the use of OCERA Scheduler.

As said before, OCERA Scheduler has also been used in the server side, the DVRT. In this case we want to guarantee a constant CPU budget to the Koala Video Server application. The next two figures show the percentage of CPU used by the Koala server versus the total CPU used by the system. In Figure 5-3, the test was run without scheduling, we can distinguish areas where the system is not using the 100% of CPU, there is about a 60% for Koala and about 20% for other tasks. But, when system CPU is at 100%, CPU percentage that Koala can use could be very low, what is very far from what Koala needs. On the other hand, when the Scheduler is used, Figure 5-4, a 60% of CPU is always available for Koala, even in overload conditions.

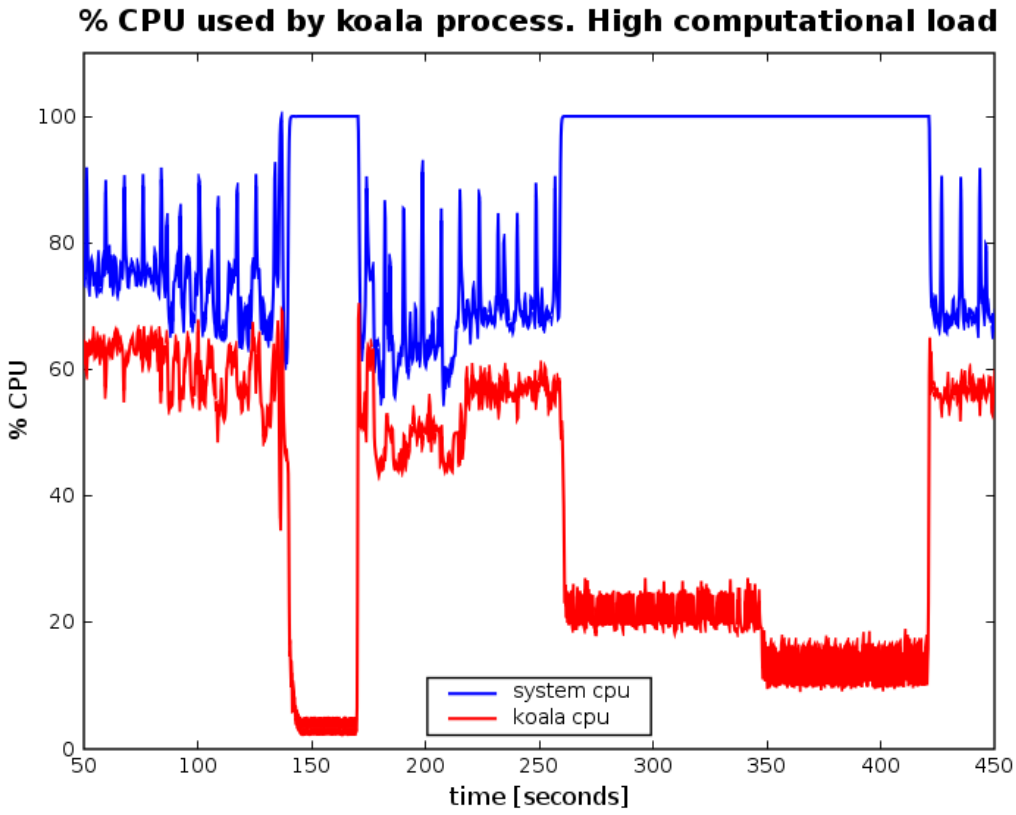

Figure 5-3. Behaviour of Koala Video Server in high load conditions without OCERA Scheduler.

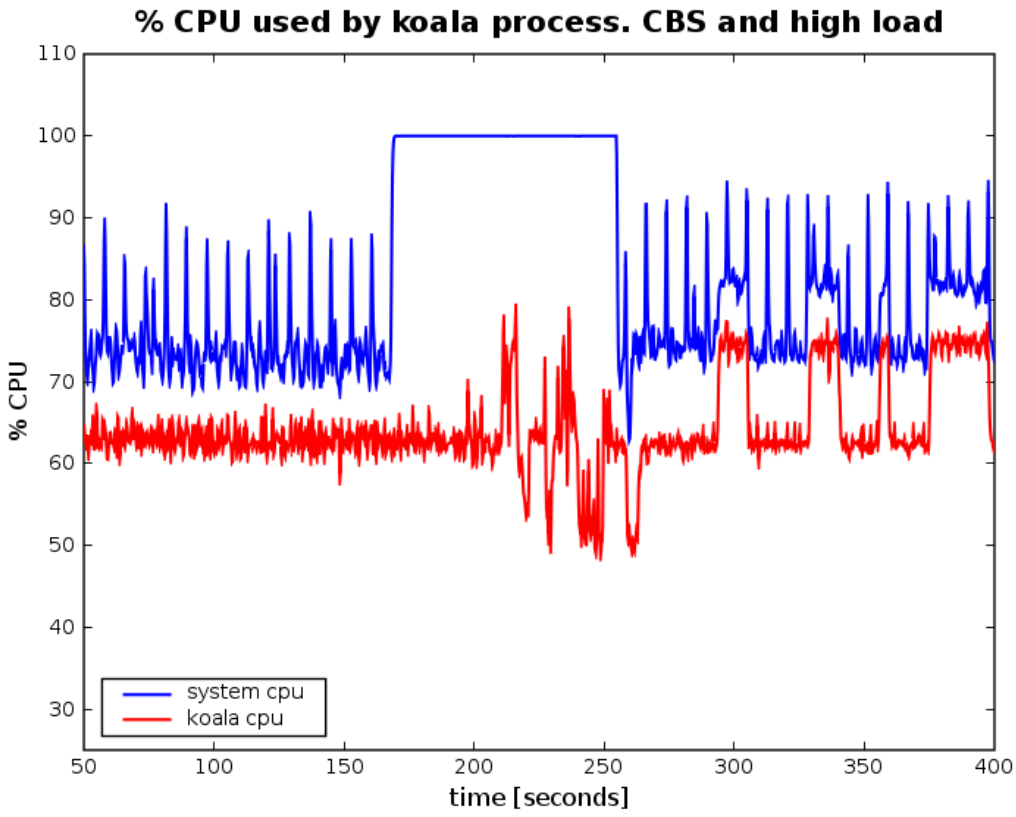

Figure 5-4. Behaviour of Koala Video Server in high load conditions with

OCERA Scheduler.

Some information about JPEG vs. MPEG transmission has also been compiled. In Figure 5-5 and Figure 5-6 some characteristics of the video stream decoding in the VTWV are shown.

First, we observe the maximum frames rate that can be reached using different image resolutions. In the case of MPEG we can decode more frames per second for all the resolutions, and is useful to notice that the difference between is more evident when the image size to decode is smaller.

The second figure represent the compression level with different image resolutions.

We will always use MPEG codification if possible. For example, on the PocketPC the MPEG decoder is not available, in this case we will use the decoding class for JPEG built-in into Ewe.

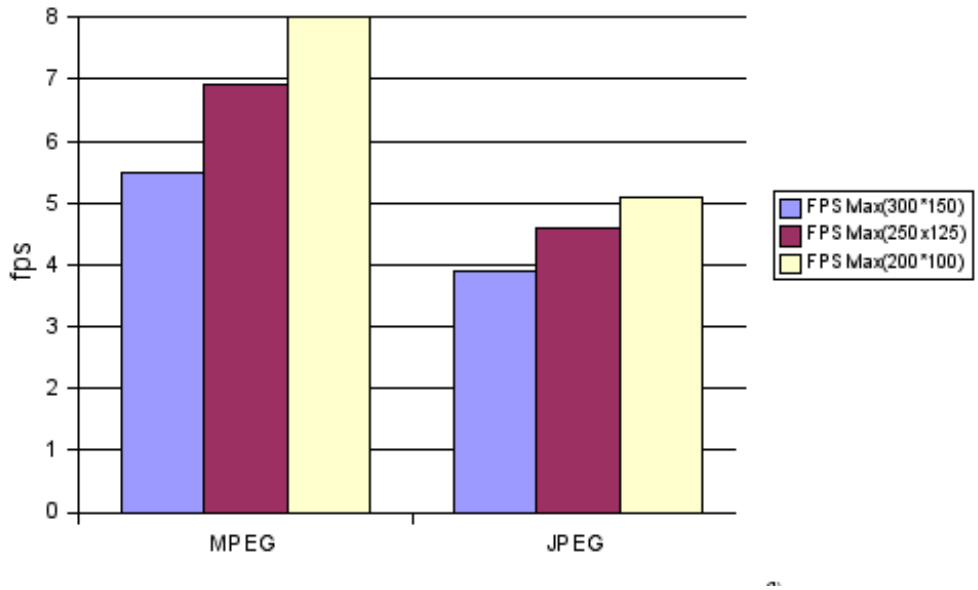

Figure 5-5. JPEG versus MPEG benchmarking. Maximum frame rate.

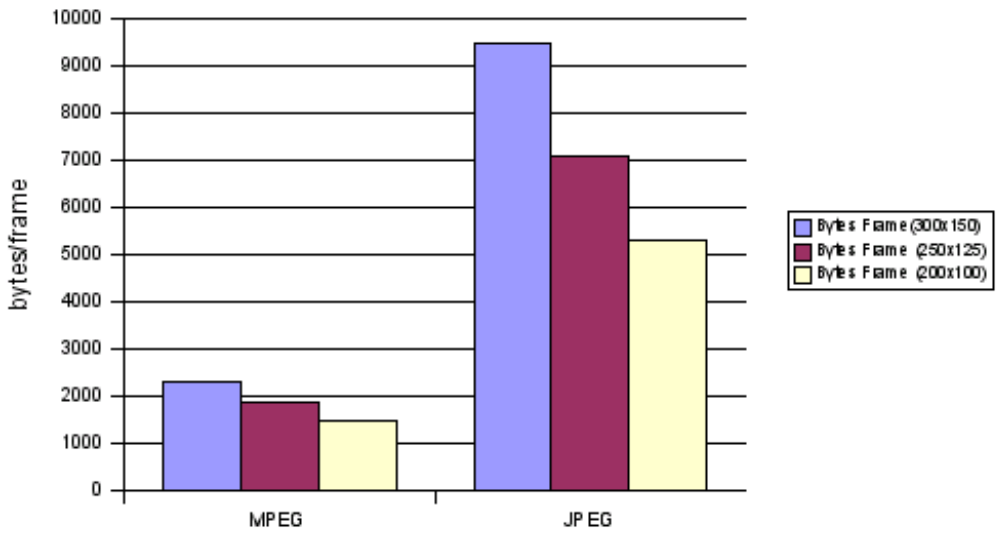

Figure 5-6. JPEG versus MPEG benchmarking. Bytes per frame.

Bibliography

[OCERA D9mm1] Adrian Matellanes, 2003, *"Multimedia Application Requirements and Platform Analysis" http://www.ocera.org*.

[OCERA D9mm2] Adrian Matellanes, 2003, *"Multimedia Application Component Specification" http://www.ocera.org*.

[OCERA D9mm3] Adrian Matellanes, 2003, *"Multimedia Application Version 1" http://www.ocera.org*.

[OCERA D2.1] Adrian Matellanes et al., 2003, *"OCERA Arquitecture and Components Integration" http://www.ocera.org*.

[Familiar] `http://familiar.handhelds.org` .

[EWE] Michael Brerenton, 2003, *"Ewe Application Development" http://www.ewesoft.com*.

[Ewe vs PersonalJava] Michael Brerenton, *"Comparison of the Ewe VM with a PersonalJava VM" http://www.ewesoft.com/EweDetails/VersusPersonalJava.htm* .

[Arbaugh et al. 2001] William A. Arbaugh, Narendar Shankar, and Y.C. Justin Wan, 2001, *"Your 802.11 Wireless Network has No Clothes"*.

[Walker2000] J. Walker, March 2000, Tech. Rep. 03628E, IEEE 802.11 Committee, *"Unsafe at any size key: an analysis of the WEP encapsulation"*.

[Borisov et al.] N. Borisov, I. Goldberg, and D, Wagner, *"Intercepting Mobile Communications: The Insecurity of 802.11."*.