DEPARTMENT OF COMPUTER SCIENCE

RPVR: Remote Personal Video Recorder

Matthew James Strain

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering

April 2006 | CSBSC-06

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Matthew James Strain , April 2006

ABSTRACT

Many people throughout the world are addicted to watching their favourite television programmes. The consumer electronics market shipped over 9.4 million DVD recorders in 2004, which is expected to rise to 67.7 million in 2009 [6]. If you add that to the amount of video recorders that have been sold, there is clear evidence that consumers are willing to spend their hard earned cash not to miss their favourite programmes.

Most timer recording systems are cumbersome, difficult to use and rely on the user remembering to programme them. I was recently caught out myself setting a timer recording, due to the hour change to British Summer Time.

Many homes now have computers with broadband connections and the number of computers with TV Tuner cards is rising. This project is about utilising this processing power and internet connectivity to provide the ultimate recording system.

This project is an application suite, which allows the viewing and recording of TV programmes on a computer, something which is not new. What is new is that it also offers a HTTPS server and a mobile phone application. The secure web server will allow the user to schedule a recording from any web browser on the internet and the mobile phone application will allow the user to schedule a recording from their phone.

A user may be sat on a beach in Thailand and realise they have forgotten to record Eastenders. With the Remote Personal Video Recorder (RPVR) on their phone, they can set their computer to record it for them, without even having to get up from their sunbed.

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

1.1 Motivation

Many years ago, we were told on TV programmes such as Tomorrows World, in the press and even in books, that we would soon live in a world where we can control everyday household items such as washing machines, ovens and central heating systems from not only any part of our house but also from anywhere in the world.

Unfortunately so far this is not the case. Perhaps this is fortunate as I am not sure if I want the security risk of someone being able to hack into my heating system, turn off my hot water, so I have to wake to a cold shower in the morning.

For household appliances, remote control is not possible because of the human intervention required. A person is still required to put dinner in the oven before turning it on or to put washing in the washing machine to clean it. So until we develop robots that can do it for us, we will have to do it ourselves whilst standing in front of our appliances.

However what about our Televisions, Videos or PVR recorders? These could in fact be quite easily controlled remotely, in fact many cable TV set top boxes have unused Ethernet sockets in them.

Why would we want to be able to control Television devices you may wonder? The answer to this is to consider the fact that on average around 12 million viewers tune in to watch each episode of Coronation Street and Eastenders. The majority of these viewers being regular watchers, hooked on their favourite TV programme. There are many people that hate to miss their favourite programmes and use a facility to record them if they are unable to watch them when broadcast. However what if they forget to set the timer on their video recorder? Well that is where the Remote Personal Video Recorder (RPVR) comes into play.

1.2 General Aims

The focus of this project is to provide an easy to use and reliable facility to record television programmes on a computer, with the primary objective to provide an interface to allow recordings to be scheduled via a web browser or a mobile phone over the internet.

Stealing a slogan from a consumer electronics manufacturer, I would like to be able to provide ‘simplicity’, by producing an application that is both intuitive and simple to use. The reason for this is to try to ensure that it becomes attractive to the large consumer market, to which it is aimed. This is a project that could be used by age groups ranging from teenagers to adults in their 60’s if it is produced correctly.

Reliability of product will also be key in developing a worthwhile and useful product. For example when a user selects to set a timer recording, the application must ensure that it can be relied upon to actually record it.

Security must be considered to ensure that user's computers can't be intruded upon and their hard drives filled with recordings they didn't schedule. For this reason SSL connections over HTTPS will be used.

Efficiency will also be important, especially with regard to internet traffic to and from a mobile phone, which can be expensive. This traffic must be kept to a minimum for people to want to use it. Considering that all traffic with have the overhead of HTTPS SSL packet headers, keeping the actual data which is sent between clients to a minimum is a must.

As an overall aim, if I am able to achieve the specific goals listed below, I will have considered this project a success.

1.3 Specific Goals

The goals of this project are to:

1. Provide a program to watch and record television on a computer.
2. Provide a timer recording mechanism.
3. Provide a web browser based facility to control timer recording.
4. Provide a mobile phone application to control timer recording.

1.4 Limitations

Due to the large range of different hardware devices available, this project will be developed for particular TV Tuner device, simply for the fact I can't afford to buy many TV Tuner cards to test with; however the need for program adoption for different hardware manufacturers will be kept in mind and will be the subject of future development.

The application will also be developed for use with the five analogue terrestrial channels available in the UK; again however the need for portability to other countries and also Digital DVB reception will be kept in mind and will be the subject of future development.

1.5 Usefulness

This is definitely an implementation project rather than a research project. I chose this sort of project as I knew I would find it exciting and fun to do. Beyond this though, this project will involve many different aspects of software development, in many different programming languages, all of which will improve my skills as a Software Engineer.

As well as the personal benefits I will gain while developing this product, it also opens the possibilities of actually producing a product which is marketable and useful to many consumers throughout the world.

1.6 Structure of Thesis

This document outlines the development steps of this project, which has mainly been an extreme programming approach, using prototyping followed by design, coding and then testing on small modules of the project rather than the typical Spiral lifecycle model.

However this report does follow the Spiral lifecycle model, as each development stage has been combined into the following sections:

Section 2 covers the background research and investigation. This looks at what products already exist and what the potential market is for a product such as RPVR. This is mainly in the form of a market survey, which is suitable, considering the type of project.

Section 3 covers the prototyping, design and specification steps that I took, with an indication for the reason for particular design choices and any early drawbacks that were discovered following prototyping.

Section 4 covers the actual code development process for each part of the project. This is divided into three sections, one for each of the programs written for the project.

Finally section 5 illustrates the results of the project, what features are successful, what works and what does not. It also investigates any future developments that could take place.

2 BACKGROUND AND RESEARCH

2.1 The Market

Many homes in the UK and worldwide, have a PC with a broadband connection. This device is obviously able to be controlled remotely via the internet and is not a new development. However most remote access features are currently designed for remote working to log on to servers at work or to allow IT support personnel to assist users. My idea is to use these PC's which are often sitting idle not doing anything, to record TV programmes.

Whilst at present homes with centrally controlled media centres, providing Audio and Visual entertainment is limited to the rich and famous, this trend is decreasing as both the equipment becomes cheaper and with the advancement of PC's being able to do the work.

At the end of September 2005 there were 6.38 million broadband connections in the UK and over 125 million broadband connections worldwide [1]; this means there are a large number of PCs that can be accessed via the internet.

At the end of 2004 over 1.1 million TV tuner cards had been sold worldwide [2] and with many PC Motherboard manufacturers looking to include TV tuners in their standard motherboard specifications, there are likely to be many users watching TV on their computers. In fact a recent survey found that 51% of online users were interested in recording TV programmes on their computers [3].

Although TV tuner cards and external adapters plug into a computer, that does not mean that the computer has to output to a computer monitor. For many years a TV out socket has been included with new PC purchases and there are large numbers of users who have their PC hooked up to their TV, this is mainly thanks to the number of users watching downloaded movies. A recent survey found that 40% of online users were interested in watching PC recorded TV programmes on their televisions [3].

My idea is to utilise, PCs with broadband internet connections and TV tuner cards to be able to record TV programmes. This idea is not new and there are several products currently available which do this, in fact the current version of the Windows Media Centre has features which provide this function.

However what the current programs do not facilitate is remote activation of recording. In essence this is the fairly simple idea of starting to record a program remotely either via a mobile phone or from another computer on the internet.

There are 61.2 million mobile phone subscribers in the UK [4], the majority of which have an internet connection. These internet connections are either: 3G (3rd Generation), GPRS (General Packet Radio Service) or WAP (Wireless Application Protocol). This means that a user's mobile phone could easily be used to send a message to their PC, to tell it to start recording a TV programme on a particular channel.

With these figures in mind, there is clear evidence for a market for this type of product.

2.2 The Competition

Whilst I have been unable to find any technical solutions which match my project idea, the project does have some competition in the form of rival technologies, these are:

2.2.1 Sky+

Sky+ is a system which is an advanced set top box, in that it contains a hard disk for program recording. This system can record, pause and rewind live TV. The main feature of Sky+ which is a competitor to my Remote Personal video Recorder (RPVR) is that you can schedule the box to automatically record an entire series for you, as well as providing scheduled recording. However it does not provide the main feature of RPVR, it does not allow you to start recording a program remotely. Also whilst the series record function is good if you forget to schedule a recording, it has limitations. For example if you wanted it to record every episode of Eastenders you would be okay, as all Eastenders programmes are on at regulars times. However if you wanted it to record all live Skiing events, it would not be able to do it as they occur at different times throughout the ski season.

2.2.2 Personal Video Recorders (PVR's)

Current personal video recorders basically provide the same features as Sky+, in that they allow you to perform scheduled recording and to pause, record and rewind live TV. Again however they do not provide the facility to activate recording via the internet.

2.2.3 TV on your Mobile

In November 2005, Vodafone launched its TV on your mobile facility, where users can watch TV programs on their mobile phone. Whilst this may seem attractive at first there are some drawbacks.

The main drawback is the screen size of the handset. The average screen size for these handsets is 1.5" which is far too small to be able to get an adequate viewing experience. Also the service is limited to when the handset is in 3G coverage, which at present is limited and also does not perform well for TV and data purposes when travelling at speed in a car or on a train.

Also to be able to view the TV programmes a subscription is required which starts at £5 per month rising to £10 per month [5] depending on the amount and types of channels subscribed to.

2.2.4 TV on Demand

HBO in America launched 'HBO On Demand' a few months ago and UK company Home Choice launched their own version in the UK recently. TV on demand is a subscription service which allows you to choose from a bank of programs and films stored by the provider to watch when you want.

However the bank of programs is limited and in the UK only the most popular TV programmes are stored and only that week's episodes are available. In the US, HBO also offers recent sporting events as well as typically the last six episodes of a particular series.

The main drawback to this service is that it is only available to digital cable subscribers and also a subscription fee applies.

3 SPECIFICATION AND DESIGN

3.1 Program Structure

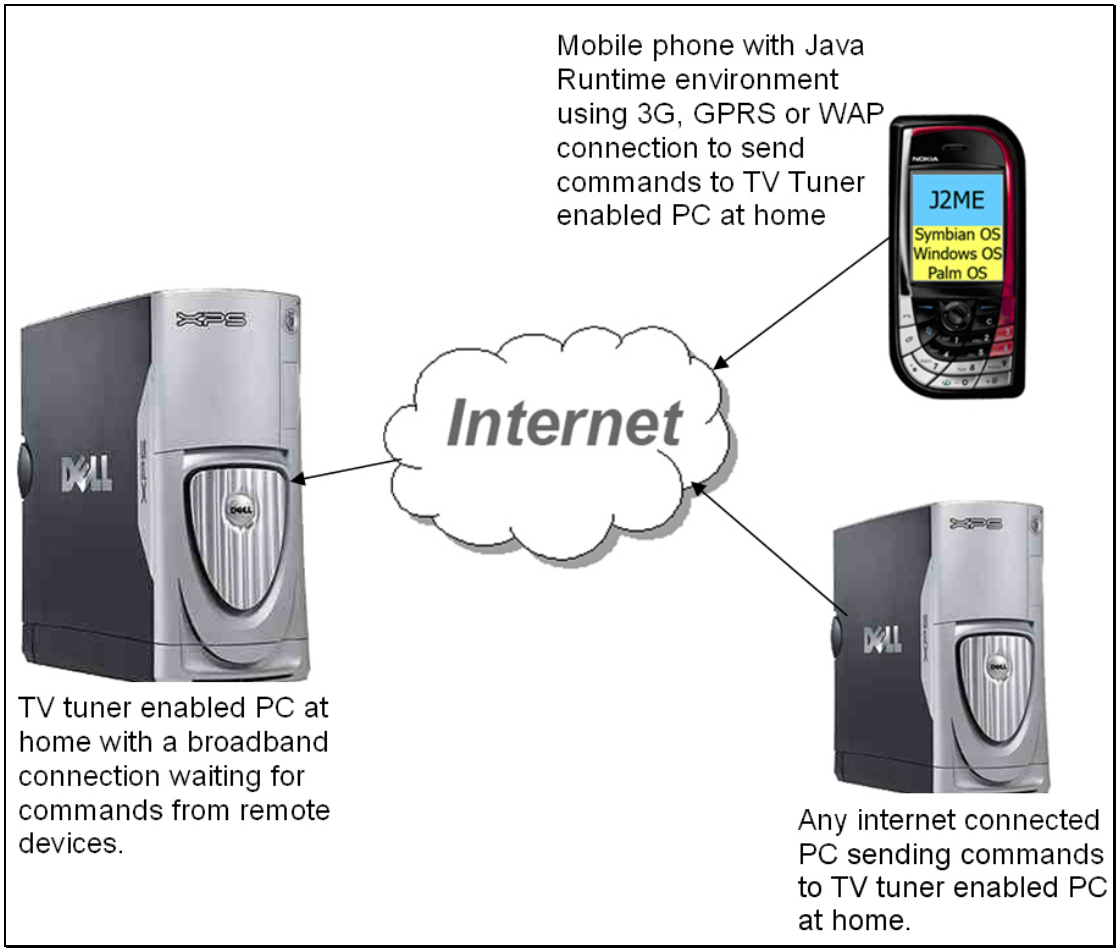

Figure 1 - System Overview

Figure 1 above shows an overview of the system architecture. In all the system will consist of three programs. One program will control and operate the TV tuner card, the second will be listening for connections from remote sources and the third program will reside on a mobile phone to send commands to the TV tuner enabled PC. Remote computers will use a web browser to access the PC remotely.

3.1.1 TV Tuner Operating Program

This program will operate the TV tuner card or adapter connected to the PC. It will provide features such as:

- Channel Tuning (including auto tune).
- Record Live TV.
- Full screen and partial (windowed) screen viewing.
- Media file playback

3.1.2 TV Tuner Scheduler and Remote Access Point Program

This is a program which will always be running in the background while the PC is turned on. It will activate the TV tuner operating program to record a program upon a scheduled recording time being reached.

It will also act as server program listening for remote connections from clients, wishing to schedule a recording of a TV program from remote sources being either a mobile phone or another computer on the internet.

3.1.3 Remote Mobile Phone Program

This program will reside on a mobile phone and will communicate via an internet connection to the user's home PC remote access point program. It will allow a user to log on to their remote access point program, retrieve TV schedule information and schedule a recording of a TV programme on a particular channel at a particular time or immediately.

3.2 Early Technical Issues Raised From Prototyping

I originally planned to write all program code in Java. I thought this will be easier for me as I have more experience coding in Java but also I would have the benefits of cross-platform compatibility as well. Unfortunately early prototyping discovered this would not be possible as the following sections describe.

3.2.1 TV Tuner Device Control

I planned to use the Java Media Framework (JMF) to control the TV tuner card, however I came to realise this was not going to be possible. I managed to write some program code that would allow you to watch TV on a computer; however as JMF use's legacy 'Video For Windows' (VFW) drivers, for which most TV tuners claim compatibility, I stumbled across a major problem.

For my program to work I obviously need to be able to tune (change the channel on the TV tuner) programmatically. This is because I would like to be able to provide an attractive and easy to use interface to change channels with nice channel up and down buttons. I also need to be able to change to a particular channel to record a program based on a scheduled recording made by the user.

However using JMF the only way to change channel is by calling a control object. This control object calls the TV Tuner card driver interface which delivers the window shown in Figure 2.

This window is in essence a tuning window not a channel change facility and it does not store TV channels. This would result in both a poor interface to the user and also as you have to physically click the buttons to change channel I would not be able to change channel programmatically in my code, rendering my project impossible.

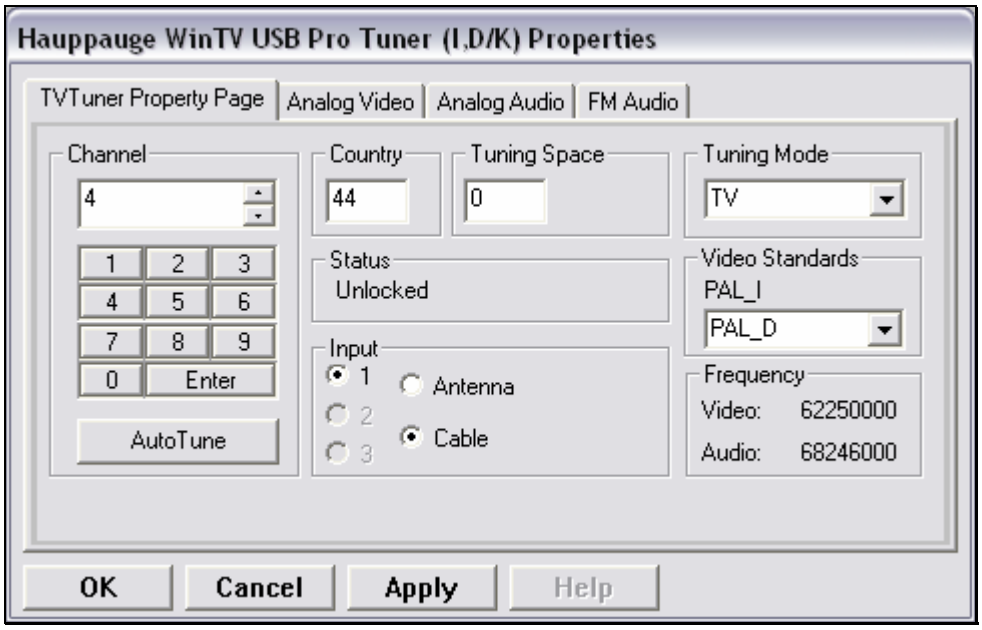

Figure 2 - JMF TV Tune Window

At first I was not convinced that this was the only way to change channels but after much investigation and lots of threads on the JMF forum on Sun's website I soon realised this was the case.

The main reason for this is the fact that JMF uses VFW drivers, which are being phased out and are being replaced with Windows Driver Model (WDM) drivers, which are not compatible with JMF.

I soon discovered that I would need to write this program using Microsoft's .net environment using an API of DirectX called DirectShow. DirectShow uses WDM drivers, which are compatible with all TV tuner devices and DirectShow provides a set of classes, which would make this program possible.

This however meant that my program would only run on Windows operating systems, therefore stopping my cross-platform compatibility.

Whilst at first I was disheartened by this, I soon came to realise this was not such a big issue as I myself was unable to get my TV tuner card to work under Linux and whilst I don't consider myself to be a Linux guru, I believe I am more experienced than the average PC user.

I also felt that using the .net environment would be a good learning experience as I expect it is an environment I will use when I go into industry, although it does mean learning to use a new environment and a style of C++ I have never used before.

3.2.3 Mobile Phone Internet Communication

The program to reside on a mobile phone will be written in Java 2 Micro Edition (J2ME), which runs on mobile phones, PDA's and other handheld devices.

Early research into the programming API for this environment led me to believe I would be able to use a Socket class, which provides a standard TCP/IP socket and is very similar in functionality to the Socket class of J2SE for PC's.

It was my original plan to open a socket and establish input and output streams to then be able to send bytes of data from the phone to the remote PC. However an early prototyping program soon led me to discover that this would not be possible.

J2ME is made up of two profiles which are the Mobile Information Device Profile (MIDP) and the Connected Limited Device Configuration (CLDC). These two profiles are the API's which provide the classes for the J2ME runtime environment.

However for devices to be MIDP 2.0 compliant it is not required for hardware manufacturers to implement the TCP/IP socket classes and also even if they did, not all mobile phone networks support TCP/IP traffic over their networks in this way.

To work around this problem all devices must support HTTP and HTTPS protocols to be MIDP 2.0 compliant, so use of these protocols would allow me to communicate via a mobile phone.

3.3 DirectShow

DirectShow is a programming API of DirectX and is comprised of two classes of objects: filters, the atomic entities of DirectShow; and filter graphs, collections of filters connected together to provide a specific functionality [7].

3.3.1 DirectShow Filters

Filters are the basic units of the DirectShow API and are the essential components of a filter graph. Filters can have many different functions but they must contain some method to receive or transmit a stream of data.

Each filter has a least one pin, which provides a connection point from that filter to other filters in the filter graph. These pins are either input or output pins that either receive a stream from another filter or send a stream to another filter. Filters have a least one of these pins, some have many; it all depends of the type of filter.

There are three basic classes of filters, which are source filters, transform filters and output or rendering filters.

Source filters produce a stream of data to be sent along a filter graph. Examples of source filters are 'File Filters' (used for reading media files) and 'Capture Filters' (this might be a filter to capture sound from a sound card).

Transform filters receive an input stream from a source filter and perform some operation on that stream before passing it on to another transform filter or a rendering filter. Transform filters can also perform a tee in the stream, which means that the input stream is duplicated and placed on two or more output pins.

Renderer Filters translate a DirectShow stream into some form of output. This might be a video rendering filter, to render video to the monitor, an audio rendering filter, to send sound to speakers or a File Writing filter, to write the stream to a media file.

3.3.2 DirectShow Filter Graphs

Filter graphs execute sequentially, from the first filter to the last. Data enters the graph at the source filter, passes along to the second filter and so on. Filter graphs execute continuously like a waterfall descending a series of stairs in the form of a data stream.

Filter graphs provide the mechanism to start, stop and pause the filters in the graph. It also provides the synchronisation clock to ensure that all filters and the media samples they are playing are kept in sync.

A simple filter graph to capture audio from a microphone, play it back and record it to a file in pseudo code might take the following structure:

```
FilterGraph()
{
      AudioCaptureFilter();   // The source filter
      TeeSplitterFilter();    // The transform filter
      AudioRenderer();        // A rendering filter
      FileWriter();           // Another rendering filter
}
```

3.4 The TV Tuner Program

This program will be written using Microsoft Visual Studio in the form of a C++ Win32 Application project. This type of project uses the raw C++ API, without any graphical form editing tools. The reason for this choice is that it is required for using DirectShow.

The alternative to this would be to create a Dynamic Linked Library (DLL) for the DirectShow operations, and then create another project using .NET for example to create the GUI and call the DLL for the required operations. However early investigations have led me to believe this is quite a complicated way of doing things and as I have not used the development environment before I decided to keep things simple and do it all as a Win 32 application project.

3.4.1 Graph Edit

Graph Edit is a tool which allows you to visually create a filter graph, add filters and connect the pins of the filters. This is done before programming begins to design the graph and work out which filters are required to perform the necessary operations.

It soon became clear that I would be dealing with three different filter graphs for the TV Tuner program. One to watch television, one to record television and another to play media files.

3.4.2 The TV Viewing Filter Graph

Figure 3 shows filter graph required to watch TV on a computer.

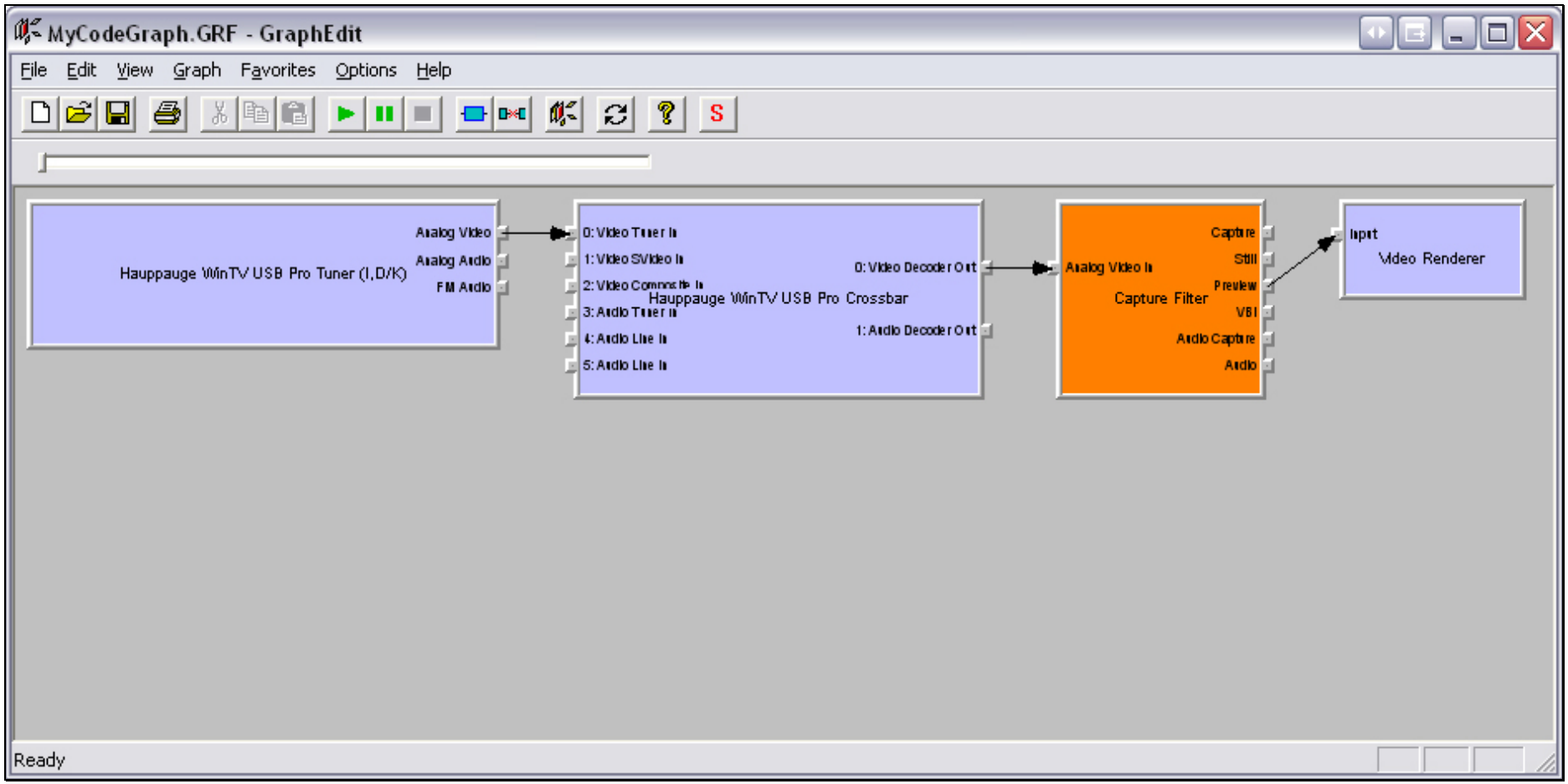

Figure 3 - The TV Viewing Filter Graph

The Hauppauge WinTV USB Pro Tuner is a hardware specific source filter, which generates the media steam for the filter graph. This filter allows setting of the country tuning space (the range of frequencies for a particular country) and the aerial source. The only output pin that is required to be connected is the Analogue Video output pin. This filter also exposes an IAMTV interface, which allows the tuner to be tuned to a particular channel programmatically.

The Hauppauge WinTV USB Pro Crossbar is a transform filter and is connected to the tuner filter. This again is hardware specific and can be used to separate media signals such as FM radio and also the S-Video input. These features are not used as they are hardware specific and do not relate to this project.

The capture filter is a DirectShow filter that takes in the video input from the Crossbar filter and provides various output pins, one of which, the preview pin, is connected to a Video Renderer filter which displays the picture in a window.

This Video Renderer filter would be substituted for a Full Screen Renderer filter to show a full screen view of the TV program rather than a windowed view.

The audio is actually handled by the capture filter but is connected internally. The reason the connection is not shown on the graph is because TV Tuner cards have a cable which runs directly from the tuner card to the line-in connection of the sound card.

3.4.3 The TV Recording Filter Graph

Figure 4 shows the filter graph required for recording TV. This filter has exactly the same filters as shown in Figure 3, except it has a few more, to handle the recording of a video file. The two extra filters are an audio capture filter and a Windows Media File Writer filter.

The audio capture filter exposes the features of the sound card. The capture output pin is connected to the Windows Media File Writer filter. The audio capture pin is a source filter in this context, which is why none of its input pins are connected. Again the reason for this is because the TV Tuner card has a physical connection to the line-in port, of the sound card.

The Windows Media File Writer filter accepts both the audio and video streams and encodes them to the Windows Media Video format and saves them to a video file. This filter allows setting the name of the file to be encoded and also the type of encoding used. After testing which formats, produce the best results, with regard to performance and also video file quality, the format of “Windows Media 8 - Best Quality based VBR for Broadband” was selected. This is a variable bit rate encoding scheme, which did not use too much system resources to encode, produced a video file of reasonable size and of good quality.

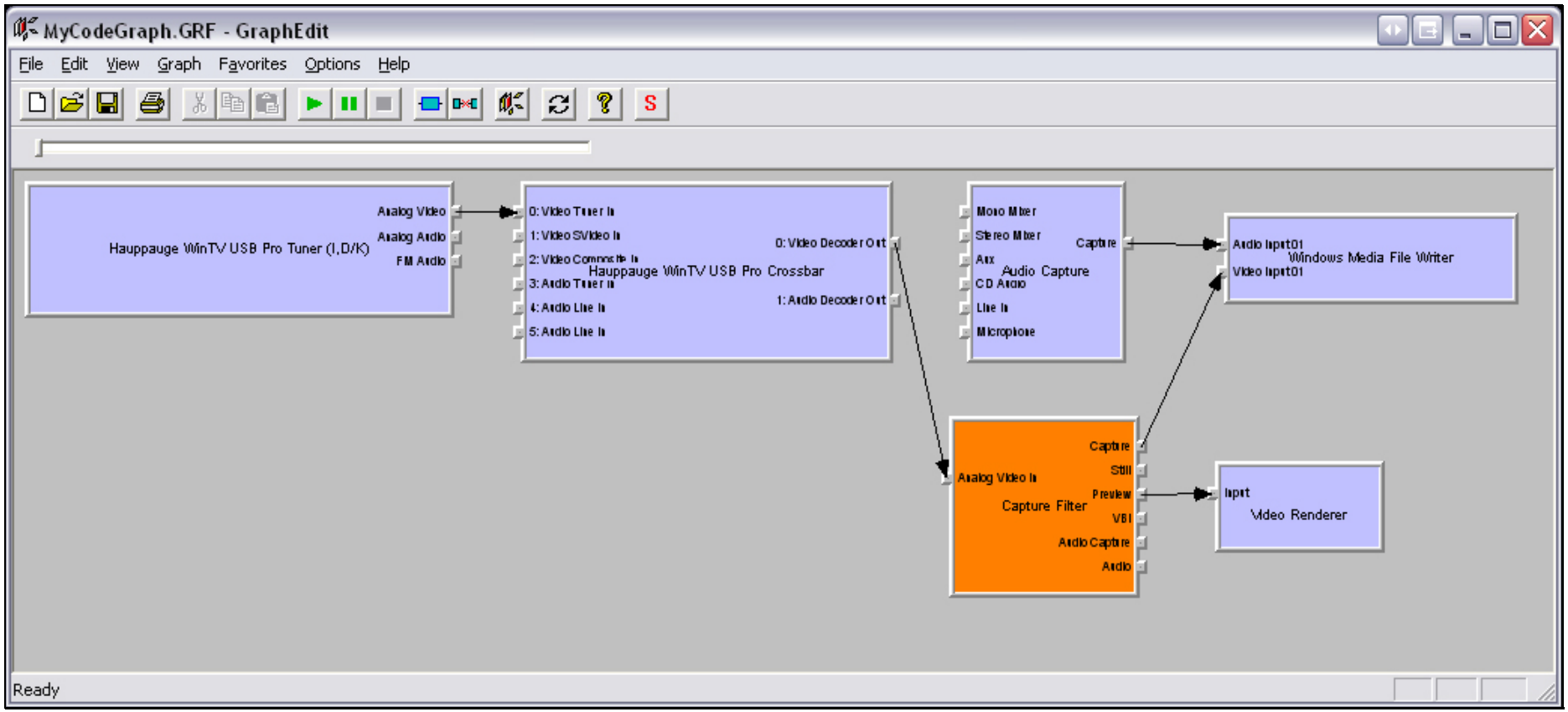

Figure 4 - The TV Recording Filter Graph

3.4.4 The Media File Viewing Filter Graph

Figure 5 shows the filter graph required to play a Windows Media Video file. The WMV reader filter reads the file and passes the audio stream to a WM Audio decoder, which is a transform filter which passes the stream to the Audio Renderer filter.

This process is similar for the video stream which is passed to a transform filter, the WM Video Decoder filter, which passes the stream to a Video Renderer filter.

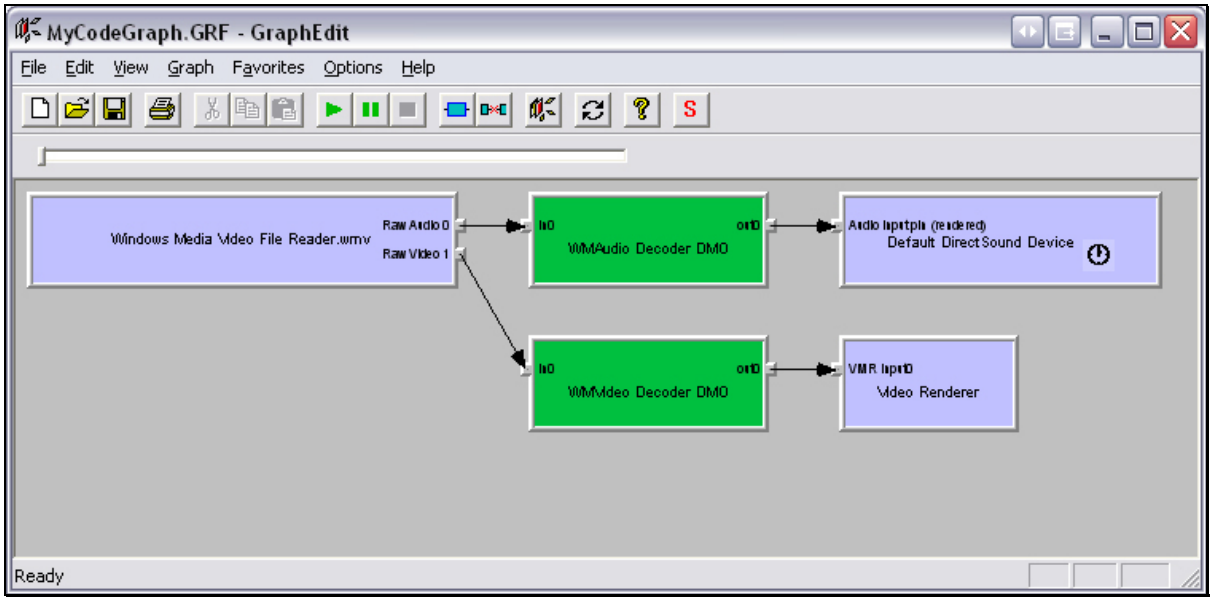

Figure 5 - The Media File Viewing Filter Graph

In fact playing media files is relatively easy, as a method of the Filter Graph class the RenderFile method can be called, with the name of a file passed as an argument. This method automatically builds the required filter graph for that particular file, if the correct codec's are found on the system.

3.4.5 Program Design

Early prototyping soon led me to realise that a GraphManager class would be required for building the various filter graphs, controlling the filters, the video output and also to start, stop and pause playback.

I also decided that using another class to operate the IAMTV Tuner interface a Tuner class could be used, to set the channel and other tuner operations.

After using the process of verb / noun analysis and using CRC Cards I deduced the basic classes as shown in Figure 6.

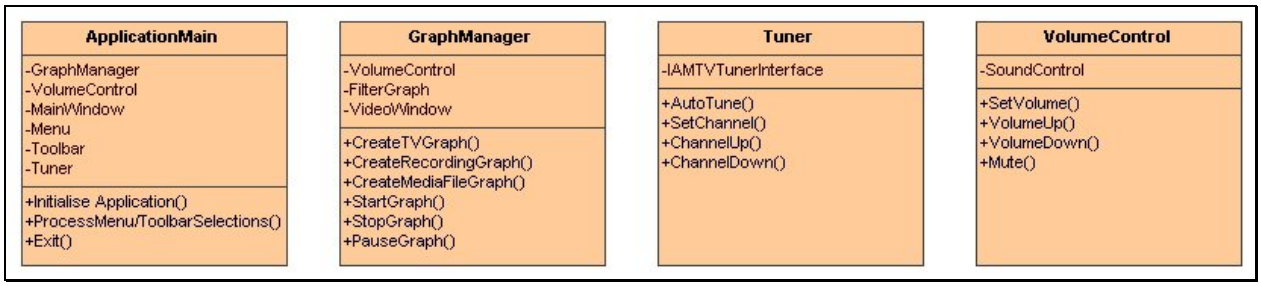

Figure 6 - The Base TV Tuner Application Classes

3.5 TV Tuner Scheduler and Remote Access Point Program

This program will be written in Java and will run in the background with an icon in the system tray. It will provide an interface to view, add, edit and delete recording schedules and also network configuration items such as which port it should listen for connections on.

It will manage the recording schedule and when a scheduled recording time is reached, it will get a runtime object and automatically start the TV tuner program, passing it command line arguments including which channel to tune to and the duration of the recording.

This program will also act as a secure web server using the HTTPS protocol and Secure Sockets Layer technology. Using these encrypted transport mechanisms will ensure that data sent between the server and clients will not be able, to be viewed by third parties especially during password authentication.

This program will listen for internet connections from the internet from either the user's mobile phone or another computer on the internet to allow recording schedules to be added, changed or cancelled.

The program will detect whether the client is using a web browser or the Remote Mobile phone program and pass back results accordingly. This is because the amount of traffic sent to a mobile phone must be limited to save money and also as a mobile phone has limited memory and processing resources.

When receiving requests from web browsers, the use of JavaScript pages will be used to perform client side programming to perform the necessary tasks of the standard html pages that will be delivered.

On the server side, some of the html pages will be created or modified at runtime from within the code of this program.

An alternative to this method would have been to deploy Tomcat Server to run as the web server, using a database to store the recording schedule and Java Servlets to perform the necessary actions. This method I feel however is too complex and would make matters far more complicated than necessary.

Using a database to store the information would be far too expensive, especially considering that it would only ever hold a few records at most. Also all the features of

Tomcat are not required. Using this option would not only over inflate the application, but would make installation a nightmare.

On the security side, as well as the SSL connections, frames based web pages will be employed to hide the real contents of the web requests in the user's web browser address bar. Also care will be taken to ensure that no caching of the web pages will be made by the client web browsers, again this will help improve security.

Using the same methods of verb / noun analysis and CRC cards, the classes shown in Figure 7 were deduced.

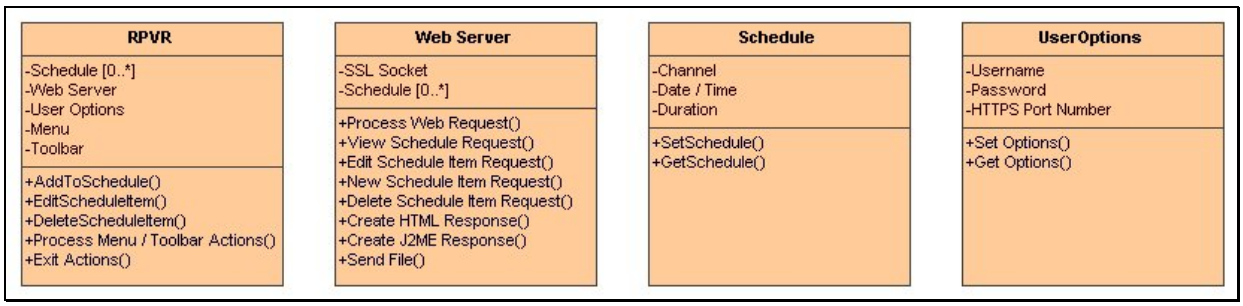

Figure 7 - The Scheduler Program Base Classes

3.6 Remote Mobile Phone Program

As mentioned previously the mobile phone application will be written in J2ME, using the Mobile Information Device Profile (MIDP) 2.0 and the Connected Device Limited Configuration (CDLC) Profile 1.0.

This program will be responsible for providing an HTTPS connection to the server, and providing a facility to view and modify the recording schedule.

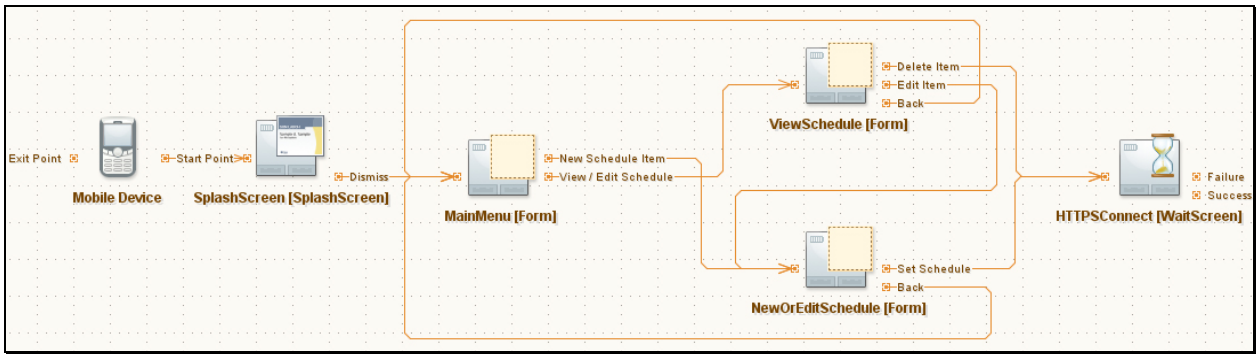

Figure 8 - Mobile Phone Application Flow

Figure 8 shows the proposed program flow between the various displays of the program.

The program will start, showing a splash screen, while the program initialises. A main menu will then be displayed offering choices of actions to the user.

From the main menu the user will be able to select to create a new schedule, which will load the New or Edit display screen, which will be used to set a new schedule.

Also from the main menu will be an option to view the existing schedule, at which point the program will connect to the remote computer and retrieve the recording schedule.

From the view schedule screen, the user will be able to edit and delete schedule items.

When a connection to the remote computer is necessary, a wait screen will be displayed whilst internet communication takes place.

4 IMPLEMENTATION

4.1 TV Tuner Program Development

The TV Tuner application took by far the longest to develop. This was partly due to its complexity but also to the fact it was written in C++, a language I haven't used for at least 18 months, but also as I was using the Microsoft Visual Studio development environment, which was completely new to me.

The first stage was to create a Visual Studio project, which created some skeleton code, which basically produces a window, which didn't do anything.

4.1.1 The GraphManager Class

This class is the core media control component of the program. This class is what uses the DirectShow classes. This was very difficult to develop, mainly as I had never used DirectX before. I used the MSDN libraries as a source of information and I also purchased a book 'Programming DirectShow for Digital Video and Television' (see reference 7), which proved to be a wealth of information.

After an instance of this class is created, the init method must be called. This initialises all the DirectShow components that are required to operate the media system. I chose this option, as the result of this initialisation must be successful for any further operations to continue. Had the initialisation taken place in the constructor, there would have been too much processing in the constructor and the method to get the result of initialisation would have been untidy.

This init method initialises the filter graph object as well as the video window interface and the media control interface. It also sets for the example which window is to receive window notification messages. These notification messages are a bit like paint () messages in Java. I pass these messages on to the parent window, so it can deal with them. It is important these messages are passed otherwise the video will not be displayed properly. Not only do they inform the parent window of repaint requirements, but also for example window resize events, focus events and media events.

To build the filter graph to control the TV Tuner card, the startTVMode method is called. This first calls destroyGraph which removes all filters from the graph and stops it if it is already built. The startTVMode method then calls buildTVGraph which actually builds the filter graph. The buildTVGraph method is shown in Figure 9.

This method starts by calling the getDeviceByName method. This method takes a name of a component, to search the system for. It also takes a CLISD category identifier. What this method does in this case, is to search the system for all video capture devices. It then iterates through the found capture devices and inspects the name of those devices to see if it matches the name passed in as a parameter.

In this case I actually pass in the name 'Hauppauge' as this is the manufacturer of my TV Tuner card. This returns the Hauppauge WinTV USB Pro Tuner filter to control the TV Tuner card as shown in Figure 3 - The TV Viewing Filter Graph on page 16. In

reality a marketable version of this product would populate a dialog box of all the available Video Capture sources in the system, for the user to select which was their TV Tuner device. This would be necessary as I would not know the name of their device, but also as devices such as web cams would also appear in this list, so you can't just pick the first device you find.

```
683  // Builds a filter graph for the TV Tuner
684  BOOLEAN GraphManager::buildTVGraph(BOOL record)
685  {
686      HRESULT hr;
687      if(!getDeviceByName(&pVideoInputFilter, L"Hauppauge", CLSID_VideoInputDeviceCategory))
688      {
689          hr = E_FAIL;
690          errorHandler(hr,"Unable To Find A TV Tuner Device. Please Ensure It Is Connected","Get TV Device");
691          return FALSE;
692      }
693
694      if(!errorHandler(pGraph->AddFilter(pVideoInputFilter, L"Capture Filter"),
695          "Unable to initialise TV Tuner", "Instantate TV Tuner Filer"))
696      {
697          return FALSE;
698      }
699
700      pGraph->SetDefaultSyncSource();
701
702      if(!errorHandler(pCaptureGraph->RenderStream(&PIN_CATEGORY_PREVIEW, &MEDIATYPE_Video,
703              pVideoInputFilter, 0, 0), "Unable to initialise TV Tuner", "Render TV Stream"))
704      {
705          return FALSE;
706      }
707
708      // Search upstream for a crossbar.
709      pXBar = NULL;
710      hr = pCaptureGraph->FindInterface(&LOOK_UPSTREAM_ONLY, NULL, pVideoInputFilter,
711          IID_IAMCrossbar, (void**)&pXBar);
712      if (SUCCEEDED(hr))
713      {
714          // Try to connect the audio pins.
715          hr = ConnectAudio(TRUE);
716          if (FAILED(hr))
717          {
718              // Search upstream for another crossbar.
719              IBaseFilter *pF = NULL;
720              hr = pXBar->QueryInterface(IID_IBaseFilter, (void**)&pF);
721              if (SUCCEEDED(hr))
722              {
723                  IAMCrossbar *pXBar2 = NULL;
724                  hr = pCaptureGraph->FindInterface(&LOOK_UPSTREAM_ONLY, NULL, pF,
725                      IID_IAMCrossbar, (void**)&pXBar);
726                  pF->Release();
727                  if (SUCCEEDED(hr))
728                  {
729                      // Try to connect the audio pins.
730                      hr = ConnectAudio(TRUE);
731                      pXBar->Release();
732                  }
733              }
734          }
735          pXBar->Release();
736      }
737      return TRUE;
738  }
```

Figure 9 - The buildTVGraph Method

Once the Hauppauge filter has been found, it is added to the filter graph on line 694. Note this line also calls the errorHandler method. This method takes the result of the operation as a parameter, as well as an error message to display and a debugging message. This method then inspects the result of the operation and if it failed it displays an error message to the user as well as storing the debugging message into a log file.

Once the Hauppauge WinTV USB Pro Tuner filter has been added to the filter graph, it is set as the default synchronisation source of the graph, this means it dictates the clock of the filter graph and ensures that the video and audio are kept in sync.

Line 702, then uses the Intelligent Connect feature of DirectShow by calling the RenderStream method. This method actually then basically builds the rest of the filter graph and connects the filters together. This is achieved as each filter specifies what output and input pins it has, which allows DirectShow to know what filters to put where. This saves a lot of work as otherwise each filter would have to be added manually and then each pin connected manually.

The only part of the filter graph which the RenderStream method does not connect is the audio stream. This is because as mentioned previously, the TV tuner card has a physical connection to the line-in port of the sound card. Lines 710 to 736 then actually force this connection to render the audio stream. This happens by searching upstream in the filter graph for the Hauppauge WinTV USB Pro Crossbar filter. This sometimes has to be searched for twice as it doesn't find it on the first attempt. After the filter has been found, its output pins are investigated, looking for an audio output pin to map it to the sound card.

The startRecordMode method, builds the filter graph to record TV programmes. First this method ensures that the TV graph is already built, if not it calls the buildTVGraph method to build it. Once the TV filter graph is built it adds the required filters to enable recording to a Windows Media Video file. This is undertaken in the addRecordingComponent method which is shown in Figure 10.

This method starts by searching for the sound card capture filter. In this case I directly provide the name of my sound card, in reality this method should populate a dialog of the available sound recording sources for the user to select. This is because it searches for audio input devices, which can again include web cams as well as devices such as headsets and even the TV tuner card itself.

Once the audio capture filter is found it is added to the filter graph. Again the result of this operation and the others which follow are passed to the errorHandler method to check the result of the operation, to display an error message if required and also to add to the log file if required.

With the audio capture filter added to the graph, the Advanced Systems Format (ASF) writer filter needs to be added and configured. The ASF writer filter is what encodes the audio and video streams into the Windows Media Video file.

The filename of the output video file is set by creating a name based on the current date and time. The ASF writer is configured using this name via an IFileSinkFilter2 object. The next step is to load a Windows Media profile into the ASF Writer. The profile is the encoding settings for the file, which sets for example the bit rate of the audio and video streams. No cropping of the video source is needed as in fact it only gets delivered from the TV Tuner card at a resolution of 320 x 240 pixels. This is low, which is quite good as it means the video file size is reasonable. No stretching of the video takes place as there is no point; this can be handled reasonably well during playback by this application or any other media application in which it is played.

```
404 // Adds recording capability to the TV Filter graph
405 BOOLEAN GraphManager::addRecordingComponent(void)
406 {
407     CComPtr<IBaseFilter> wmWriter;
408     CComPtr<IBaseFilter> audio;
409     HRESULT hr;
410     // get audio card
411     if(!getDeviceByName(&audio, L"SigmaTel", CLSID_AudioInputDeviceCategory))
412     {
413         hr = E_FAIL;
414         errorHandler(hr,"Unable To Record Files","Get Audio Device Output Pin");
415         return false;
416     }
417
418     // add sound card source to graph
419     hr = pGraph->AddFilter(audio, L"Audio Capture");
420     if(!errorHandler(hr,"Unable To Record Files","Add sound Card Capture To Graph"))
421         return false;
422
423     // get the ASF Writer
424     hr = wmWriter.CoCreateInstance(CLSID_WMAsfWriter);
425     if(!errorHandler(hr,"Unable To Record Files","Get ASF Writer"))
426         return false;
427
428
429     // Add the ASF Writer to the filter graph.
430     hr = pGraph->AddFilter(wmWriter, L"ASF Writer");
431     if(!errorHandler(hr,"Unable To Record Files","Add ASF Writer Filter To Graph"))
432         return false;
433
434
435     // Set the file name.
436     CComQIPtr<IFileSinkFilter2> pSink(wmWriter);
437     if (pSink)
438     {
439         char filename [500];
440         WCHAR wszFileName[500];
441         getFileName(filename);
442         MultiByteToWideChar( CP_ACP, 0, filename,(int)strlen(filename)+1, wszFileName,
443             (int)(sizeof(wszFileName)/sizeof(wszFileName[0])));
444         hr = pSink->SetFileName(wszFileName, NULL);
445         if(!errorHandler(hr,"Unable To Record Files","Set ASF Writer Filter Filename"))
446             return false;
447     }
448
449     IWMProfile *profile;
450     hr = ppProfileManager->LoadProfileByID(WMProfile_V80_BESTVBRVideo, &profile);
451     if(!errorHandler(hr,"Unable To Record Files","Load ASF Writer Profile"))
452         return false;
453
454     CComPtr<IConfigAsfWriter> pConfigAsf;
455     hr = wmWriter->QueryInterface(&pConfigAsf);
456     if(!errorHandler(hr,"Unable To Record Files","Get profile Interface"))
457         return false;
458
459     hr = pConfigAsf->ConfigureFilterUsingProfile(profile);
460     if(!errorHandler(hr,"Unable To Record Files","Set ASF Writer Profile"))
461         return false;
462
463     profile->Release();
464
465     //connect audio capture filter capture pin to audio-in in asf filter
466     IEnumPins *pEnum;
467     IPin *iPin;
468     hr = audio->EnumPins(&pEnum);
469     if(!errorHandler(hr,"Unable To Record Files","Get Audio Capture Pins Enum"))
470         return false;
471
472     BOOL found = false;
473     while(pEnum->Next(1,&iPin, 0) == S_OK)
474     {
```

```
475         PIN_DIRECTION pinDirThis;
476         iPin->QueryDirection(&pinDirThis);
477         if(pinDirThis == PINDIR_OUTPUT)
478         {
479             found = true;
480             break;
481         }
482     }
483 
484     if(!found)
485     {
486         hr = E_FAIL;
487         errorHandler(hr,"Unable To Record Files","Get Audio Device Output Pin");
488         return false;
489     }
490 
491     hr = pGraph->Render(iPin);
492     if(!errorHandler(hr,"Unable To Record Files","Render Audio Device Output Pin"))
493         return false;
494 
495     pEnum->Release();
496     iPin->Release();
497 
498     //connect video capture filter capture pin to video-in in asf filter
499     hr = pVideoInputFilter->EnumPins(&pEnum);
500     if(!errorHandler(hr,"Unable To Record Files","Get Video Capture Pins Enum"))
501         return false;
502 
503     found = false;
504     while(pEnum->Next(1,&iPin, 0) == S_OK)
505     {
506         PIN_DIRECTION pinDirThis;
507         iPin->QueryDirection(&pinDirThis);
508         if(pinDirThis == PINDIR_OUTPUT)
509         {
510             PIN_INFO pinInfo;
511             iPin->QueryPinInfo(&pinInfo);
512             WCHAR cap[] = L"Capture";
513             if(wcscmp(cap,pinInfo.achName) == 0)
514             {
515                 found = true;
516                 break;
517             }
518         }
519     }
520 
521     if(!found)
522     {
523         hr = E_FAIL;
524         errorHandler(hr,"Unable To Record Files","Get Video Device Output Pin");
525         return false;
526     }
527 
528     hr = pGraph->Render(iPin);
529     if(!errorHandler(hr,"Unable To Record Files","Render Video Device Output Pin"))
530         return false;
531 
532     pEnum->Release();
533     iPin->Release();
534     return true;
535 }
```

Figure 10 - The addRecordingComponent Method

The profile used is the “Windows Media 8 - Best Quality based VBR for Broadband”, which as mentioned previously was chosen as it produced the best results in relation to both playback quality and file size.

To configure the ASF Writer to use this profile, after it has been loaded it needs to be mapped to the ASF writer. This is achieved via an IConfigAsfWriter interface. This interface is obtained by querying the ASF writer. Once obtained its ConfigureFilterUsingProfile method is used to set the encoding profile.

Now the filter graph is in the following state: it has the TV graph already built, the audio capture filter is present and also the ASF Writer filer is added and its encoding format is set. The only remaining steps are to connect the capture output pins of the TV Tuner capture filter and the audio capture pins to the video-in and audio-in pins of the ASF writer.

This final connection process is achieved by iterating through the output pins of the audio capture device, looking for the appropriate pin. Once found the Render (pin) method of the filter graph is called. This method uses the DirectShow Intelligent Connect feature to automatically connect the pin. It does this by investigating the filters which are already in the graph and the pins of those filters which are not connected and which are compatible with the pin passed as the parameter. This method connects the audio capture pin to the audio-in pin of the ASF writer.

This process is then repeated, except this time the TV Tuner capture filter's output pins are searched for a capture pin. This pin is then connected to the video-in pin of the ASF writer, again using the Render (pin) method of the filter graph.

4.1.2 The Tuner Class

This class provides the ability for the TV Tuner card to tune to a particular channel. This is achieved using the IAMTVTuner interface provided by DirectShow. This interface can be reached by querying the Hauppauge WinTV USB Pro Tuner filter. It provides the necessary methods for manipulating a TV Tuner card.

This class is initialised by setting the UK country code of 44. In a marketable product this should really have a dialog box for the user to select their country. Once the country code is set, the interface can be queried for the min and max channels. This provides the tuning space for the country.

This class contains methods to change the channel and provides channel up, channel down facilities. It also contains the names of each of the channels available. This is hard coded as the five channels available in the UK. This would need to be changed to make the product marketable in other countries but also to allow for Digital TV Tuner cards which allow for other channels as well.

This class also maintains a list of the stored channels available ready to tune to. This list is loaded on application start up so the user does not have to search through the available frequency range to find the programme they want to watch, such as for example finding BBC1 on channel 39. The list as mentioned is populated on application start-up from a text file. If the file is not found the auto tune feature is activated.

This feature has a dialog box (the Channel Manager window shown in Figure 11) which allows the user to search through all the available channel frequencies, for channels

which contain a signal. The user can then name the channels, change the order of the channels and remove any unwanted channels.

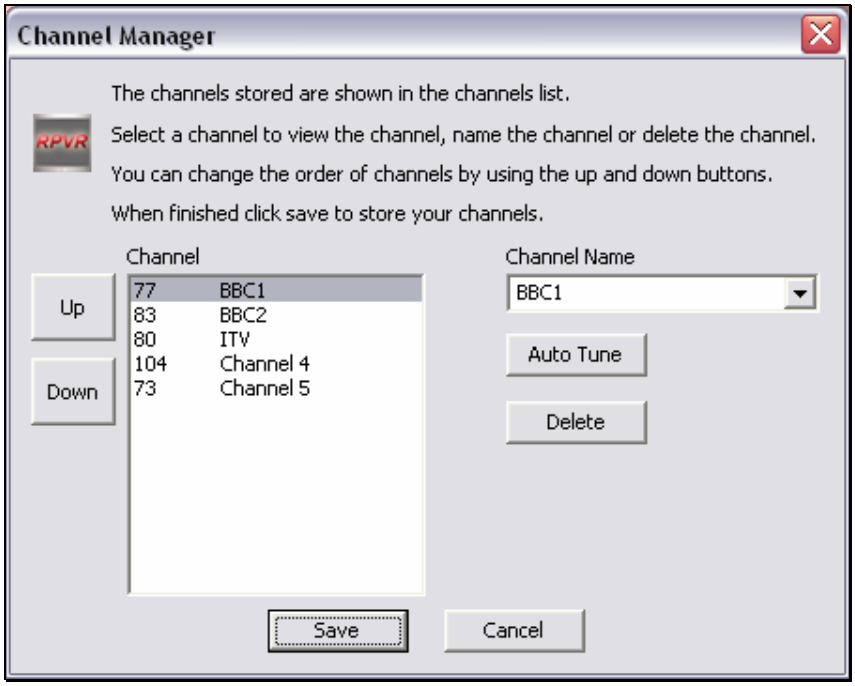

Figure 11 - The Channel Manager Dialog

The auto tune feature populates the channels in the channel box shown in Figure 11 above. The code in Figure 12 shows how the auto tune feature is provided.

```
14  // Auto Tune Function which looks at every available frequency and checks if a signal
15  // is present on that frequency.
16  void Tuner::autoTune(void){
17      long signal;
18      int i;
19
20      vector <int> chans;
21      for(i=chanMin; i<=chanMax; i++)
22      {
23          HRESULT hr = tuner->AutoTune(i,&signal);
24          if (SUCCEEDED(hr)){
25                  if(signal != 0){
26                      hr = tuner->put_Channel(i,-1,-1);
27                      if(SUCCEEDED(hr)){
28                          hr = tuner->SignalPresent(&signal);
29                          if (SUCCEEDED(hr)){
30                              if(signal == 1){
31                                  chans.push_back(i);
32                              }
33                          }
34                      }
35
36                  }
37          }
38      }
39      tuner->StoreAutoTune();
40      updateChannels(chans);
41  }
```

Figure 12 - The autoTune Method

The autoTune method iterates through the available frequency range for the UK and calls the AutoTune method of the IAMTV interface, to set the TV Tuner card to scan for a precise signal on the current iteration frequency. The TV Tuner is then tuned to that frequency using the put_Channel method of the IAMTV interface. Once the channel is loaded, the SignalPresent method retrieves the strength of the signal on the precise frequency obtained from the call to the AutoTune method. If a signal is found, the channel number is stored in a vector and passed to the updateChannels method to add the found channels to the list of channels already stored (if there are any). Also a call is made to the IAMTVTuner interface method, StoreAutoTune which stores the fine-tuning information for all channels in the Windows Registry.

4.1.3 Other Class Development

There were several other classes that were required to be developed to allow the full functionality that the application provides. A quick overview of these classes follows.

The ChannelManager class provides the dialog box shown in Figure 11. It manages the events which occur due to user mouse and keyboard events. This class took a long time to develop and was quite frustrating. This was because I found making dialog boxes in C++ difficult, as it was not intuitive like Java. In fact it took about 20 hours, where as in Java I expect it would have taken about 1 and a lot less code. However I did learn from the experience and did eventually obtain the result I wanted.

The VolumeControl class provides access to the Sound Control of the Windows system. This class is used to modify the volume level from within the program and also to mute the sound at various times during program execution. This for example is used when changing channels to avoid a ‘crunching’ sound. It is also used when the filter graph is started, again to improve the viewing experience.

The Toolbar class provides the toolbar at the top of the screen. Unpredictably this was one of the most difficult tasks, as I was unable to find sufficient documentation to create a toolbar. In fact the only help I could find was using the method I eventually used which is using the CreateToolbarEx method of the Windows Shell API. The problem with this however is that it only allows 16 colour toolbar icon images to be used. Although it provides the required functionality, I am a bit disappointed with the quality of the appearance and I feel it lets the application down, especially as the toolbar is a focal point of the main application window.

The TVTuner.cpp file is the main application entry point. It was originally created by Visual Studio and provided the main application window. This file basically processes the command line arguments and starts the application based on the arguments passed in. It also handles the events caused by user actions such as mouse events but also the events raised by the video window.

The FileSeeker class allows the user to fast forward and rewind a media file. This class acts as a thread, that while the toolbar rewind or fast forward toggle button is depressed or the rewind or fast forward menu item is checked, moves the current media file position. The thread moves the current position of the file forward or backwards 10 seconds twice every second. This allows the user time to see the current position and to

stop the action. I was hoping to implement a timeline facility, however ironically I didn't have the time.

The Timer class is a relatively simple class which acts as a thread, which is called by TVTuner.cpp, when it receives command line arguments to start the application in timer recording mode. The Timer class waits for the required duration (received in the command line arguments) then stops the filter graph and closes the application. This allows timer recording.

The final class is the Thread class. This class was obtained from the Code Project website [8], as was very useful, as it provided a means to create threads for the FileSeeker and Timer classes using Java Thread class like operations. I chose to use this as it allowed me to create threads in a method that I was familiar and comfortable with.

4.2 Timer Scheduling Program Development

Once the TV Tuner application was complete, I breathed a sigh of relief as I was now able to complete the application in my favourite language Java.

The first stage I undertook was to work out how to run a Java application in the System Tray of Windows. I found that the forthcoming Java 1.6 API provides this, but as it is still in a beta format, I didn't want to use it. Instead I found the JDesktop Integration Components (JDIC) Library [9] provided the functionality I required so I decided to use that library instead.

4.2.1 The Scheduler Class

After developing an application entry point, with a system tray icon providing a popup menu to launch to perform various operations, I started development of the part of the application that would provide a GUI to create and modify a timer recording schedule and the functionality of timer recording.

Timer recording was an easy function to implement as I had already written the TV Tuner application to accept command line arguments in the form of "record channel duration". So to enable a recording to take place all that was required was to obtain a runtime object from the JVM to start the TV Tuner application.

The method which performs this task is shown in Figure 13. The startApp method takes a schedule object as a parameter which contains information about the recording such as start time, channel to record and the duration of the recording. The method starts by checking if the TV Tuner application is already running, if it is, it closes it. It then calculates the required recording duration based on the start time specified, the current time and adds an extra minute for safe measure. It then starts the TV Tuner application with the required command line arguments.

The startApp method is contained within a nested class of the Scheduler class called Task which inherits from the TimerTask class. TimerTask objects define the work carried out by Timer objects of the Java API. Timer objects can be used to perform repetitive tasks that need to be carried out at regular intervals. In this case I use a Timer object to call the run method of my nested class Task every 60 seconds.

```
882         /**
883          * Method to launch the TV application in Recording mode.
884          * @param s the schedule information for this recording
885          */
886         private void startApp (Schedule s)
887         {
888             if(TVApp != null)
889             {
890                 try
891                 {
892                     TVApp.exitValue ();
893                 }
894                 catch(Exception e)
895                 {
896                     TVApp.destroy ();
897                 }
898             }
899
900             Runtime r = Runtime.getRuntime ();
901             try
902             {
903                 long end = s.getTime () + (s.getDuration () * MINUTE_MILLS);
904                 long now = System.currentTimeMillis ();
905                 int duration = (int) ((end - now)/ MINUTE_MILLS) + 1;
906
907                 TVApp = r.exec ("TVTuner.exe record " + s.getChannel ()
908                         + " " + duration);
909             }
910             catch (Exception ex)
911             {}
912         }
```

Figure 13 - The startApp Method

The run method of the Task class, looks at the collection of schedule objects, gets the first schedule and checks the start time. It then takes the appropriate action such as call the startApp method, depending on the start time of the schedule object.

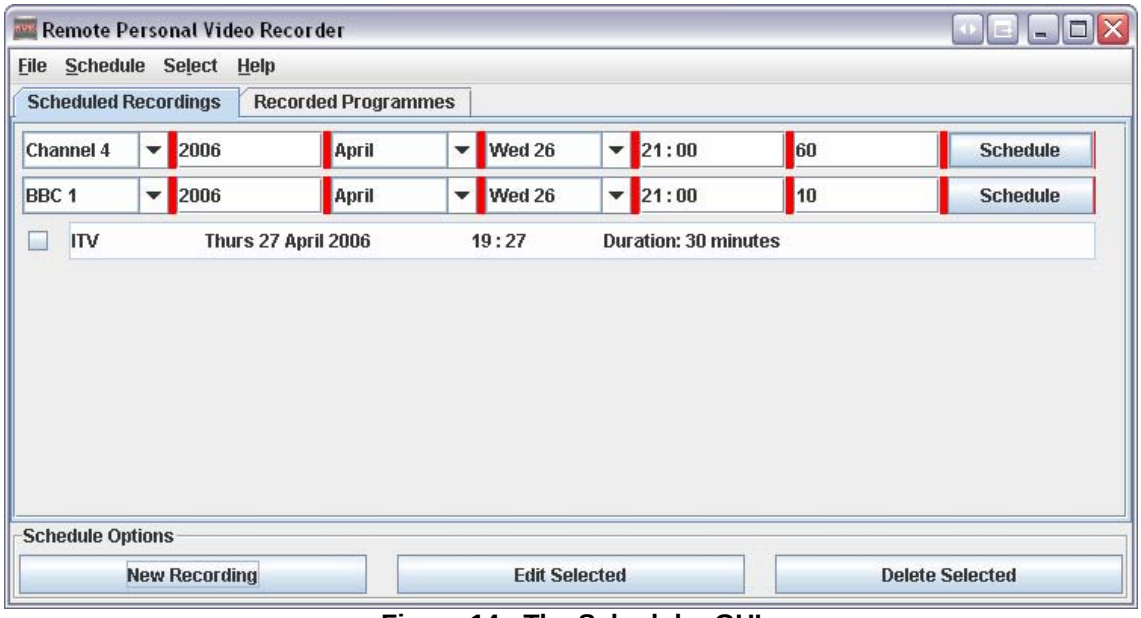

Figure 14 - The Scheduler GUI

The scheduler GUI is shown in Figure 14. It basically has a top menu and a tabbed pane consisting of two tabs. The first tab provides the interface to allow the user to modify the recording schedule. The second tab allows the user to view the video files which have been recorded, to play then, rename them and delete them.

This interface provides an easy to use method of scheduling a new recording by providing drop down selection boxes, which allow setting the various schedule options. The date aspect of the recording schedule is made easier as the drop down box which shows the day of the month automatically updates, when a change is made to the year or month fields.

When a schedule is made, if a recording clash is found, the offending schedules are shown in red as shown in Figure 14.

4.2.2 The RPVRServer Class

The RPVRServer class is the HTTPS server of the application. This is the class which makes scheduling timer recordings over the internet from a web browser or a mobile phone possible.

This class runs its own thread, which runs permanently in the background whilst the application is running in the system tray and starts by creating an SSLServerSocket. The SSL socket listens for connections on a port number specified in the user's preferences.

When the socket receives a connection it reads the request received and the header information provided by the client. I have added a specific header in the mobile phone application which allows the RPVR Server to distinguish mobile phone clients from web browsers.

All the information required for the request is contained within the 'Get' request from the client, this includes their username and password, which was why SSL connections was chosen. I had to devise protocols for the various requests and responses that would be sent between the clients and the server. For example when a web browser client logs in they make this request to the server:

https://users_ip:users_port/login&dologin&username&password

As an SSL connection over HTTPS is being used as only the https://users_ip:users_port part of the request would be visible to any hackers listening in to the communications. The rest of the request is encrypted.

When dealing with web browser clients, web pages are sent to the user. When the client first connects, the web page shown in Figure 15 is sent to the client, as well as the required image files and the RPVR JavaScript file. The page is a frames page which contains inline frames. This allows the requests to be sent without the full request being shown in the address bar of the user's browser. It also allows the RPVR JavaScript file to contain variables and methods that can be shared by all pages that are loaded into the main frame of the page.

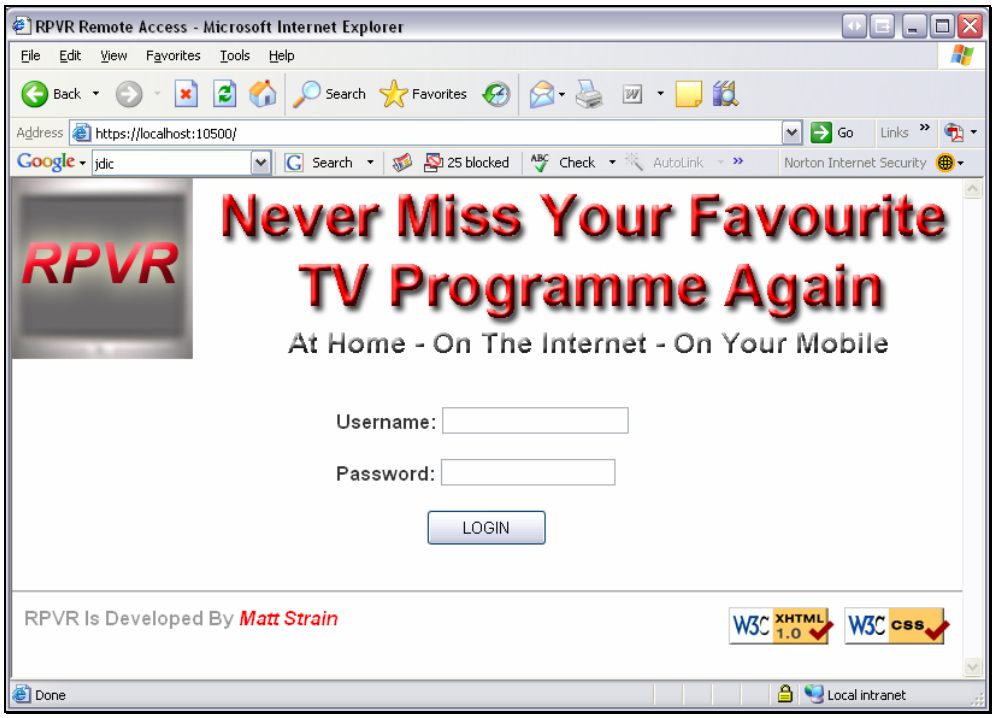

Figure 15 - The Web Browser Client Frames Page

Once the user has successfully logged in, they are shown their timer recording schedule, the page for which is dynamically generated at runtime by the RPVRServer class. This operation is handled by the loadWebSchedule method which is shown in Figure 16.

The method starts by loading an html template page, containing the page elements and an empty table into a string buffer. The schedule is then inserted before the closing'</table>' html tag in the form of html, as table rows.

If there are no recordings currently scheduled, a row is inserted to the table with a cell containing the text "No Recordings Scheduled"; this can be seen on line 452.

However if there are scheduled recordings, each schedule is inserted as a row in the table. As well as the information about the schedule, buttons are provided to edit or delete the schedule item.

When building the html for these buttons, the function calls required for the 'onclick' events in the html is provided by adding the code, to call the necessary function of the RPVR JavaScript file but also dynamically adding the required parameter in the method call. This can be seen in lines 461 to 471. The results of this can be seen in Figure 17.

Web browsers can also edit and add schedules via a web page interface, which is also dynamically loaded, when editing a schedule to set the various form items with the current schedule data. The page to create a new or edit an existing schedule uses another JavaScript file called calendar.js, which provides the functionality to set the date and time correctly in a way which is similar to the actual GUI interface of the main Scheduler application, i.e. it allows drop down selection of channels and dates. It also dynamically populates a day of month drop down list depending on the year and month selected.

```
438         /**
439          * This method dynamically creates the html page which displays
440          * the users recording schedule.
441          */
442         private void loadWebSchedule ()
443         {
444             try
445             {
446                 StringBuffer sb = getFile (new File ("schedule.html"));
447                 Vector <Schedule> recordings = schedule.getSchedule ();
448                 int index = sb.indexOf ("</table>") - 1;
449
450                 if(recordings.size () == 0)
451                 {
452                     sb.insert (index, "<tr><td>No Recordings Scheduled</td>" +
453                             "<td></td><td></td><td></td><td></td></tr>");
454                 }
455                 else
456                 {
457                     String [] channels = new String[] { "BBC 1", "BBC 2", "ITV",
458                     "Channel 4", "Channel 5" };
459                     for(Schedule s : recordings)
460                     {
461                         sb.insert (index,
462                                 "<tr><td>" + channels[s.getChannel () - 1] + "</td>\r\n"
463                                 + "<td>" + s.getDateTime () + "</td>\r\n"
464                                 + "<td>" + s.getDuration () + "</td>\r\n"
465                                 + "<td><input class=\"button\" type=\"button\" id=\"edit"
466                                 + s.getTime () +"\" value=\"Edit\" onclick=\"parent.getEdit('"
467                                 + s.getTime () + "');\" /></td>\r\n"
468                                 + "<td><input class=\"button\" type=\"button\" id=\"delete"
469                                 + s.getTime () +"\" value=\"Delete\" onclick=\"parent.deleteSched('0$"
470                                 + s.getTime () + "$0');\" /></td></tr>\r\n");
471                         index = sb.indexOf ("</table>") - 1;
472                     }
473                 }
474                 sendStringBuffer (sb);
475             }
476             catch(Exception e)
477             {}
478         }
```

Figure 16 - The loadWebSchedule Method

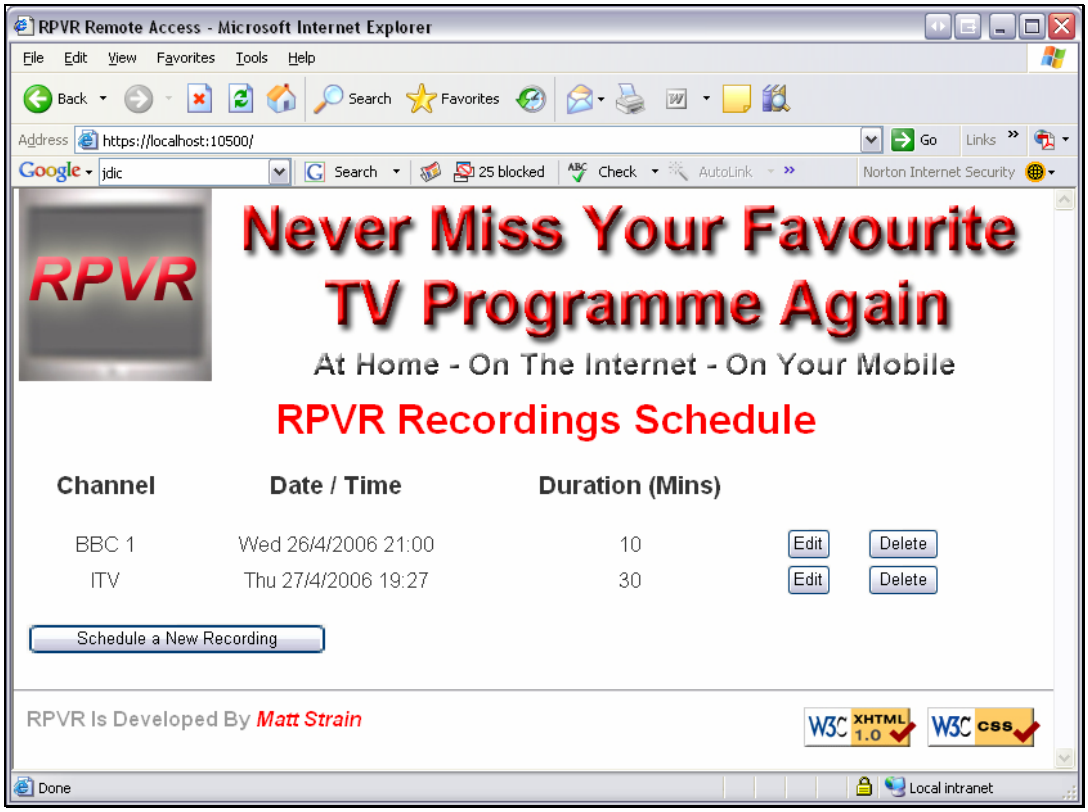

Figure 17 - The Web Schedule

Another important feature of the RPVRServer class is that it ensures web browsers do not perform any caching of the pages. It does this by providing HTTP response headers that contain flags the following flags:

Cache-Control: no-store
Cache-Control: no-cache
Expires: -1

The combination of all three methods is used to cope with different web browser clients, as for example Internet explorer ignores the no-store flag but obeys the no-cache flag.

4.2.3 Other Class Development

The rest of the development for the Scheduler program consisted of developing further Java classes, JavaScript files and also html pages to be delivered via the RPVR Server.

The Preferences class provides an interface to set user options such as the port number which the server socket listens on, the user's username and also their password. To make life easier when using the mobile phone application, I stipulated that the password would be in the form of numeric characters.

The Recordings class provides the interface which allows the user to play a recorded video file, to rename it or to delete it. This interface is added to the recordings tab of the main window.

The Schedule class provides two interfaces, one to edit or create the schedule information and another to view it. Only one of these interfaces is visible at the time and they are added to the schedule tab of the main window.

4.3 Mobile Phone Program Development

To make things easier to create the mobile application, I designed the screens and the command switching between screens using Netbeans, which provides a visual tool for developing mobile applications. This however produces code which in my view is not very readable and considerably over inflated.

I took the code produced from NetBeans and refactored it, so that is easier to read and more efficient. In fact to give you an idea I halved the amount of code required.

Once I had the basic skeleton program, I had to break it. This is because I had made it so that when for example you clicked login, it opened the main menu. What I needed it to do was to connect to the remote computer and if authorised, show the main menu.

When performing tasks such as networking on a mobile phone, a new thread is required to perform the task otherwise the application may become unresponsive. This however gives the impression to the user that nothing is actually happening. To combat this I used a Netbeans library object, called WaitScreen. The WaitScreen object allows a visual display to be displayed to the user, with a cancel option while in the background; a new thread is performing a task. The new thread is managed by the WaitScreen object

in the form of a SimpleCancellableTask. The SimpleCancellableTask object allows an executable function to be set within it.

I used the SimpleCancellableTask object to perform the necessary steps to make a connection to the remote computer, to send a request and receive a response. The steps taken by the SimpleCancellableTask are shown in Figure 18. When a command action in the program requires a connection prior to loading the next display screen, the wait screen is displayed and the code in Figure 18 executes. Prior to this code executing the command action sets a variable to indicate the request type and initialises other variables that are required for the request.

```
553     private SimpleCancellableTask getConnect ()
554     {
555         connectTask = new SimpleCancellableTask ();
556         connectTask.setExecutable (new Executable ()
557         {
558             public void execute () throws Exception
559             {
560                 HttpsConnection conn = null;
561                 DataInputStream in = null;
562                 String message = "";
563                 try
564                 {
565                     conn = (HttpsConnection) Connector.open ("https://" + IP
566                             + ":" + port + "/&" + getRequest ());
567                     conn.setRequestProperty ("Client", "RPVR Mobile Account");
568                     // obtain a DataInputStream from the HttpsConnection
569                     in = new DataInputStream (conn.openInputStream ());
570 
571                     // retrieve the contents of the response from Web server
572                     int ch;
573 
574                     while ((ch = in.read ()) != -1)
575                     {
576                         message = message + (char) ch;
577                     }
578                     processResponse (message);
579                 }
580                 catch (Exception ex)
581                 {
582                     Alert alert = new Alert ("Connection Error",
583                             "Error Unable To Connect To Server: "
584                             +  ex.getMessage () + " " + message,
585                             null, AlertType.ERROR);
586                     alert.setTimeout (Alert.FOREVER);
587                     connectScreen.setNextDisplayable (alert,getLogin ());
588                 }
```

Figure 18 - The SimpleCancellableTask Connection

The new thread starts on line 565, where an HTTPS connection is made using the provided IP address and port number. The request string is added to the HTTPSConnection object using the getRequest method. This method looks at the variable stating the connection type, specified by the calling command action and builds the request as appropriate. A header item is then added which allows the server to identify that this is a mobile phone client, which allows the server to send different responses, that are not in an html format, but simply strings for the mobile application to process.

The connection is actually made on line 569 when an input stream is established with the RPVR Server. The response is then read in from the server and it is processed in the processResponse method. An alert is used to display an error message to the user, should anything go wrong in the connection process.

The processResponse method receives the response string which will either contain "ok&schedule_data" or "error&error_message". If an ok response is received, this method will then set the next display screen depending on the connection action, which was just processed. If an error response is received, the error is displayed to the user using an alert message and they are then redirected back to the screen where they called the connect function from.

The schedule data passed in the ok response method is in the form of a channel number, the recording duration and the date / time of the recording in milliseconds since 1970, all separated by '$' characters. This is then used to construct a collection of Schedule objects, which are then displayed in the view schedule screen which is shown in Figure 19.

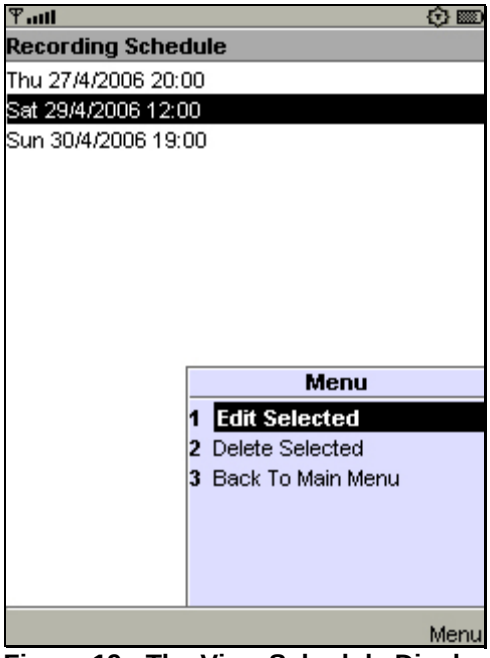

Figure 19 - The View Schedule Display

The View Schedule display shows the various recording times that have been set by the user in a selectable list format. That means each item is selectable. If the user clicks select on an item the edit schedule screen is displayed. If the user clicks menu, the menu shown at the bottom right of Figure 19 is displayed.

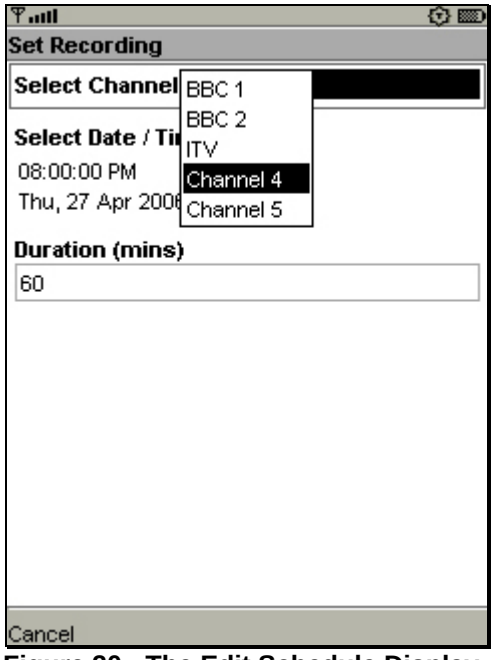

Figure 20 - The Edit Schedule Display

It should be noted however, that due to the J2ME specification, this functionality might not be the same on all devices. This is because the J2ME runtime environment leaves the actual implementation down to the devices operating system. This is because of the variety of different devices available, all with different configurations. For example a phone or PDA with three option buttons, might not display the menu at all. This is because there are only three options in the menu, so it would display the three options as buttons at the bottom of the display. While this makes J2ME development a bit harder, as more consideration has to go into the labelling of buttons, menu items and even the display, in the end it improves the user interaction of the application.

Figure 20 shows the display used to create a

new schedule or to edit an existing schedule. Again the actual final appearance and layout is down the device itself, however in most cases it should appear as shown in Figure 20.

This screen displays a drop down list, which allows the channel to be recorded to be set. It also contains a date field, which allows the user to select the date and time of the recording. The implementation of the date field is left down to the device, but the date field does provide calendar checking in that it will only allow valid dates and times. On some devices the date field may be displayed as shown in Figure 21.

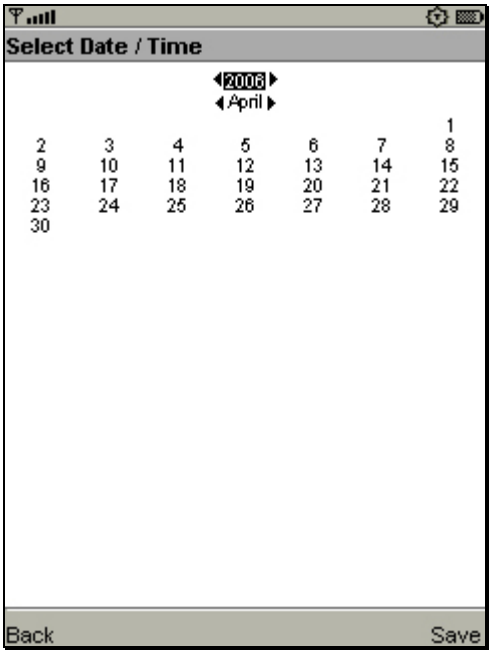

Figure 21 - Setting the Schedule Date

Once all the information is set within the display of Figure 20, the user will connect to the RPVR Server to add the new schedule or edit the existing schedule, depending on what option they had previously selected, to bring them to the edit schedule display screen.

There were few problems whilst developing the mobile phone application. The only problems encountered were getting the internet communications to work. At first I was having problems with the internet connection on my phone. This however was not due to the program itself, but due to the Vodafone internet settings I had. Another problem which caused unnecessary confusion was the emulator that runs on a PC, to simulate the program.

The emulator was unable to make internet connections, so in fact I spent some time trying to debug problems with the program, when running on the emulator, only to discover after running the program on my phone, that the code was fine, it was just a problem with the emulator.

5 RESULTS, CONCLUSIONS AND FUTURE WORK

5.1 Project Implementation Results and Conclusions

Going back to the specific goals set on page 6, all of the targets have been met. Using this as a measure, the project would be deemed as a success. In general I am very satisfied with the final product that has been produced.

The project has required me to use many different Software Engineering techniques and to develop code in many different languages such as C++, Java and also internet technologies such as JavaScript. This has enabled me to combine lots of the skills I have learnt during my degree to produce a complete product.

There are some aspects of the application that I would like to do again to improve the product. The main aspect is the TV Application itself. I now think that I chose the wrong option to keep it as a separate program, with its interface written in C++.

I believe that the user interface could be a lot better and this perhaps could be achieved by writing the application in Java. The use of DirectShow could be limited to an external Dynamic Linked Library file, which could be called using the Java Native Code API. Also since the development of the TV Tuner application, I have found via the Java forums on Sun Microsystems website, a library package, which includes a DirectShow wrapper, for accessing DirectShow components from within Java.

However overall I am pleased that I have achieved what I set out to do and in brief, I have written an application that allows a user to record a TV programme on their computer from anywhere in the world.

5.2 Future Work

There are a number of tasks which need to be completed to make this a fully marketable product. This includes improving the TV Tuner application interface as mentioned above, but also removing a lot of the hard coded, hardware specific implementations I have in place.

These include providing options for the user to select their TV Tuner card and their sound card. Also the country code used to tune the TV card would need to be selected by the user, depending on which country they were in.

The list of channel names would also need to be changed for the application to work in other countries and also with digital television. This could perhaps be linked with an Electronic Programme Guide (EPG) which would actually inform the user of the various programme listings for channels, allowing them to select a program to record, rather than having to select the channel, date, time and duration. I did originally look into this in my early investigations; however getting access to the programme listings databases is quite expensive.

Another feature which could be added is the ability to pause live television programmes, a feature which is offered by devices such as Sky+.

Another drawback to the application as a whole is that it requires the user to have a publicly accessible static IP address. This is not the case with most broadband connections as most users have their IP address change every few days. Also with large numbers of broadband users often having a router or firewall device, it requires them to use port forwarding to the computer that is hosting the RPVR application. This may also run into difficulties as their computer may obtain a different local LAN IP address, each time their computer starts up.

A workaround for this is that RPVR users could actually connect to a central server, which would keep a record of their recording schedule and would allow them to just connect to this site to modify their schedule. Their computer running the RPVR application would then also connect to the central server to retrieve their schedule at regular intervals to know when a timer recording needs to take place.

Another benefit of this is that it would improve both the web browser interface and mobile phone interface as the user would not need to worry about IP addresses and port numbers. It would also allow the browser interface to be continuously updated and improved without the user having to download a new version.

This might also be able to generate extra income by charging a subscription to the service but also advertising space could be sold on the web pages, as well as the opportunity to sell statistical data, such as types of programmes recorded etc. to third parties.

6 REFERENCES

[1] Steve Ranger, 20th December 2005, 'Internet TV Drives Broadband Uptake', ZDNet UK News, viewed 12 January 2006, http://news.zdnet.co.uk/communications/broadband/0,39020342,39242903,00.htm

[2] Unknown Author, June 2004, In-Stat/MDR, Digital TV Coming Home On PC-TV Tuners and Digital Terrestrial Set Top Boxes.

[3] July 2003, Jupiter Research, Jupiter Research / Ipsos-Insight Music Survey.

[4] Ofcom, 2005, The Communications Market Report - Telecommunications, viewed 22 January 2006 http://www.ofcom.org.uk/research/cm/cm05/.

[5] Unknown Author, 2005, 'Mobile TV', Vodafone UK Entertainment, viewed 12 November 2005, http://vodafone-i.co.uk/live/live_tv.html

[6] Michelle Abraham, March 2005, DVD Market Continues to Grow As Recording Units Gain Share, In-Stat, viewed 12 November 2005, http://www.instat.com/press.asp?ID=1274&sku=IN0501925ME

[7] Pesce, Mark D: Programming Microsoft DirectShow for Digital Video and Television, Microsoft Press 2003.

[8] Szymanski, W, 28th Jan 2004, A C++ Thread Class, The Code Project, viewed 14th March 2006, http://www.codeproject.com/threads/ThreadClass.asp

[9] Zhang, George, JDesktop Integration Components, https://jdic.dev.java.net/