IDIAP COMMUNICATION REPORT

MULTIMODAL DATA FLOW CONTROLLER

Danil Korchagin

Idiap-Com-01-2009

NOVEMBER 2009

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

Multimodal Data Flow Controller

Danil Korchagin

Idiap Research Institute,
P.O. Box 592, CH-1920 Martigny, Switzerland
Danil.Korchagin@idiap.ch

Abstract. In this paper, we describe a multimodal data flow controller capable of reading most multichannel sound cards and web cameras, synchronising media streams, being a server to stream captured media over TCP in raw format, being a client to receive media streams over TCP in raw format and using unified interface for online transmission.

Keywords: audio capturing, video capturing, synchronisation, transmission.

1 Introduction

The generic scenario of multimodal processing implies the use of one video capture device and multiple audio capture devices in a single room. The present software implementation concerns the simplification of using multiple capture devices for following multimodal analysis. This implies capturing, synchronisation and transmission via network of raw media streams. The graphical user interface (GUI) of corresponding software is shown on figure 1. It has video (left side) and audio (right side) related controls with preview windows. The software was developed and compiled under Microsoft Visual Studio 2008 for Windows platform.

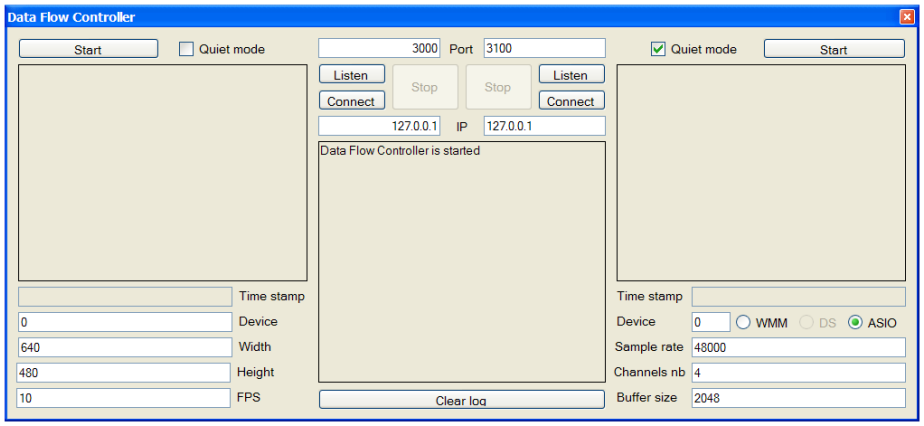

Fig. 1. Data flow controller GUI.

2 Audio Capturing

Real-time mono/stereo audio capturing is relatively easy task due wide standardisation and compatibility of available consumer sound cards and media libraries. Nevertheless in case of multiple channel use (4, 8, or even 26) we are restricted to the choice of professional sound cards. Official support for most professional sound cards is normally restricted to Microsoft Windows and Mac OS X. In Windows series preceding Widows Vista high latency audio mixing kernels (KMixer) were used. In Windows Vista, KMixer has been removed and replaced by a new WaveRT port driver. WaveRT is also known to be high latency port, which cannot provide synchronised audio from multiple devices and does not support external clocks. In addition, none of KMixer/WaveRT provides direct access to more than 2 input channels. To access more than 2 channels ASIO (Audio Stream Input/Output) protocol is the best choice for Windows platform.

ASIO [1] is a computer soundcard driver protocol for digital audio specified by Steinberg, providing a low-latency and high fidelity interface between a software application and a computer's sound card. ASIO bypasses the normal audio path from the user application through layers of intermediary Windows operating system software, so that the application connects directly to the soundcard hardware. Each layer that is bypassed means a reduction in latency, the delay between input signals from the soundcard being available to the application. In this way ASIO offers a relatively simple way of accessing multiple audio inputs and outputs independently. Its main strength lies in its method of bypassing the inherently high latency of Windows audio path, allowing direct, high speed communication with audio hardware.

ASIO interface support is normally restricted to Microsoft Windows, since other operating systems (e.g. Mac OS X or Linux) do not have such mixer latency problems (Core Audio and ALSA). As of 2007 there is also an experimental ASIO driver for Wine, a Windows layer for Linux. All ASIO interfaces are written in pure C.

Another known issue – Microsoft Windows, Linux and Mac OS X are based on multitasking. A multitasking works by running lots of separate programs or tasks in turns, each one consuming a share of the available CPU (processor) and I/O (Input/Output) cycles. To maintain a continuous audio stream, small amounts of system RAM (buffers) are used to temporarily store a chunk of audio at a time. If the buffers are too small and the data runs out before Windows can empty them, we get a glitch in the audio stream that sounds like a click or pop. If the buffers are far too small, these glitches occur more often, firstly giving rise to occasional crackles and eventually to almost continuous interruptions that sound like distortion as the audio starts to break up regularly. Making the buffers a lot bigger immediately solves the vast majority of problems with clicks and pops, but has an unfortunate side effect – the big latency. A typical ASIO buffer size of 256 samples at 48 kHz results in latency of 5.3 ms (43.7 ms for the buffer size of 2048 samples).

3 Video Capturing

One of the ways to implement video capturing is based on Microsoft DirectShow [2], an architecture for streaming media on the Windows platform. DirectShow supports capture from digital and analogue devices based on the Windows Driver Model (WDM) or Video for Windows. It automatically detects and uses video acceleration hardware when available, but also supports systems without acceleration hardware.

DirectShow is based on the Component Object Model (COM) and designed for C++. Microsoft does not provide a managed API for DirectShow. DirectShow simplifies capturing tasks, format conversion and media playback. At the same time, it provides access to the underlying stream control architecture for applications that require custom solutions.

DirectShow is known to be more complex architecture than the standard Windows AVICap class implemented in AVICAP32 dynamic library (Windows platform), which provides applications with a message-based interface to access video acquisition hardware and to control the process of streaming video capture to a disk.

AVICAP32 exports capCreateCaptureWindow function which is used for creating AVICap window. It encapsulates standard AVICap window and also gives independent interface to WDM drivers for USB devices.

The initialisation of most of the cameras is relatively easy and requires few lines of code in C#:

```
// Setup a capture window
mCapHwnd = capCreateCaptureWindowA("WebCap", 0, 0, 0,
           mWidth, mHeight,
           this.Handle.ToInt32(), mDeviceID);

// Connect to the capture device
SendMessage(mCapHwnd, WM_CAP_CONNECT, 0, 0);
SendMessage(mCapHwnd, WM_CAP_SET_PREVIEW, 0, 0);
```

The grabbing new frame and copying it to the clipboard requires two extra line of code:

```
SendMessage(mCapHwnd, WM_CAP_GET_FRAME, 0, 0);
SendMessage(mCapHwnd, WM_CAP_COPY, 0, 0);
```

Disconnection from the video source is performed via corresponding message:

```
SendMessage(mCapHwnd, WM_CAP_DISCONNECT, 0, 0);
```

The described interface can be used in C++ as well with implications of the C++ rules. The frame rate can be user driven, timer driven or device driven.

The data flow controller is based on AVICAP32 dynamic library, though for low-latency streaming and high fps we recommend to use DirectShow as more appropriate architecture.

4 Synchronisation

Audio/video synchronisation is essential not only to reproduce the audio/video streams in remote location, but also for joint audio/video analysis algorithms, results interpretation and composition. In a professional setup, one might expect to be synchronised via a common clock or similar [3]. Consumer level devices, however, do not normally provide such capabilities. The only exception is usage of daisy FireWire chain, though it restricts all devices to be FireWire compatible and all captured streams to be within FireWire bandwidth. In the data flow controller the synchronisation is achieved by time-stamping all streams inside software main process MasterController, responsible for inter-process synchronisation and interactions. We use the 64-bit timestamps that represent the numbers of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

Inter-process interactions are shown on figure 2. The process MasterController (C#) is responsible for all interactions including asynchronous data transmission. The process Microphone (C#) and the process MicArray (C/C++) are KMixer and ASIO implementations for capturing audio streams. The process WebCam (C#) is responsible for capturing images from web camera. The inter-process dataflow is event driven and thus does not restrict us from use of different programming languages within multimodal data flow controller.

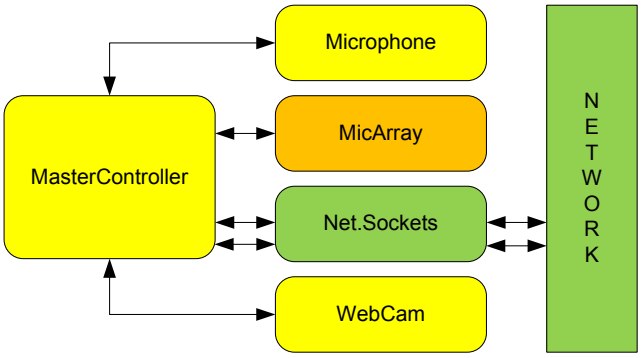

Fig. 2. Inter-process interactions.

In case multimodal capturing is done on different computers additional synchronisation of corresponding computers should be done, e.g. via Network Time Protocol [4]. The Network Time Protocol (NTP) is a protocol for synchronising the clocks of computer systems over packet-switched, variable-latency data networks. NTP uses UDP on port 123 as its transport layer. It is designed particularly to resist the effects of variable latency by using a jitter buffer. NTPv4 can usually maintain time within 10 milliseconds (1/100 s) over the public Internet, and can achieve accuracies of 200 microseconds (1/5000 s) or better in local area networks under ideal conditions.

5 Transmission

The transmission between different computers is done via stream sockets using TCP protocol and based on InterNetwork address family. Each block of raw data is prefixed by the header with additional information:

- header_size – size of the header, 22 for current version, though can be dynamic in the future, if required;
- flags – extra information concerning data (0x1 – end of data/utterance, 0x2 – big endian);
- time_stamp – 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC);
- dim1_size – width of the image or length of the signal;
- dim2_size – height of the image or number of channels;
- dim3_size – bytes per each sample (3 for 24bit images, 2 for 16bit audio).

The corresponding C++ structure:

```
struct PacketHeader
{
    unsigned char header_size;
    unsigned char flags;
    long long time_stamp;
    unsigned int dim1_size;
    unsigned int dim2_size;
    unsigned int dim3_size;
}
```

The special mode without header transmission is also available and can be activated via specifying the command line argument `/rawstream` (the command line argument `/store` enables dumping all streams to the files in raw format). The required bandwidth for different types of supported data is shown in the table 1.

Table 1. Required bandwidth for different types of data.

Type	Resolution	Bandwidth
Audio	Mono, 16bit, 16 kHz	0.24 Mbps
Audio	Stereo, 16bit, 32 kHz	0.98 Mbps
Audio	4 channels, 16bit, 48 kHz	2.93 Mbps
Audio	8 channels, 16bit, 96 kHz	11.7 Mbps
Audio	24 channels, 16bit, 192 kHz	70.3 Mbps
Video	320x240, 24bit, 1 fps	1.76 Mbps
Video	512x384, 24bit, 3 fps	13.5 Mbps
Video	640x480, 24bit, 5 fps	35.2 Mbps
Video	960x540, 24bit, 15 fps	178 Mbps
Video	1920x1080, 24bit, 30 fps	1.42 Gbps

6 Performance

The performance for capturing and transmission within the same computer (Intel Core 2 Quad 2 GHz) or via direct Ethernet cable is shown in the table 2.

Table 2. CPU load during capturing and transmission.

Type	Resolution	CPU usage
Audio	Mono, 16bit, 16 kHz	1%
Audio	Stereo, 16bit, 32 kHz	2%
Audio	4 channels, 16bit, 48 kHz	6%
Video	320x240, 24bit, 1 fps	2%
Video	512x384, 24bit, 3 fps	5%
Video	640x480, 24bit, 5 fps	8%

There is also strong dependency between packet/buffer size, latency and CPU load. Smaller packets have lower latency and higher CPU load. Bigger packets have higher latency and lower CPU load. In our measurements we were using packetisation per video frame and per 128 ms audio window at 16 kHz (64 ms at 32 kHz, 43 ms at 48 kHz).

7 Conclusion

In this paper, we have described a multimodal data flow controller capable of reading most multichannel ASIO-compatible sound cards and web cameras, synchronising media streams, being a server/client to stream/receive captured media streams over TCP in raw formats and using unified interface for online transmission. The performance shows acceptable CPU load for SD resolution raw video stream and CD quality multichannel raw audio streams, which is enough for research purpose. For industrial purpose we strongly recommend to use more appropriate industrial standards, which have better performance, though more difficult to use.

References

1. Steinberg Third-Party Developer, http://www.steinberg.net/en/company/3rd_party_developer.html
2. Microsoft DirectShow, http://msdn.microsoft.com/en-us/library/dd375454(VS.85).aspx
3. Verrier, Jean-Marc: Audio Boards and Video Synchronisation. AES UK 14th Conference: Audio - The Second Century (1999)
4. Network Time Protocol, http://en.wikipedia.org/wiki/Network_Time_Protocol