The Java Media Framework Extension Layer and its applications to biosignal acquisition

Conference Paper · January 2010

DOI: 10.1515/BMT.2010.674

4 authors, including:

Olaf Christ
University Medical Center Freiburg
13 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Ulrich G. Hofmann
University Medical Center Freiburg
235 PUBLICATIONS **1,393** CITATIONS
SEE PROFILE

JMFEL
The Java Media Framework Extension Layer and its applications to biosignal acquisition

O. Christ[1], Sebastian Otto[2], Gregor Radzimski[3] and U.G. Hofmann[4]
[1]Grad-School for Computing in Medicine & Life Sciences, University of Luebeck
[2,3,4]Institute for Signal Processing, University of Luebeck, Luebeck, Germany

Correspondence to: christ@isip.uni-luebeck.de

Abstract

Due to its mainly commercial background, multimedia technology is not too popular in academic science and research. Here, we show how we have benefited from extending an existing multimedia framework – the Java Media Framework – for a sophisticated multimodal data acquisition application. We discuss some important problems commonly faced during the development of data acquisition systems integrating more than just electrophysiological recordings. Using our extension layer we provide an example application for recording and analyzing multichannel EEG, ECG and EOG in sync with a video stream of the test subject. The solution is demonstrated in a psychophysical research project, corroborating the power of multimedia methods in basic research.

1 Introduction

Today's research questions in experimental biosignal processing ask for a more efficient and better use of resources, especially in measurement setups. At the same time a strong desire for non-proprietary measurement devices exist, which can be performed with general purpose computers running general purpose operating systems. Although powerful data acquisition frameworks already exist, nontrivial problems such as synchronization of several media types are very common. In this work we approach this multimodality problem by extending an open source multimedia framework to perform bio signal acquisition, thus effectively reducing the original complexity to writing a multimedia application. At the time this study was started, only the *Java Media Framework (JMF)* was a feasible option. However, the key concepts apply to any other multimedia framework. An extension layer helps speeding up the development by encapsulating important framework parts that are useful but often hard to use at the same time. This is particularly true for the powerful but often difficult to use *JMF*.

2 Developing an application with JMFEL

Like most multimedia APIs, the *JMF* uses key components, such as *Datasources* representing actual physical sources of data, *Players* for playing the data like an audio player does, *Plugins* for processing data, *Datasinks* to represent endpoints such as a file on the hard disk or entries in a *Database Management System (DBMS)* and *Renderers* used by *Players* to appropriately present the processed data. Figure 1 shows a data path commonly seen in video and audio processing from one *Datasource* to another *Datasource* or *Datasink*. The source on the left contains an audio and a video track. The input *Datasource* is demultiplexed and the streams are further processed by plugins before they are rendered by their respective renderer and then multiplexed back into on single output stream, which is fed into another *Datasink* or *Datasource*. This second *Datasource* could be used as an input for further processing.

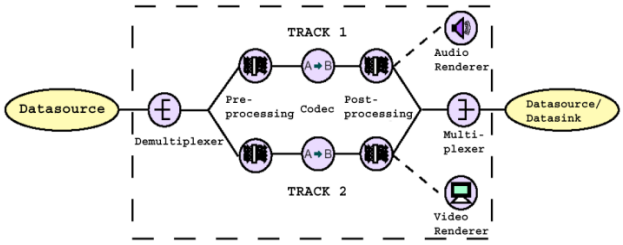

Fig 1: *Data path from one source to another. Data processing happens between two sources or a source and a sink. (Image adapted from [7])*

Every time we are working with separate data streams, keeping them in sync becomes an important issue. Complex real world applications commonly have several concurrent paths from a *Datasource* to a *Datasink* - all of them needing synchronization. The JMF provides different synchronization mechanisms. One is to use a *Player* for each stream and set one of these *Players* as a master and all the other players as slaves. In practice, this method is only feasible for a small number of Players and not very accurate. By far the most accurate method is the one used by JMFEL's *DataSourceMerger* class. Internally, this class uses *JMF's RawSyncBufferMux* multiplexer, which enables synchronization by altering the "cname" (Canonical end-point identifier SDES item. (see: RFC[6] 3550 Section 6.5.1))

The *RawSyncBufferMux* multiplexer uses the track, with the highest sample rate as a master track to synchronize all other media tracks to it. Therefore, a video track will always be synchronized to its audio track, but the audio track will never be synchronized to video track.

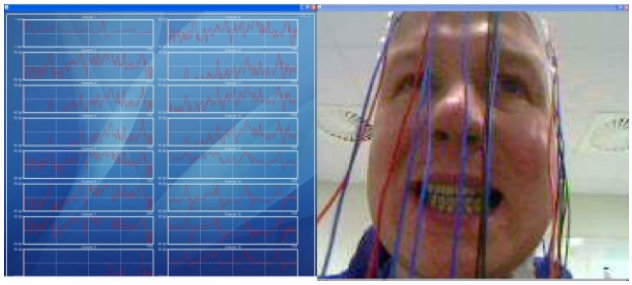

Fig 2: *Muscle artefacts in the EEG are in sync with the video*

Figure 2 shows a simple experiment demonstrating the accuracy and effectiveness of the *RawSyncBufferMux* method. The data path for this experiment is the multiplexing part shown in fig 1.

The code involved to use the *DataSourceMerger* class is straightforward:

```
Vector datasources = new Vector();
datasources.add(dsEEG_1);
datasources.add(dsVideo_2);
mergingDataSourcePlayer.setDataSource(DataSourceMerger.mergeDataSources(datasources));
```

In the following we present a simple application showing the anatomy of a typical *JMFEL* application. Access to a *g.USB Amp* [1], our biosignal amplifier of choice, is given by our g.USBAmp *JAVA API* (see fig. 3). We call the *Datasource* required by the frame *GtecCaptureDevice* and use it to control the amplifier and to acquire EEG data. The g.USBAmp *JAVA API* can of course also be used without the *JMFEL*.

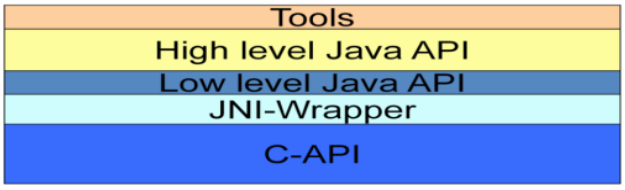

Fig 3: *The layered structure of our g.USBAmp Java API*

In this example we could just use a standard player and a data sink to write the data to the hard disk. To make our application a bit more interesting, we use custom components to present and process data. Our new example application will record EEG, EOG, ECG and a video of the subject. All these signals are acquired by the *g.USB Amp* and the *GtecCaptureDevice*. The video feed is provided by a webcam (see fig. 5).

We use frame based media, meaning that the data stream is split into consecutive chunks or frames of data each having a sequence number and a timestamp.

For EEG we want to implement some simple electrode mapping. Each data frame is organized as a matrix with one row per channel, thus we can simply subtract rows and write the result back to the original matrix effectively implementing online EEG analysis. Mapping between channel numbers and symbols commonly used in neuroscience is done by the *ElectrodeAssignment JMFEL* utility class (see fig. 4).

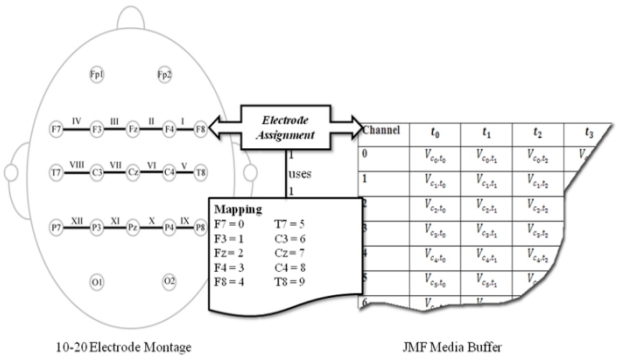

Fig 4: *Each roman numeral represents one bipolar derivation. E.g. IV represents the bipolar derivation between F7 and F3. The ElectrodeAssignment class facilitates a mapping between electrode symbols and channel numbers.*

The algorithm implementing the derivation scheme shown in figure 4 looks like this:

```
ElectrodeAssignment electrodeToChannelAssignment;
double[][] matrix = arrayToMatrix(getInData());
for (int p = 0; t < getTransferBufferSize(); t++) {
    bipolar_I  =  matrix [electrodeToChannelAssignment.getF4()][t] - matrix
[electrodeToChannelAssignment.getF8()][t];
     bipolar_II  =  matrix [electrodeToChannelAssignment.getFz()][t] - matrix
[electrodeToChannelAssignment.getF4()][t];
    bipolar_II  =  matrix [electrodeToChannelAssignment.getF3()][t] - matrix
[electrodeToChannelAssignment.getFz()][t];
    bipolar_IV =  matrix [electrodeToChannelAssignment.getF7()][t] - matrix
[electrodeToChannelAssignment.getF3()][t];
}
```

Fig 5: *Image A shows the test subject lying on a mattress in the lab. Image B shows the subject at the end of the recording session after 1:24:44 hours.*

ECG is another physiological measurement we integrated into our multimodal recording system. Recording is done with standard Ag/AgCl ring electrodes using the back of the hand for the pulse/ECG, FCz as the reference and AFz as GND. For EEG we use positions Fp1, Fp2, Fpz, C3, C4, O1, O2, Oz, F3, F4 and Fz. For EOG one electrode is placed left of the left eye, right of the right eye and below each eye (see fig 5, image B).

Besides displaying the heart signal(see fig. 6), its rate is calculated by a plugin implementing a simple adaptive thresholding algorithm, which is known to work well for healthy subjects [5].

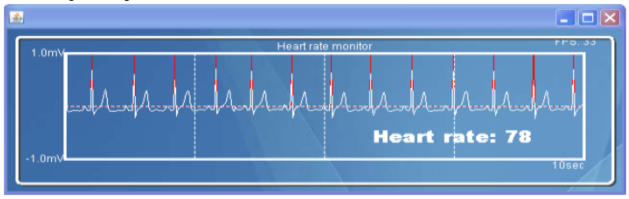

Fig 6: *The heart rate plugin uses a 10 sec ring buffer to calculate the heart rate. It also highlights the R parts [5].*

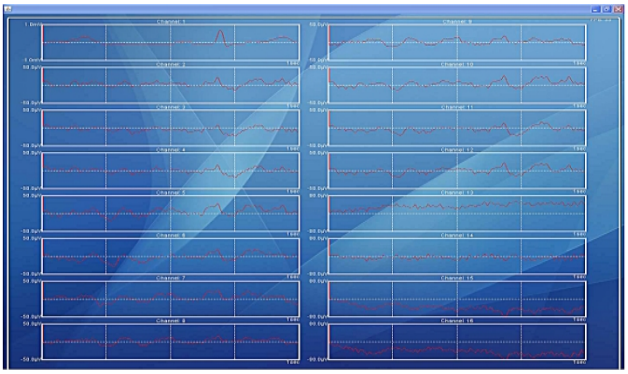

Fig 7: *All 16 channels, are monitored during the recording* session.

```
double maximum = 0.0;
 for (int i = 0; i < bufferSize; i++) {
                maximum = Math.max(maximum,signalArray[i]);
}
for (int i = 0; i < bufferSize; i++) {
double value = signalArray[i]-maximum/2.0;
if(value>0){
signalArray[i]=getDoubleDataBufferContainr().getDataBuffer(getECGChannel()).getMaxVol
tage();
                withinPeak = true;
                }else if((value<=0)){
                                signalArray[i] = 0.0;
                                withinPeak = false;
                }
      if(lastPeakState&&!withinPeak){ peakCounter++;}
        lastPeakState = withinPeak;
}
 heartRate = peakCounter*6;
// This is the complete algorithm to extract and count R parts within a 10 second
ECG signal
```

Our recording application features a live video, the calculation of the heart rate and a 16 channel display of EEG, ECG and EOG (see fig. 7).

All we need now is to merge the video and the bio signal data streams using *JMFEL's DataSourceMerger* class to get a single synchronized stream. This stream might then be fed into a *Datasink* to write the stream into a file on the hard disk or an *RTPDatasink* to transmit the stream over a network e.g. to a handheld device.

Our recording application is mostly put together using *JMFEL's* core and its many support classes. Only the bipolar derivation scheme had to be implemented. The 16 channel display could be used as well in other applications with up to 16 channels.

In general, all *JMFEL* applications share the same simple anatomy. The developer is mostly concerned with laying out data paths and "connecting the dots". Hence, 310 lines of non-trivial *JMF* code will boil down to just 13 lines of quite trivial and easy to understand *JMFEL* code.

```
public static void main(String [] args) {
DataSource dsFile  = null;
SignalDataSource aFileDataSource;
ProcessorPlayer aPlayer;
aFileDataSource = new SignalDataSource();
aFileDataSource.setMediaLocator(new
MediaLocator("file:\\" c:\\testaudio.wav");
aFileDataSource.init();
dsFile = aFileDataSource.getDataSource();
aPlayer = new ProcessorPlayer(dsFile);
aPlayer.setVisible(true);
aPlayer.start();
}
```

3 AdRacer Experiment

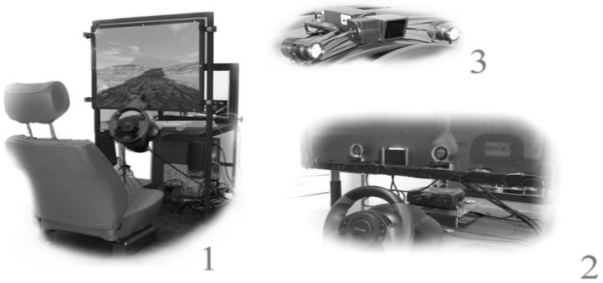

Fig. 8: *Driving Simulator running the AdRacer research system; 2: Eye Tracker camera and infrared lights above the steering wheel; 3: The Eye tracker.*

The original goal of this project was to investigate the emotional, behavioural and cognitive effects of video game violence on brand memory. Results have been published elsewhere [2][8]. With the help of the JMFEL API, we have developed a robust application to acquire and synchronize data streams coming from our driving simulation, the AdRacer(see fig. 8) research system[2], a commercial eye tracking system [3] and our EEG g.USBAmp amplifier. While the application received the eye tracking and the AdRacer research system data via a LAN connection, the EEG data was acquired on the main PC as described earlier. Instead of writing the data stream to a file on the hard disk, we used the JMFEL Database *Datasink* to write the data into an MSSQL Database (see fig. 9).

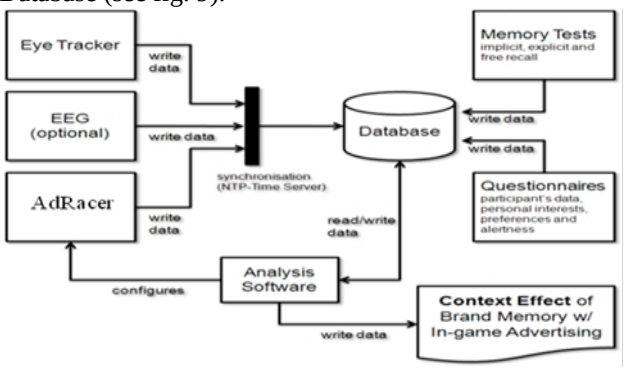

Fig 9*: Our JMFEL application ensures synchronization of different data sources and writes the merged and processed data into a DBMS.*

Because of our heterogeneous setup, we had to use an NTP-Time Server to synchronize all PCs. Although this mechanism is not as precise as the most precise method described in RFC 3550 (see previous chapter), careful offline data analysis showed that the accuracy was sufficient.

To test the eye tracking system and validate the correctness of the gaze positions in our recordings we have designed a simple "passive setup"(see fig 9). Here, the subject is not driving and keeping his/her head still while watching scenes from the game. This provides us with a good baseline for the actual study, because under those controlled conditions we can tell the subject to follow a specific object with his/her eyes and observe the crosshair following the target. This is not possible during the actual experiment.

Fig 9: *The passive setup allows us to check the eye tracking system under controlled conditions, because the test subject is completely passive and just watching scenes from the Adracer research application. EEG is recorded from positions C3, Cz and C4.*

The currently tracked gaze position is shown on the control screen right next to the centre screen. (see fig 10)

Fig 10: *The centre screen is where the test subject is looking at. The control screen on the right displays a crosshair at the current gaze position.*

Additionally, we are simultaneously recording an EEG at positions C3, Cz and C4(see fig. 9).

4 Discussion

By employing our *JMFEL* framework, we successfully cut development time down. This corroborates our belief that extending a multimedia framework, and then working with the extension layer, relieves us from problems usually arising from "low level" programming. The *JMF* has not been updated for years and its future is completely uncertain. It is still available for download from the *Sun Developer Network (SDN)* website. However, if Oracle or Sun decides to put an end to the *JMF*, there are already very good alternatives like e.g. the very active Open Source Project *Gstreamer*, which also provides Java support[10][11]

4 References

[1] Guger Technologies, http://www.gtec.at/

[2] Dec 2008 A. Melzer, B.J. Bushman and U.G. Hofmann: "When Items Become Victims: Brand Memory in Violent and Nonviolent Games" *ICEC 2008 - 7th International Conference on Entertainment Computing/*, Lecture Notes in Computer Science - Entertainment Computing:, 2008, Pittsburgh *Springer*, 2008, Pittsburgh

[3] SensoMotoric Instruments GmbH, Insight Eye Tracker, http://www.smivision.com

[4] JMFEL The Java Media Framework Extension Layer, Olaf Christ 2008 Diploma Thesis(ISIP, University Luebeck)

[5] Textbook of Medical Physiology Eleventh Edition, Guyton & Hall p. 124

[6] http://www.faqs.org/rfcs/

[7] Java Media Framework API guide (1999), p.32

[8] Anderson, C.A., Bushman, B.J.: Effects of violent video games on aggressive behavior, aggressive cognition, aggressive affect, physiological arousal, and prosocial behavior: a meta-analytic review of the scientific literature. Psychological Science 12(5), 353–359 (2001)

[9] A SQL Server based data collection, maintenance and analysis tool for multimodal data, Gregor Radzimski 2008, Diploma Thesis (ISIP, University Luebeck)

[10] http://www.gstreamer.net/

[11] http://code.google.com/p/gstreamer-java/