USAGE OF THE GSTREAMER FRAMEWORK FOR THE GENERATION, ANALYSIS, PROCESSING AND VISUALIZATION OF SONAR SIGNALS

KRZYSZTOF CZARNECKI, DORIAN DĄBROWSKI

Gdansk University of Technology
Faculty of Electronics, Telecommunications and Informatics
Department of Marine Electronic Systems
Narutowicza 11/12, 80-233 Gdańsk, Poland
krzycz@eti.pg.gda.pl

In this paper, the usage of the GStreamer framework in applications of classical digital signal processing is discussed. Especially, its adaptation regarding sonar technique is presented here. Signal generation, analysis, processing, and visualization are implemented as GStreamer plugins. The new plugins and the structure of data transmitted through the GStreamer pipeline are both briefly discussed. The introduced plugins are published as free software under the GROJ project.

INTRODUCTION

The need for an open software platform given over to application development is evident. Moreover, technical support, the availability of documentation, and a lack of dependence on the hardware and operating system is desirable here. Essentially, the benefit of using such a platform is source code portability, software standardization, and finally, cost reduction in the development process. These requirements are fulfilled by The GStreamer framework (GSF) [1] and The GNU Radio Toolkit [2,3], amongst others. The considerations in this paper are focused on the GSF and its use in the analysis and processing of sonar chirp signals.

GStreamer is an open source multiplatform framework originally dedicated to the streaming and processing of multimedia data. It is known to work on Linux, Solaris, Microsoft Windows, FreeBSD, Mac OS X, and many other popular operating systems. It is also widely employed as an engine for many multimedia applications, such as media players and capture encoders, for example: Banshee, Kaffeine, Rhythmbox, Sound Juicer, Totem and so on. The framework is published on the GNU Lesser General Public License and supported by freedesktop.org. It has application programming interfaces (API) or bindings for some programming languages: C/C++, Python, Perl, Ruby, Vala, etc. Each GStreamer application consists of plugins, that can load, process and save streams of information. Plugins are joined and packed into virtual

Fig. 1. The GStreamer logo.

pipelines. In general, the pipeline provides an interface between the plugin system and graphical user interface (GUI) [4]. Today, thousands GStreamer plugins are available. They all contain substantial functionality in the multimedia application field [5–8].

The processing of multimedia streams is related to the classical digital signal processing (DSP). Both can be organized as pipelines consists in many modules. Each of the modules implements a dedicated restricted task such as:

- in media processing: converting, coding, mixing, etc.
- in DSP: resampling, filtering, modulation, etc.

It seems that the GStreamer framework can be also successfully used in DSP. There are many advantages of GSF, amongst others:

- A unified application structure.
- A hardware-independent interface for specific devices which can be provided by the selected specific GStreamer plugins.
- An once developed module can be multiple use by different users and applications.
- A division of the whole processing schema into smaller modules, which simplifies the maintenance and management of a project.

In the paper, an exemplary usage of GSF in classical applications of DSP – especially within a sonar technique – is presented. A similar field of research was considered in [8], where the authors used GSF for the development of a digital radio system. The GSF architecture allows us to develop new plugins which are capable of using specialized hardware and can maintain high algorithm performance such as signal processors or embedded digital filters. At this point, the GStreamer plugin plays an interface role between the hardware and the high-level application layer. Futhermore, provide of work of a DSP system in real-time is also possible. This is particularly important in professional applications of sonar signal processing. GSF has inbuilt mechanisms which support real-time transmission. Then again, GSF was originally given over to multimedia processing. There are implemented plugins that can be used in the development of operator stations situated at the end of a processing chain, for instance, in on-line visualizations [5] and when listening to received signals from a water column.

This paper is organized as follows. A short description of the conception, architecture, and, most importantly, the GStreamer parts and mechanisms is presented in Section 1. Then in Section 2, the GROJ project is introduced. Consideration of sonar signal generation by GSF is introduced in Section 3. Then, respectively, the example of a signal analysis in the joint time-frequency domain and then an online visualization of a signal or its various representations are all discussed in Sections 4 and 5. Finally in Section 6, the following are presented: a signal processing approach and a simple example which includes in the ability to listen to a received high-frequency signal.

1. GSTREAMER

The conception of the development of GStreamer applications derived from the GObject model [9,10]. The mechanism of signal and object properties is used. All objects can be queried at runtime regarding their various properties and capabilities. Plugins are dynamically loaded and can be extended and upgraded independently [4].

Sonar systems are often given over to the processing of huge amounts of information [11]. Therefore, high performance in such systems is a priority. In GSF high performance is achieved, by among others: the use of a dedicated allocator, extremely light-weight links between plugins – data can travel along the pipeline with a minimal overhead, resulting in mechanisms directly working on target memory; a plugin can, for example, write directly to extended cards' memory space or X server, allowing hardware acceleration by using specialized plugins [4].

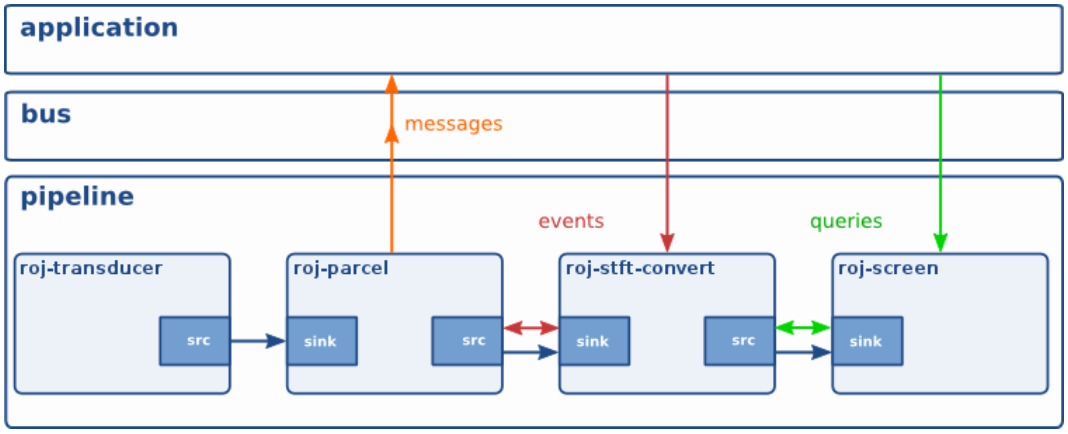

Fig. 2. Scheme of the GStreamer pipeline consisting of GROJ plugins with information flows.

Essentially, GStreamer plugins are the main element of applications. GSF provides different methods that allow for controlled communication between the plugins, that is to say: messages, events and queries. These mechanisms function based on an abstracted control layer – *bus*, Fig. 2. Each plugin has data streaming interfaces – *pads*. The pad is known as *src* if its sends data or as *sink* if it receives it. Moreover, plugins can be configured by a programmer using so-called *properties*. Pads are also used for negotiating links and for data flow between plugins. Links are only allowed between two pads that can deal with the compatible data type. The negotiation process is automatic. However, a programmer can force a specific data format from the availability of both pads. Data format is shortly known as *caps*. All plugins are packed into containers – *bins*. The containers can control plugins as well as provide an interface for management and the synchronization of all internal elements. Bins also forward bus messages from their contained plugins or different bins (such as error messages, tag messages or end-of-stream messages) and mediate in communication with a user application. A *pipeline* is a top-level bin [4, 12].

2. GROJ PROJECT

We developed several GStreamer plugins which are useful in digital signal processing and analyzing. This project is referred to as **GROJ**. The source code of the project is published under free license on the following webpage: https://github.com/dsp-box/gROJ. Currently, the project consists of the following plugins:

- **roj-gener**
  This plugin is dedicated to generating well-known periodic signals using in the sonar technique, such as: pulses, chirps, and sine waves.
- **roj-load**
  The role of this plugin is to load signals from files. Currently, only parsing the wave form audio format (wav) and reading simple text files are possible.
- **roj-noise**
  This plugin is used in order to add noise into a loaded or generated signals. This function can be helpful during testing and/or simulations. Currently, only additive white Gaussian noise (AWGN) can be generated.
- **roj-line**
  This plugin contains a few methods dedicated to the modification of signal basic parameters, such as: damping, amplification, (de)modulation, delay, etc.
- **roj-filter**
  This plugin implements a FIR filtration algorithm. The complex impulse response of a filter can be entered as a property of the plugin.
- **roj-parcel**
  The main role of this plugin is the frame forming – signal segmenting into frames. The frames can be overlaid. This plugin can also cut off the initial and/or final parts of an analyzed signal. The frame width corresponds to the window width in plugins which are dedicated to the calculation of STFT.
- **roj-stft**
  This plugin is designed in order to calculate the short-time Fourier transform (STFT). The format of an input stream should be complex signal frames. Each incoming frame can be tempered using the Blackman-Harris window. The data in the output is a set of complex numbers containing a single discrete Fourier transform.
- **roj-convert**
  The convertion of the STFT into the time-frequency distribution of various signal parameters is the main purpose of this plugin. Currently, the energy or phase of the STFT can be obtained. The output stream may contain many channels with time-frequency distributions of different parameters.
- **roj-kadr**
  The roj-kadr plugin is used in order to crop a selected region of a time-frequency distribution. It can help to display a part of a large image in a computer monitor.
- **roj-screen**
  This plugin is dedicated to visualize time-frequency representations of an analyzed signal obtained during either signal processing or analysis. This plugin can display an animation by using the GTK programming library. Moreover, it is also possible to generate and save imaging in Portable Document Format (PDF) as well as in Scalable Vector Graphics (SVG).

- **roj-olsa**
  The main role of this plugin is to listen to the signal. The stream is sent into a speaker by using the Advanced Linux Sound Architecture (ALSA).
- **roj-debug**
  This plugin is used in order to debug and test developed plugins. It can print data and control information transferred through the pipeline.

Each of these plugins processes a stream immediately after collecting a sufficient amount of data and then forwards results in the real-time. Some of them emit internal control tags necessary for signal processing in other plugins. All control information is also forwarded as soon as possible. More detailed information about the selected plugins is presented in the following sections and in the documentation about the GROJ project [4]. The GROJ project started out from a graduation assignment [13].

3. SOUNDING SIGNAL GENERATION

Frequency modulated short bursts and chirp signals are commonly used as sounding signals in sonar techniques. A discrete version of these signals can be simply generated using header files such as *complex.h* or *fftw.h* and mathematical formulae in the time domain, for instance, the linear frequency modulated (LFM) chirp can be expressed as follows:

$$u(t) = \exp(j\pi r t^2 + j2\pi f t) \tag{1}$$

where t, r, and f are respectively time, chirp rate, and initial frequency, $j^2 = -1$. Currently, there are two plugins dedicated to enter signal samples into a pipeline: *roj-load* and *roj-gener*.

Inside any plugin, generated complex samples are stored in a structure referred to as *GstMemory*. However, plugins send, receive, and forward structures referred to as *GstBuffers*. Each *GstBuffer* can contain pointers to *GstMemory* structures as members of a structure. A new *GstBuffer* has sixteen slots. Each *GstMemory* should be allocated explicitly. In the proposed solution, the first *GstMemory* stores a *GrojConfig* structure which contains control information, for example, a time-stamp, analyzing window parameters, a number of transmitted samples etc. Others slots are interpreted as channels intended for the transmission of different types of signal parameters, distributions, or representations in the time or joint time-frequency domain. Each *GstMemory* contains only a segment of all the information. Its size depends on the current configuration of the pipeline, the format and the types of transmitted data.

4. TIME-FREQUENCY ANALYSIS

The most common operation dedicated to the time-frequency analysis is the short-time Fourier transformation. The resultant complex transform – STFT, can be simply converted to a distribution of the energy on the time-frequency plane. The whole operation can be conducted mainly by the following plugins: *roj-parcel*, *roj-stft*, and *roj-convert*. First of all, the signal is segmented into frames. The frames can be overlaid. Then each frame is multiplied by an analyzing window and finally transformed into an instantaneous spectrum. The energy density can be obtained by the squaring. The whole process can be conducted on-line in real-time.

The Fourier transformation is implemented based upon the Fast Fourier Transform in the West (FFTW) library. This ensures high performance in the operation and flexibility in the analysis parameters selection. Currently, the usage of one of two analyzing windows is possible, namely: the rectangular and the 4-term Blackman-Harris one [14].

5. ONLINE VISUALIZATION

There are several common types of sonar signal imaging. One of them is the classical spectrogram where the horizontal axis represents time and, respectively, the vertical – frequency. Therefore the spectrogram can be interpreted as the energy distribution on the joint time-frequency plane. The *roj-screen* plugin is set aside to the graphical presentation of this spectral energy density. An example of the imaging of LFM chirp is presented in Fig. 3.

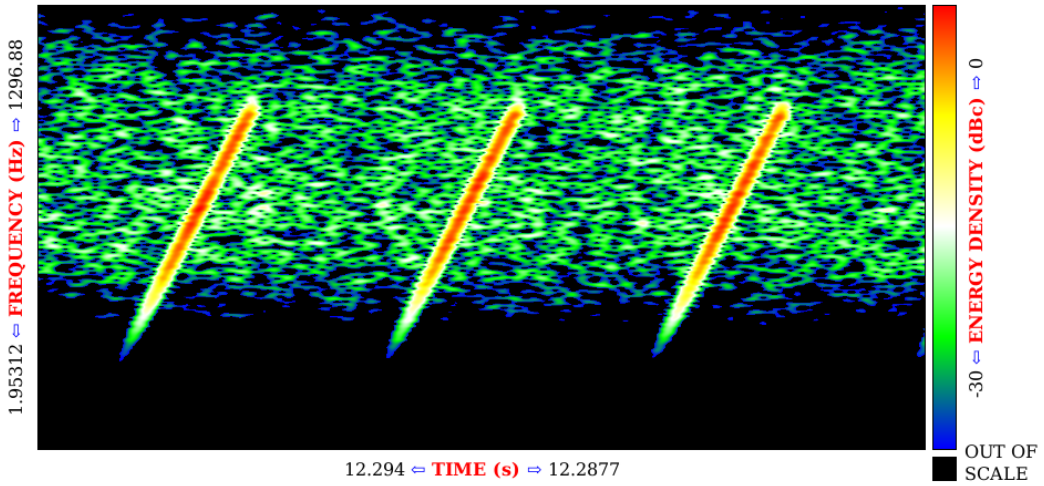

Fig. 3. An example of the imaging of LFM chirp by the *roj-screen* plugin in the logarithmic scale. The signal is analyzed by using the short-time Fourier transformation and the Blackman-Harris window.

The front screen of the *roj-screen* plugin is updated as quickly as possible and can be used for visualizing in real-time. Currently, two display modes are available: in linear and logarithmic scales. Moreover, a multi-colored palette can be used. In Fig. 4, the architecture of the plugin is graphically presented. Two main layers can be highlighted: the GStreamer processing engine and the graphical user interface implemented in the GIMP Toolkit (GTK). Both are based upon the GObject model and the Glib library. Both also support the event-based mechanism for control information handling. The information flow between these two layers is conducted through a circular buffer. The configuration of the plugin, for example, screen resolution or energy threshold, can be interactively changed without interrupting the signal processing. This feature is provided by the GSF architecture and plugin solution. Moreover, there is the option to save a screen-shot into a file – Fig. 3 is obtained in this manner.

It is possible to launch one processing pipeline in two or more machines. For example, a signal can be analyzed in one computer and the results can be presented in another machine, or many machines, using, for instance, the User Datagram Protocol (UDP) for communication. For this purpose, *udpsink* and *udpsrc* plugins can be used. These plugins are provided and supported by the GStreamer community. It is also possible to use other existing GStreamer plugins such as: *queue*, *filesink*, *tcpsrc*, *tcpsink*, etc.

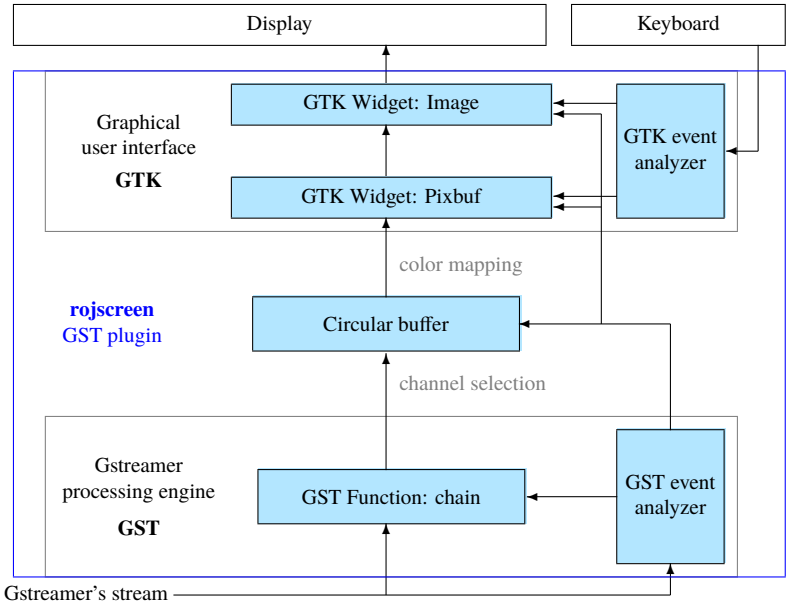

Fig. 4. Information flow diagram in the GStreamer *rojscreen* plugin.

6. SIGNAL PROCESSING

6.1. Signal filtration and modulation

The filtration operation can be realized by the *roj-filter* plugin. Currently, the finite impulse response (FIR) filtration algorithm is implemented. Filter coefficients can be entered by using properties of the plugin. For example, a frequency bandpass of the signal, whose spectrogram is presented in Fig. 3, is limited by the *roj-filter* plugin.

Methods which process isolated samples (one by one) are available by using the *roj-line* plugin. At the moment, this plugin can work as the following: modulator, demodulator, amplifier and suppressor.

6.2. Converting into audio signal

Historically, first sonar systems only allowed us to listen to signals which had come from an underwater environment. Still today, an experienced sonar operator can obtain a lot of information from this type of natural analysis. In the GROJ, project the task of sending a signal into an audio output is carried out by the *roj-olsa* plugin. The Advanced Linux Sound Architecture (ALSA) framework is used for communication with a soundcard. ALSA is reliable software, designed to ensure as small a latency as possible in signal processing. The format and other parameters of an output audio stream and its internal settings can be specified by using ALSA interface. The *roj-olsa* plugin can control this process by GStreamer properties.

Moreover, a processed signal can be adjusted, especially demodulated, by the *roj-line* plugin. It is often necessary in order to shift the signal to the audible frequency band. The *roj-line* plugin can also adjust the signal energy by digital amplification or attenuation.

7. CONCLUSIONS

The need for the use of high-level software layers in modern information technology is clear. There are many conceptions based upon, for example, things as: operating systems, software libraries or various frameworks, such as the GStreamer. Every year, new efficient and

high performance computing hardware platforms are introduced. Often new implementations of known algorithms are required in order to make use of these new hardware platforms. This raises the cost of projects, extends implementation, and, consequently, justifies the use of high-level software layers such as the GStreamer framework and the GROJ project. Then again, the GROJ project enriches GStreamer functionality. The new plugins provide the chance of the easy implementation of classical DSP algorithms.

The GROJ project is only a simple example. There are still many problems which should be solved as well as new plugins, documentation and tutorials are necessary. However, the project will be progressively developed primarily as a didactic student project in the Gdansk University of Technology.

ACKNOWLEDGMENT

The GROJ is a name of this project. The name is in honor of dr. Mirosław Rojewski, who is our longtime supervisor and mentor.

REFERENCES

[1] GStreamer: open source multimedia framework, webpage: http://gstreamer.freedesktop.org
[2] GNU Radio, webpage: http://gnuradio.org
[3] E. Blossom, GNU radio: tools for exploring the radio frequency spectrum, Linux Journal, vol. 2004, no. 122, pp. 4, 2004.
[4] W. Taymans, S. Baker, A. Wingo, R.S. Bultje, S. Kost, GStreamer Application Development Manual, 2008.
[5] S.D. Burks, J.M. Doe, GStreamer as a framework for image processing applications in image fusion, Proc. SPIE, vol. 8064, Orlando, Florida, USA, 2011.
[6] D. Darling, C. Maupin, B. Singh, GStreamer on Texas Instruments OMAP35x Processors, Proc. the Linux Symposium, Montreal, Quebec, Canada, 2009.
[7] M. Lanoe, E. Senn, Consumption analysis and estimation in the design of GStreamer based multimedia applications, Proc. Conference on Design and Architectures for Signal and Image Processing, pp. 1–7, 2012 .
[8] S. Nimmi, V. Saranya, G.M. Theertha Das, R. Gandhiraj, Real-time video streaming using GStreamer in GNU Radio platform, Proc. International Conference on Green Computing Communication and Electrical Engineering, Coimbatore, India, 2014.
[9] GObject Reference Manual, The GNOME Project, 2014.
[10] M. Warkus, Official GNOME 2 Developer's Guide, No Starch Press, 2004.
[11] J. Marszal, Digital signal processing applied to the modernization of Polish Navy sonars, Polish Maritime Research, vol. 21, no. 2(82), pp. 65–75, 2014.
[12] R.J. Boulton, E. Walthinsen, S. Baker, L. Johnson, R.S. Bultje, S. Kost, T.-P. Muller, W. Taymans, GStreamer Plugin Writer's Guide.
[13] D. Dąbrowski, Strumieniowe przetwarzanie danych spektralnych, praca dyplomowa, Politechnika Gdańska, 2015, (in Polish).
[14] F.J. Harris, On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform, Proceedings of the IEEE., vol. 66, pp. 51–83, 1978.
[15] K. Czarnecki, M. Moszyński, Using concentrated spectrogram for analysis of audio acoustic signals, Hydroacoustics, vol. 15., pp. 27–32, 2012.