Technical University of Eindhoven

Tiny Multimedia Framework

Design Document

Arash Shafiei
Sep 2014

Table of Contents

Table of Figures

Preface

In 2011, I started a project on zoom-able video streaming at National University of Singapore. The goal of the project was to implement a zoom-able video streaming server. As a result of this collaboration with Jiku team[1] we implemented a system which enables mobile phone users to access a video streaming server, navigate different cameras, and zoom into their region of interests.[2]

In 2012, I joint the GPAC group[3] at Telecom ParisTech. I started working on a video streaming project called DashCast[4]. The aim of this project was to develop a video streaming server for the new protocol Dynamic Adaptive Streaming over HTTP (DASH).

Since I had some experiences with implementing multimedia applications using ffmpeg/libav[5] library I started implementing the new application using the same library. The problem I encountered was that I spent one year doing the same things that I had done the year before. There were a lot of code that could be reused but it was not very easy to reuse them.

After publishing DashCast as an open source application, I received some messages asking questions on how certain things can be done using the libav library and how some other new multimedia applications can be developed. There were already a lot of tutorials on internet but using libav library was not trivial.

Lacking good documentations on libav and the potential that it has for developing multimedia applications inspired me to think of a multimedia framework where one can implement easily multimedia applications which they are interested in. There are already some open source multimedia frameworks such as gstreamer. I thought that would be nice to have a framework which is very simple to study and to develop easily the applications that I had to implement before.

It would be also a practice for me to understand better the fundamentals of multimedia frameworks. Therefore I started the Tiny Multimedia Framework (TMF) project to get more familiar with the challenges in the domain of multimedia framework implementation.

I would like to thank people who I have so far worked with and have always helped me with technical and non-technical problems: Ravindra Guntur, Wei Tsang Ooi, Jean Le Feuvre, and Cyril Concolato. I wish them all the bests.

Arash

Eindhoven, September 2014

---

[1] http://liubei.ddns.comp.nus.edu.sg/jiku/

[2] Jiku Live: Live Zoomable Video Streaming, Arash Shafiei, Ravindra Guntur, Ngo Quang Minh Khiem, Wei Tsang Ooi
In Proceedings of the 20th ACM International Conference on Multimedia (MM), 29 October - 2 November 2012, Nara, Japan. (Demo Paper)

[3] http://gpac.wp.mines-telecom.fr/

[4] http://gpac.wp.mines-telecom.fr/dashcast/

[5] https://libav.org/

Introduction

To implement multimedia applications, many commercial and open-source frameworks are available. Among those frameworks we can name Microsoft Media Foundation[6] and gstreamer[7] which are the most famous ones.

All multimedia applications usually use some typical components. These components include media de-multiplexer, video decoder, audio decoder, video encoder, audio encoder, and media multiplexer. A multimedia framework makes it easy to implement multimedia applications in few lines of codes. It also facilitates the maintenance of such applications since it separate the abstraction layer. A multimedia framework also makes the components reusable since a component written for one application can be used in other applications.

The difference between the Tiny Multimedia Framework and other existing open-source frameworks is that it is supposed to be very simple to use and extend. A plugin designer is able to develop a new plugin for the core framework with little knowledge of the framework. Another goal of the Tiny Multimedia Framework with other framework is that it only provides the core and all video processing APIs from other libraries (mainly libav and GPAC) are used. It is also an attempt to improve the design of some previously written applications.

In this document first we present the key concept of multimedia framework. Then we analyze the requirement of the project. We discuss few use cases, analyze the problem domain and provide our design approach of the framework.

Backgrounds

A multimedia application consists of some components that we call them filter. A set of filters are connected to each other and they transfer data between them using some buffers. Buffers can be of different types. For example a video encoder filter receives a buffer of type of raw video frame and produces a buffer of type of encoded video frame. Filters also can communicate to each other through a message channel. All these filters are put in a so-called pipeline which controls the filters and also is an interface between filters and user of the application.

To transfer data between filters, each filter has a so-called port which owns the buffer. It knows the type of data that it can accept and also it pushes the data to the successor ports. In gstreamer ports can negotiate to each other before the pipeline starts running. They negotiate about the data types they can handle. In gstreamer the port can dynamically be added to the filter. For example a demuxer filter does not know in advance what input it gets to determine how many output stream it can produce.

In Figure 1 you can see an example of a multimedia application implemented by gstreamer.

---

[6] http://msdn.microsoft.com/en-us/library/windows/desktop/ms694197(v=vs.85).aspx

[7] http://gstreamer.freedesktop.org/

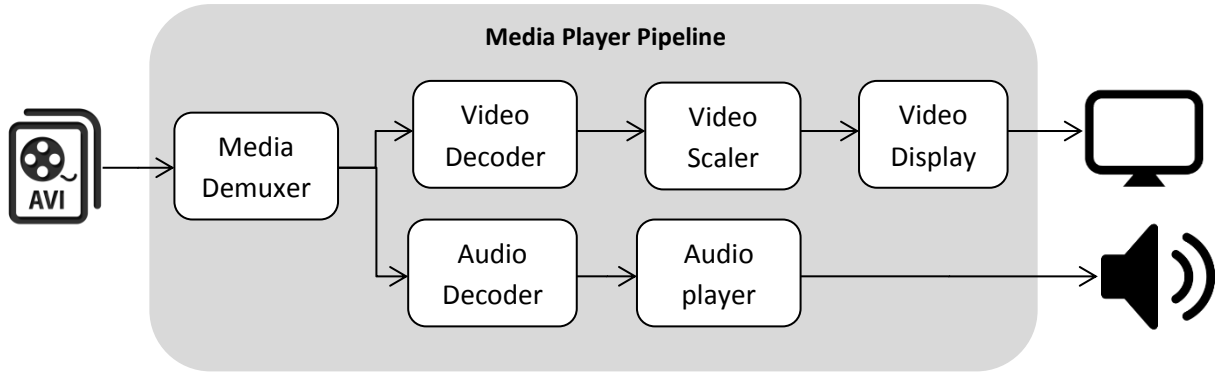

Figure 1 an example of a media player pipeline containing media demuxer, video decoder, audio decoder, video scaler, video display and audio player filters.

Although there are many applications which have been implemented using gstreamer, there are other applications which need filters that are not available. In Figure 2 you can see a multimedia application which we aim to produce using the Tiny Multimedia Framework.

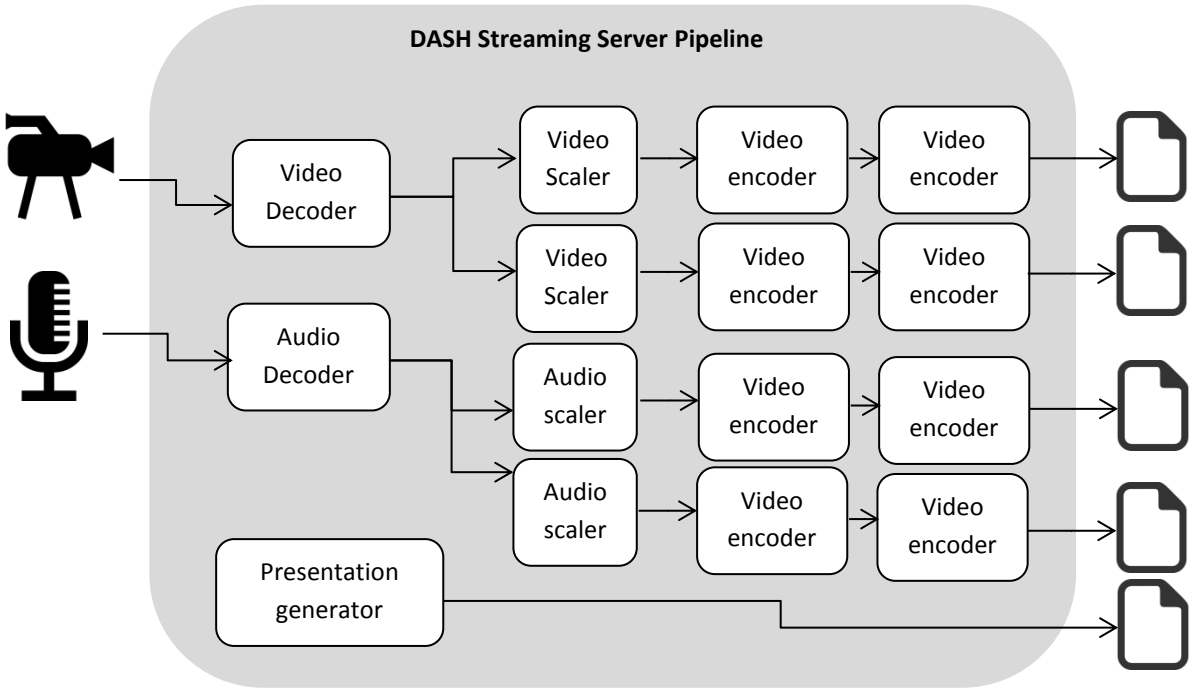

Figure 2 an example of a DASH (Dynamic Adaptive Streaming over HTTP) video streaming server. The video server decodes, scale, encode and format the media in particular format according to DASH standard. It also generate a presentation descriptor.

Requirements

There are a set of functional and non-functional requirements.

Functional: The user of the framework must be able to create pipeline and filters. She must be able to connect the filters to each other and run the pipeline. The developer of the filter must be able to

create a filter, containing ports. She must be able to read ports value, process them and write the results on the output port.

Non-functional requirement: One of the key motivators of the project was non-functional requirements. The framework must be easy to use. The filter design must be easy. The code must be extendable. It means adding a filter to the system does not have to break the implementation of the core. The core must be stable during the implementation of the filters.

Use-cases

Based on requirements, there are two main actors namely multimedia application developer and filter developer. A multimedia application developer needs to implement a pipeline based on the available filters. She picks filters that she needs connect them to each other, set the parameters, initialize, and run the pipeline.

A filter developer needs to decide what sort of processing she provides. She needs to create ports that she needs, implement initialization and processing mechanism of the filter. This means she needs to define what happens when the filter is initialized and what happens when a filter is processing data.

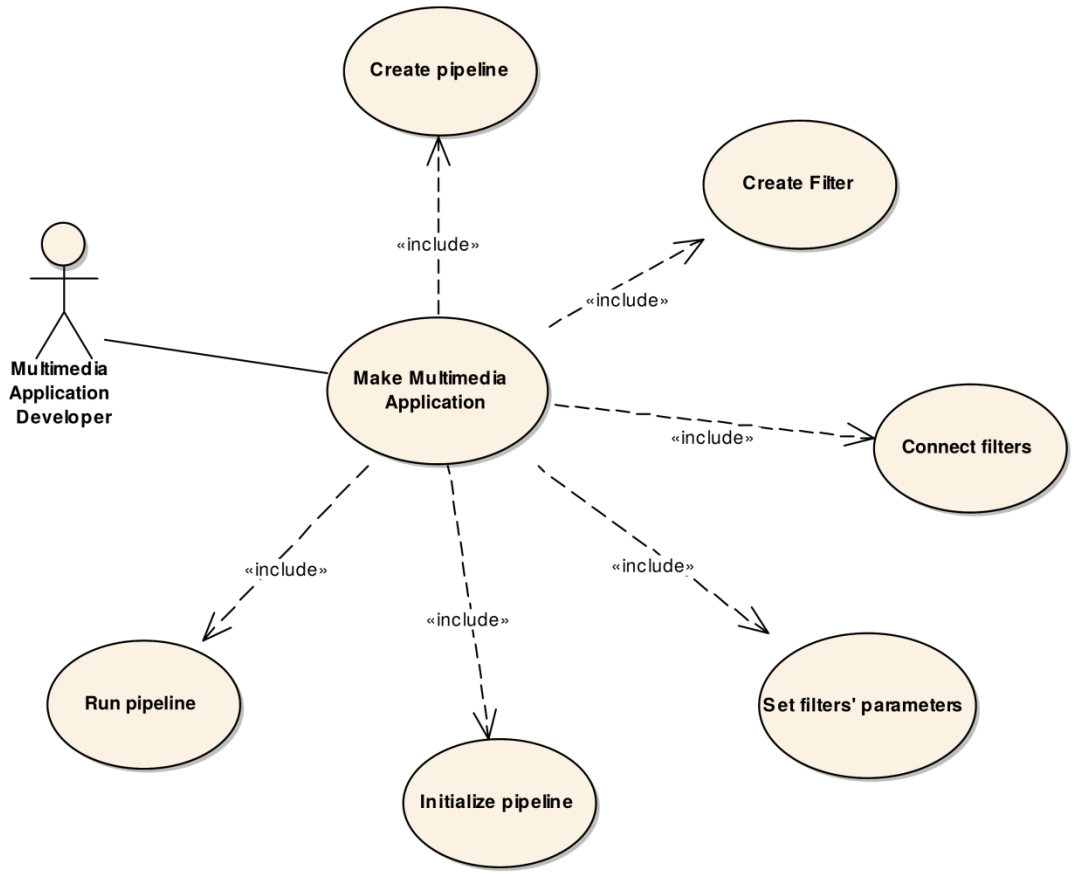

Figure 3 multimedia Application developer makes an application by creating pipeline and filters running them

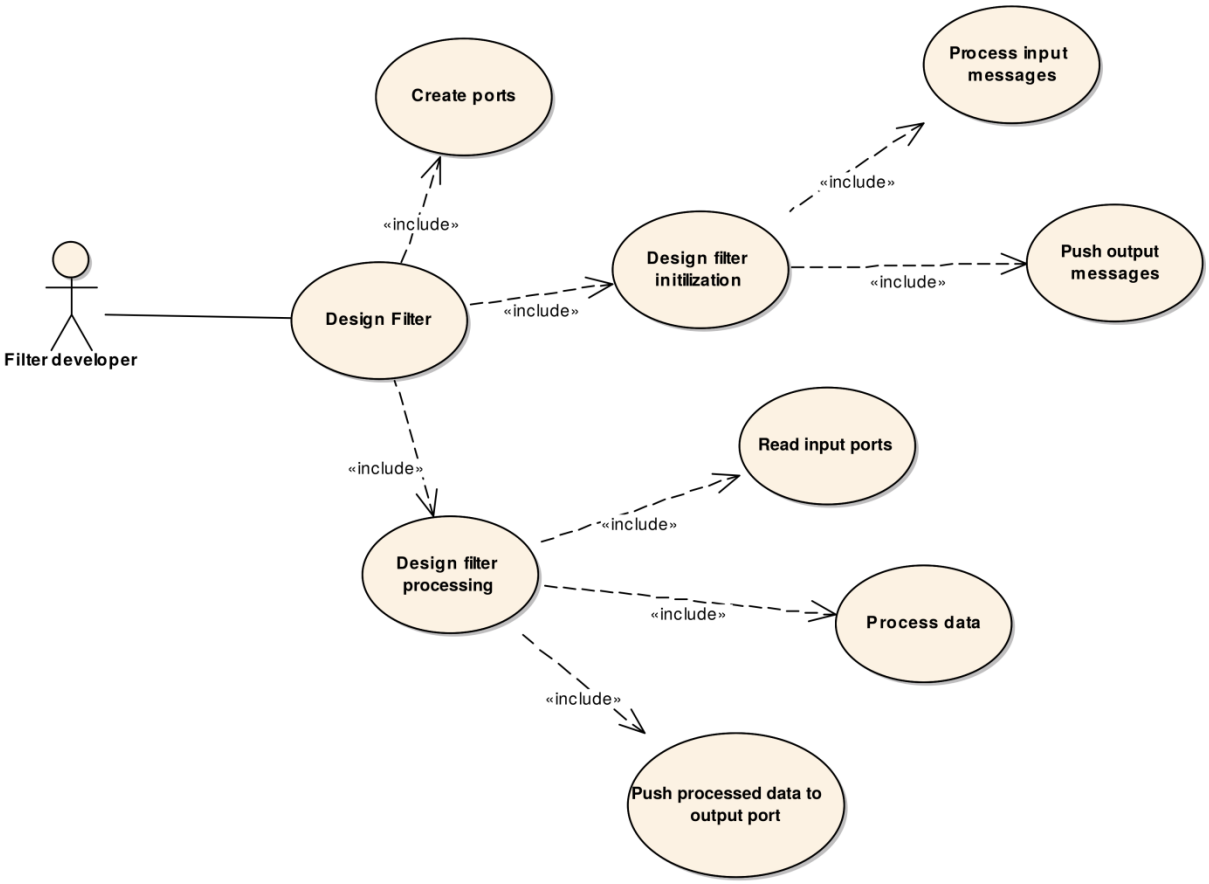

Figure 4 filter developer designs a filter by defining the ports of the filter, and defining initialization and processing mechanisms.

Multimedia Framework Design

Based on use case we propose the domain model presented in Figure 5. The user creates a pipeline and creates the filters. She adds the filters to the pipeline, initializes and runs the pipeline. The filters have a reference to their successors. Each filter processes its data, pushes its data to its successor, and calls its successor to proceed.

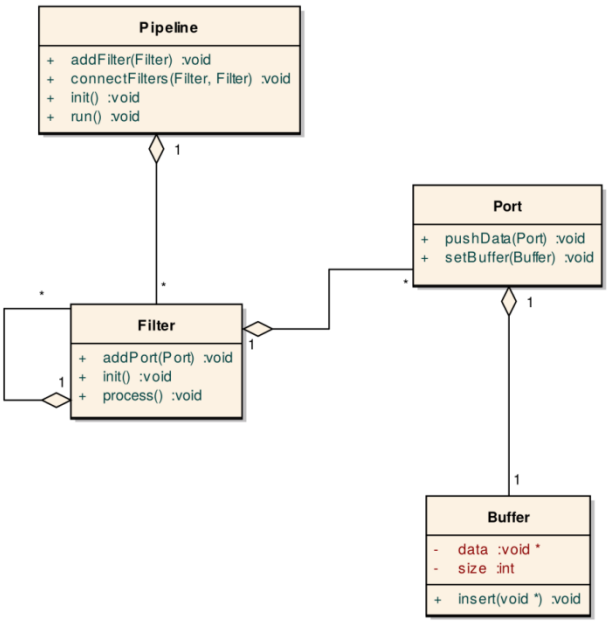

Figure 5 the domain model of the problem

In the following subsections we will address some of the problems to solve in our design.

Communication between filters

One of the problems is that filters sometimes need some information that they don't have it by themselves. For example an encoder filter needs to know about the dimension of the frame that it receives. It will be dynamically known by the previous filters and it might not be known before the run-time. For this reason we need a communication channel between the filters. In the design model we introduce a Message class with which filters can receive messages from previous filters and send messages to the successor filters.

Capabilities of filters

Filters process different types of data. For example a video encoder filter receives a raw image and produces an encoded image. A media de-multiplexer even produces multimedia data type: encoded video frames and encoded audio frames. This introduces a challenge to the design. There are different solutions that we have thought of:

1) A filter can have a buffer and all the sub filters can have buffers which inherit from the super buffer class. Disadvantage of this method is that a filter can only have one type of buffer. We would like to have filters which accept different types if input and different types of output
2) To solve the above problem, we introduce the concept of port. A port contains a buffer. All filters which needs the data of type A will be connected to the port which offers buffer of type A. So in this second solution we will have a super filter which has some ports and ports have buffer. Then there are several types of buffer which inherits from the super buffer class.

3) There are two problems with the previous solution. First, the behavior of the input port is completely different from the output port. Therefore we introduce a port class and input port and output port which inherit from the port class. Second, there are many types of buffer for which we have to create a sub class but they are not very different in behavior. For this reason, instead of letting sub buffers inherits from a super buffer, we make a template class for buffer and let the filter designer free to use any type of port. The only disadvantage is that sometimes user-defined buffers need initialization and in this case user can define its buffer in "type" package that we later explain.

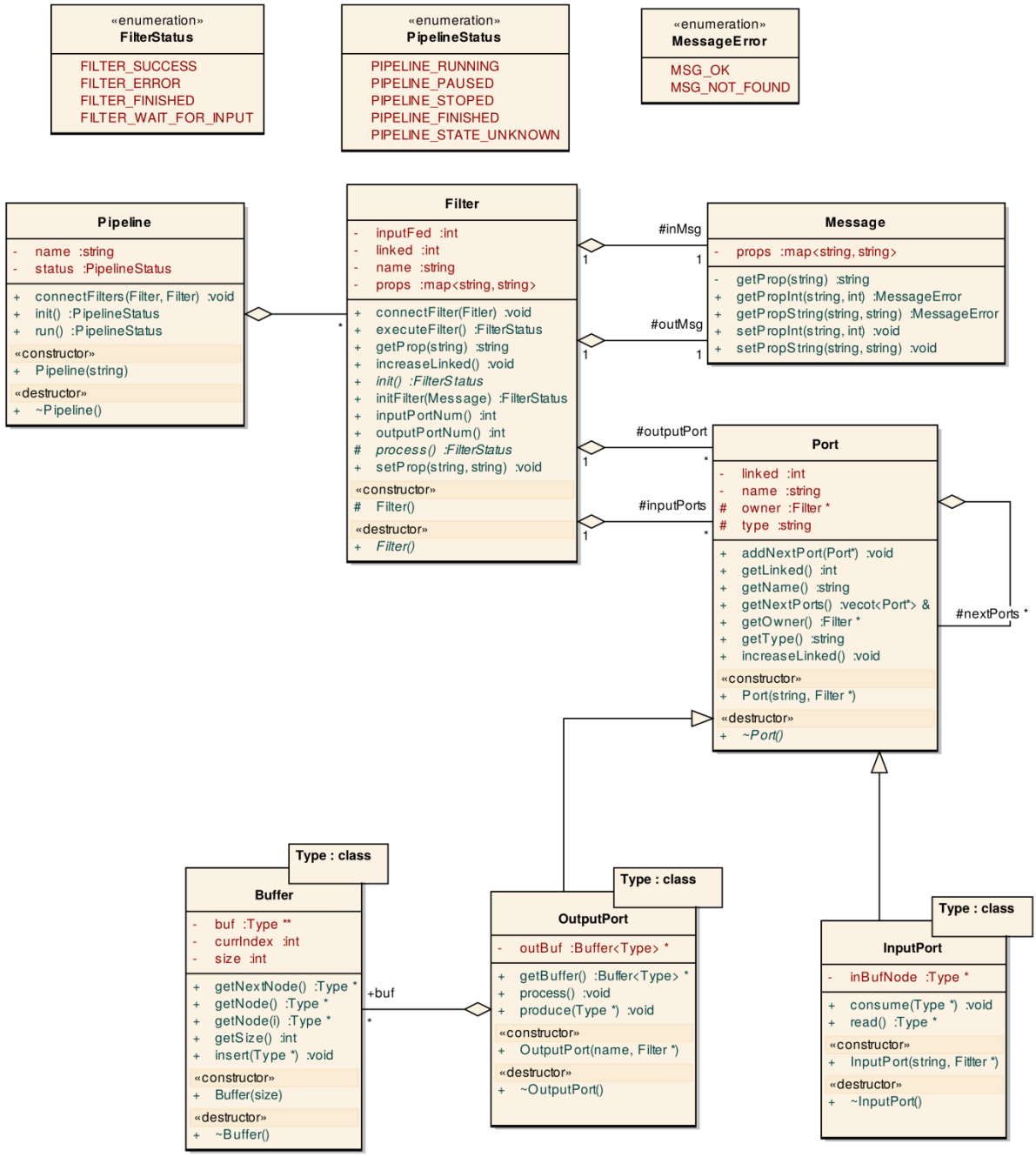

Figure 6 the class diagrams of the core package

Filter factory

Due to the non-functional requirement of ease of use, we would prefer to provide the user as minimum classes as possible. In the other hand, we would like to hide sub filters from the user of our framework. For this reason we provide a factory method class called TMF which is responsible of creating filters as well as pipeline.

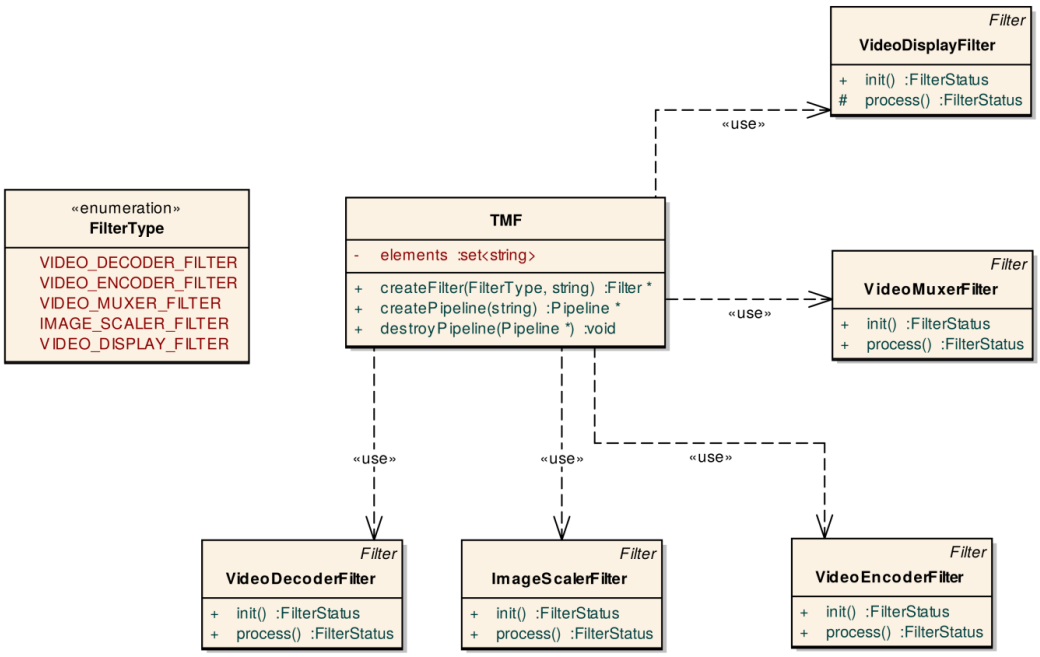

Figure 7 the class diagram of the TMF package containing the factories

Buffer data type definition

As we mentioned, we decided to make the buffer class template. The only disadvantage of this method is that for some data type we need special initialization. Another reason if that for some applications the number of frames is important. Since the framework is generic and other types of applications can be also developed, the frame number is not always necessary. We defined a Data class which contains the number of frame and the subclasses inherit from this class.

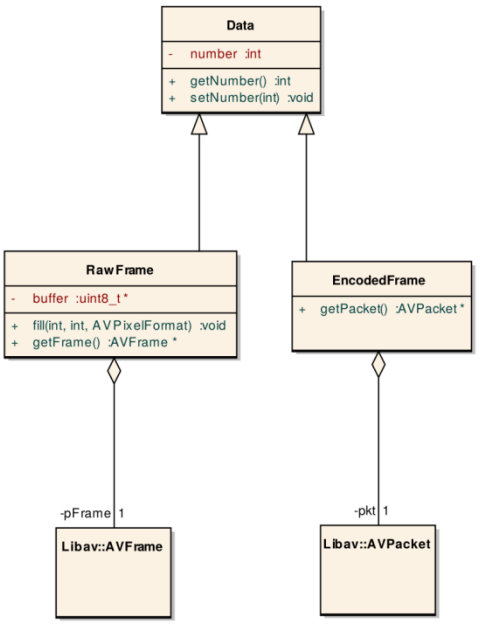

Figure 8 the class diagrams of the type package containing the additional buffer types

Filter implementation

All filters inherit from the super filter class. They must create the ports with the proper types that they need, define the initialization and process of the filter. In the initialization usually a filter receives the messages from previous filters, does its own initializations and outputs the messages to the next filter.

In order to simplify the design of the filters, some helper classes are necessary. All these classes are located in a “tool” package which will be discussed.

To implement the process of the filter, the filter developer simply needs to read the data from the port, process the data, and push the output to the output port. This is possible due to the template method which is applied in the filter class. When the filter class is executed, some of the calculations are done in the super filter class in order to prevent the sub filter classes to implement repetitive calculations. These calculations include communicating with ports and pushing data to the proper port.

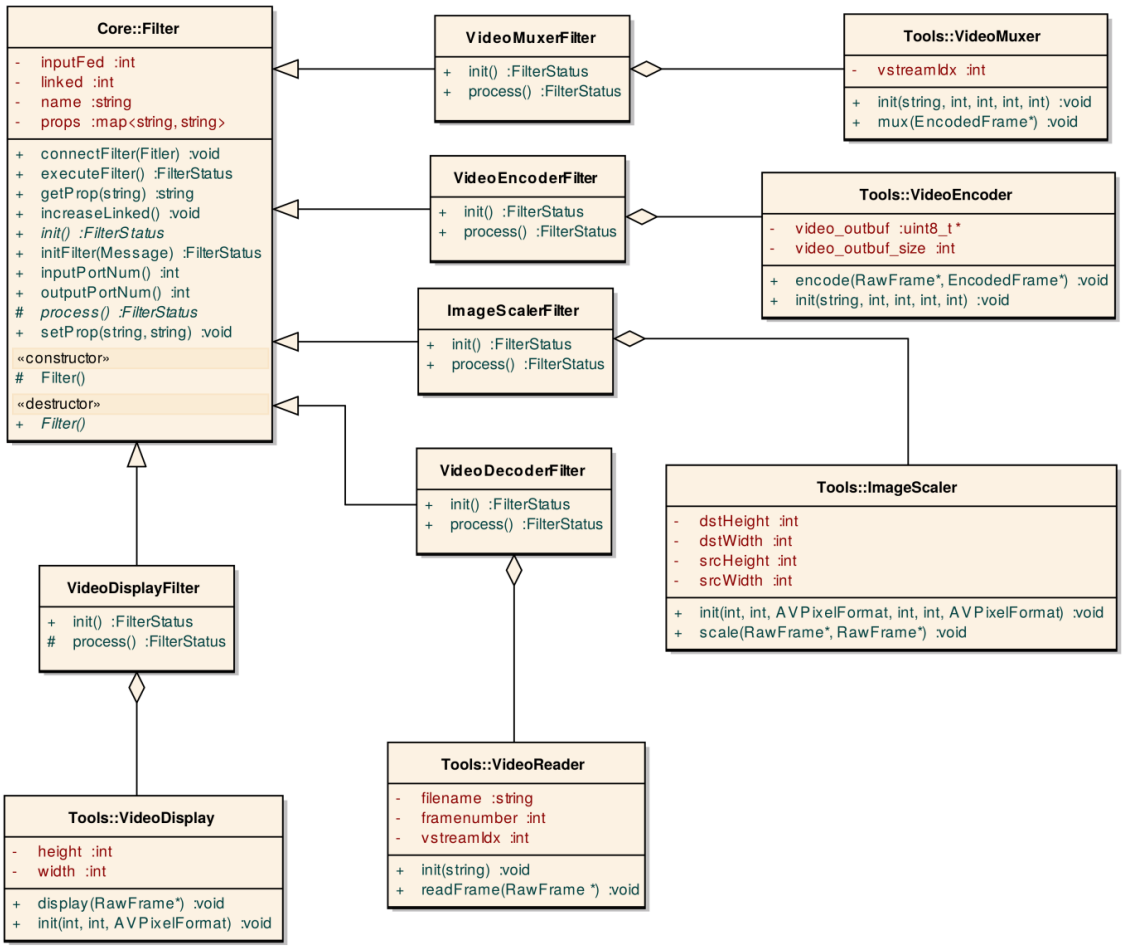

Figure 9 the class diagram of the filter package containing the implementation of the filters

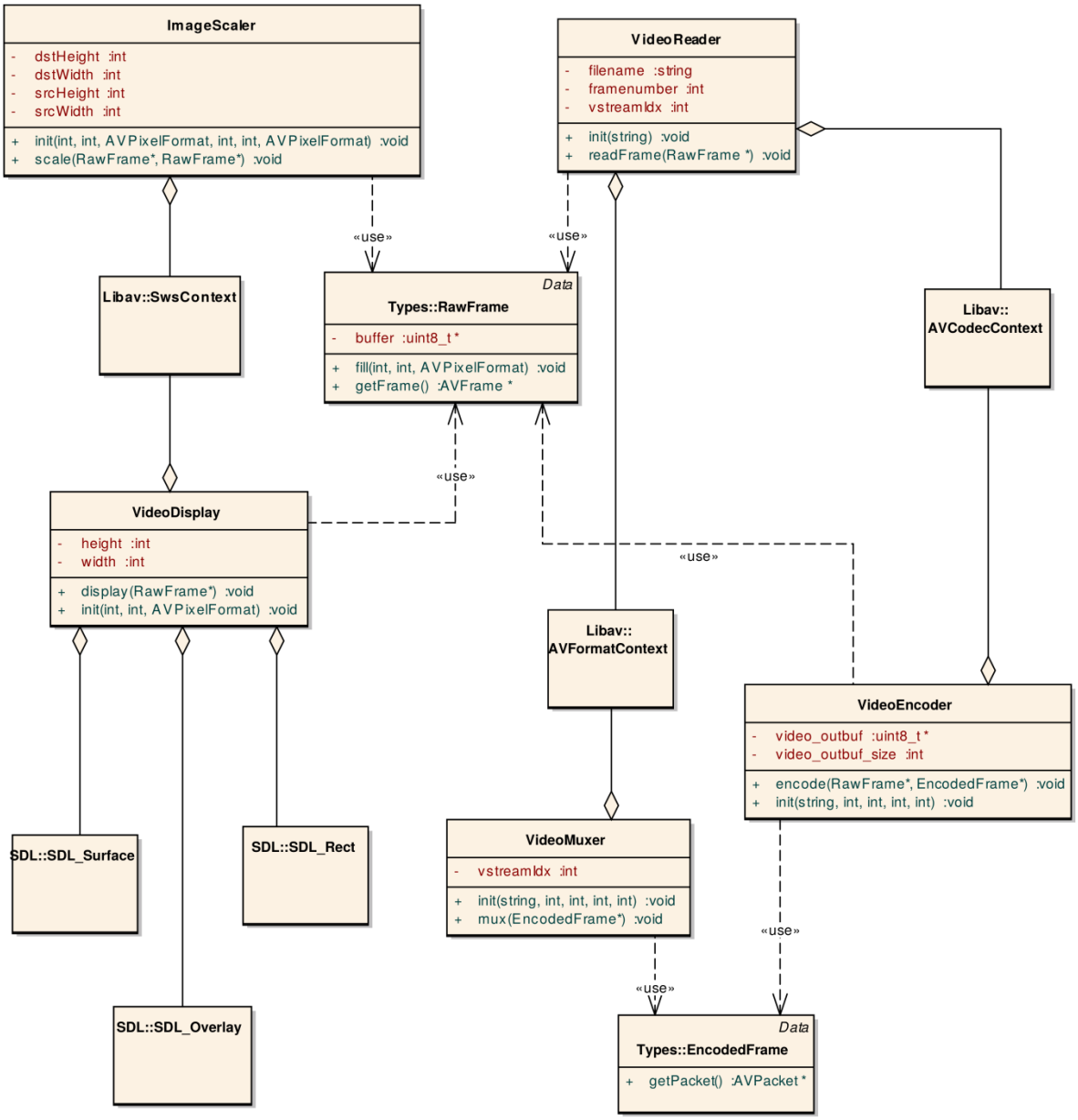

Figure 10 the class diagram of the tools package containing all the neccessary tools to develop the filters

Packaging

The packaging of the class diagrams is as follows. The core framework is placed in the core package. All implementations of the filters are placed into the filter package. The helpers are placed into the tool package and filters make use of them. All the data types needed are placed in the type package. For now the filters implemented make use of libav and SDL libraries.

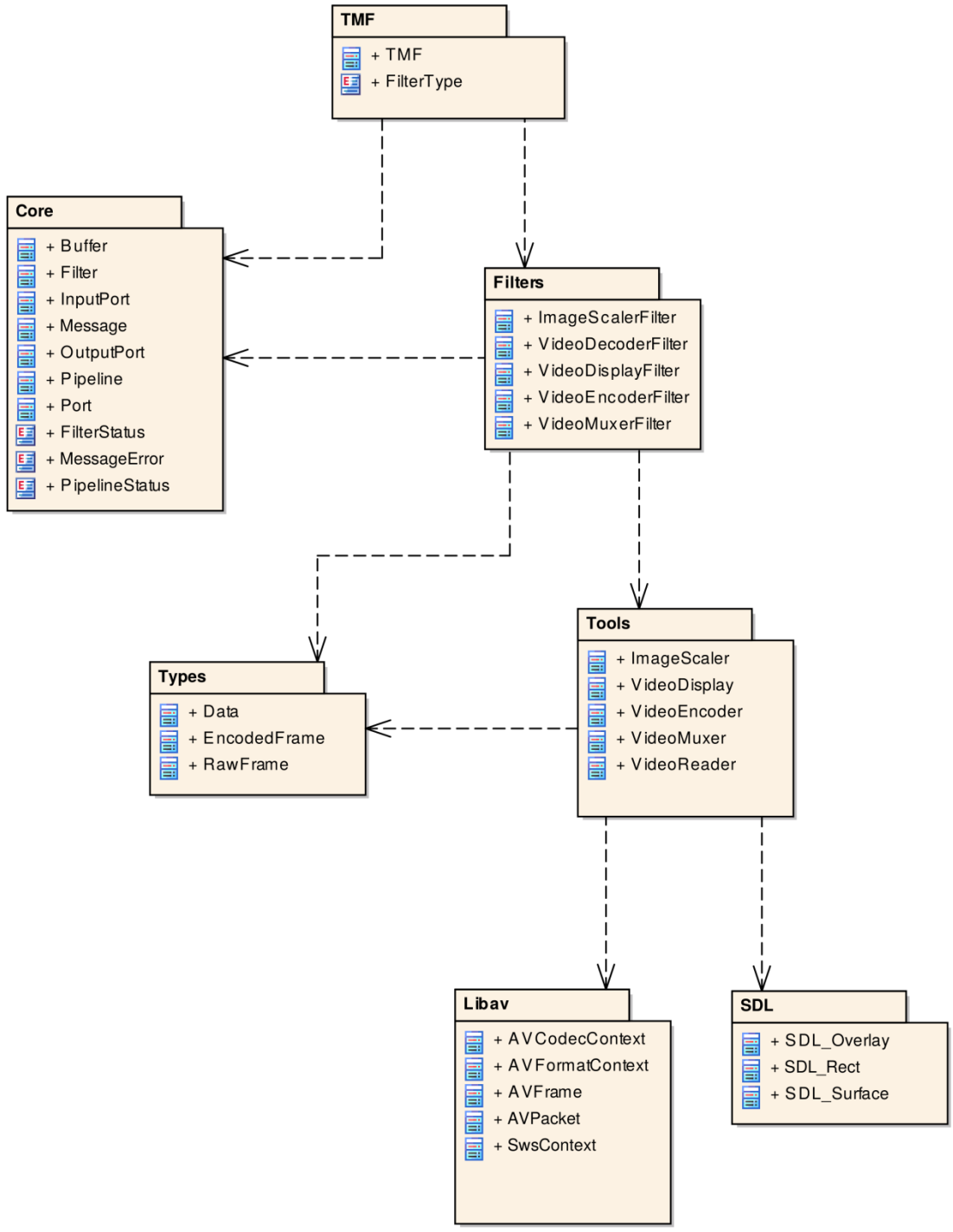

Figure 11 the package diagram

Filter processing

Running a pipeline follows a sequence of activities to push the data to all the filters and process the data on those filters. When the multimedia application developer runs the pipeline, the pipeline executes the first filter in the pipeline. The super filter class calls the process function implemented by its sub-class. The filter then puts the data on its output port and calls the process of the output port. The process function of the output port is supposed to be run on a thread. It means after calling it the filter can come back to the execution and process the future data[8]. When the process of the output port is called, the output port calls all input ports connected to it to consume the data. After data consumption, all input ports call the execute function of their filters and the same process continues for all the port until all the filters are executed for one frame.

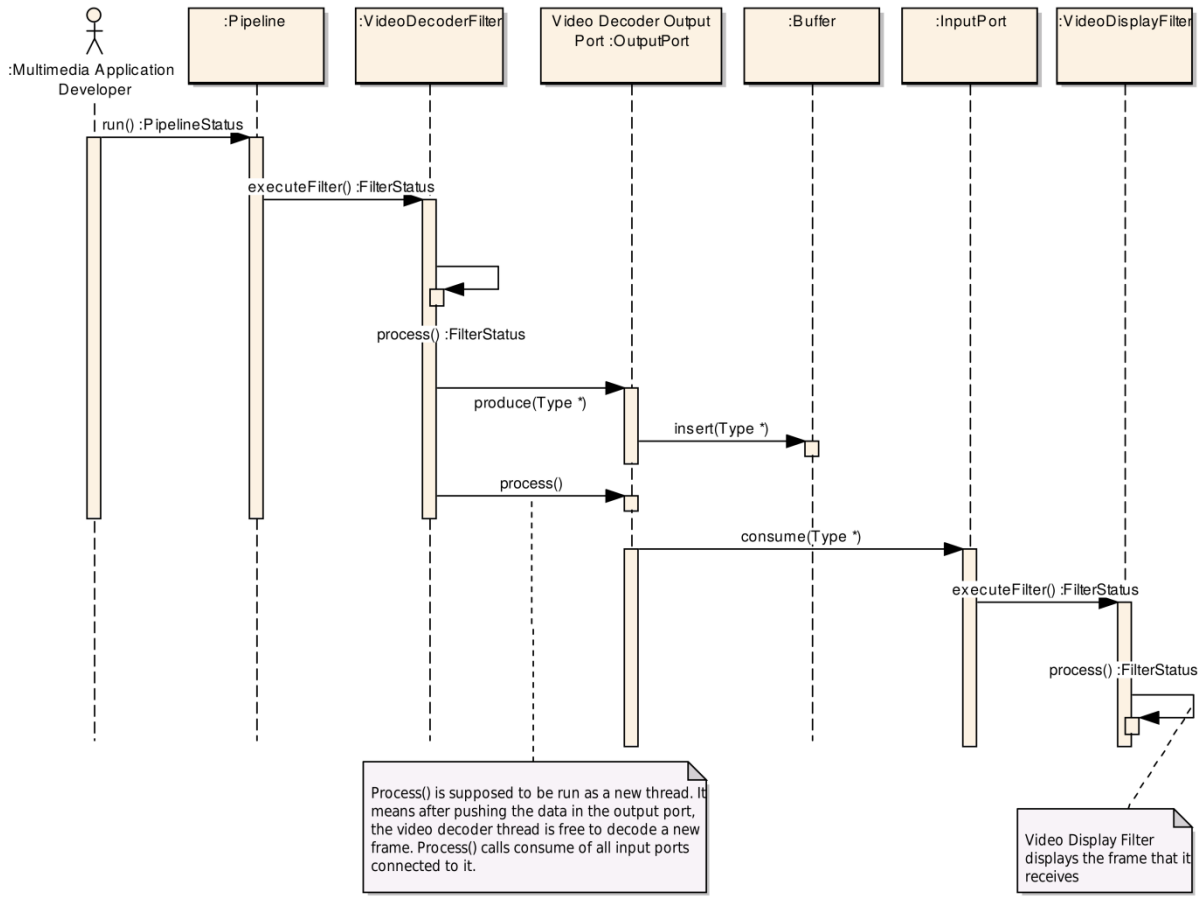

Figure 12 the sequence diagram showing the process flow when running the pipeline

[8] At this version of document the separation of threads is not implemented.

Future works

Here we list the future works to be done in order to improve the Tiny Multimedia Framework:

1) The pipeline is currently running on a single thread. In order to improve the performance we need to run each filter on a single thread and implement a synchronized buffer.
2) Currently the filter developer needs to handle the message transferring. A better mechanism could be further developed in order to reduce the filter developer's workload.
3) Error handling in the core package is not completed and it needs further improvements.
4) In the introduction we discussed a video streaming application for new streaming protocols. The filters which are currently developed do not allow us to implement that application. There are some filters which must be developed in the future.
5) A user interface can be developed to allow the user of the framework creating her application via a graphical user interface and avoid using code.