

Design and Implementation of an Open Broadband Multimedia System

Florin Lohan, Irek Defée
DMI, Tampere Univ. of Technology
FIN-33101, Tampere, Finland.
Tel. +35833653845,
Fax +35833653966,
e-mail:{lohanf, defee}@cs.tut.fi

Harri Hakulinen
Nokia Home Communications,
Tampere, Finland.
e-mail:harri.hakulinen@nokia.com

Abstract

We describe the design and implementation of a system for playing high-quality multimedia content. Our multimedia client is fully plugin-based, all the transport, demultiplexing, decoding, input and output functionalities are implemented as external plugins. Thus we can seamlessly provide streaming capability and local file based playing for several media types, different input and output possibilities. By using appropriate plugins, the player is also able to use hardware based decoders or software-only decoders, based on the computer's capabilities. The player is implemented in Linux, but it features a library containing system-specific functions. This makes portability to another operating system easier, porting only this library. The system is primarily intended to allow RTP/RTSP streaming over xDSL links of high-quality MPEG-2 content, and playing it on a Linux-based STB (Set Top Box).

1. Introduction

Digital video and audio compression and coding schemes [8] are changing our world, by bringing closer together several major players in today's world economy. The TV broadcasting industry, content producing (movies) industry, computer industry and networking service providers are likely to face similar problems in providing their customers with value-added services. Each of these industries has to step on new grounds in order to differentiate their products and services from similar ones.

Offering high quality content on demand to customers involves many problems, and may be realized with many strategies. It also involves many parts: the content has to be created, stored in a known place, transported to the customer and played. Within such strategies e.g. TV broadcasting company may like to offer its content on demand in order to reuse its value currently only broadcasted few times. A network service provider may like to offer content on demand to its customers to leverage the cost of the networking equipment for broadband access (xDSL, Cable modems). If done properly, content on demand may also boost the revenues of the content production industry, by increasing the user convenience and decreasing costs associated with classical content distribution channels. The computing industry gains by providing the technical means for all this to happen.

Because of the lack of broadband networking capacity to homes, until recently high quality multimedia made more sense for corporate users rather than for home users. But the growing adoption of xDSL and Cable modems could make this changing.

There are many differences between corporate multimedia users and home multimedia users. Probably that the most important one is that the home users are looking for a quality content, and they are willing to pay for the content they see, and not for the enabling technology. On the other hand, for the corporate users the

possibilities and the availability of the technology is the factor that matters most. Here the content is usually more oriented towards computer-aided education and communication between people in different locations rather than towards entertainment. These differences make that the business approaches towards the two markets to be different: corporate customers would be willing to pay for the enabling technology, while the home customers would be willing to pay for the content they receive and its quality. Thus the billing system has to be different, for the corporate customers selling the technology may be enough, but for the home customers a pay-per-view billing system may be more appropriate.

2. Hardware aspects

2.1. Media servers and communication links

The Internet of today may be a bottleneck for streaming high-quality multimedia over long distances. Several Mb/s required for good quality MPEG-2 streaming may be difficult to provide on backbone network when hundreds or thousands of users in remote locations try to access a single big server. Even if the user bandwidth from his or her home to the local telephony exchange (in the case of xDSL modems) or to the cable company (in the case of Cable modem) is enough, the aggregate backbone bandwidth required by all users to access remote servers might not be enough.

One solution to overcome this aspect is to place farms of small servers near (or even inside) customers' Internet Access Points e.g. in the ADSL DSLAM racks in the telephone exchange office. Server in the server farm can be a PC with RAID hard-disks and broadband network card either ATM, Gigabit Ethernet or few 100 Mbit/s Fast Ethernet cards. Hard disk capacity is rising very quickly, at the moment this paper is written, there are 75-80 GB hard-disks with ATA-100 interface on the market. These disks are very cost effective. a RAID array 4-8 such disks would provide 300-600 GB of memory space, enough to store more than 150-350 hours of high-quality MPEG-2 content encoded at 4 Mb/s. This would be enough for 80 to 200 full-length movies. One such server could be used by about 50 users and a farm of such servers could be formed by replicating the servers at a very low cost per user. In such architecture backbone network would be used only, apart of the standard Internet access, for server management and content update. The need for extremely high bandwidth is the backbone would be alleviated

2.2. Broadband multimedia clients

PC is not the only solution for home users to receive and view streamed high-quality multimedia content. New generation of set-top boxes is able to access content both from digital TV broadcast and from broadband network [1]. These set-top boxes connected to xDSL or Cable network, are ideal platform for a high-quality multimedia clients because of high degree of interconnectivity with the environment the customers are used to watch (TV, DVD) and new types of content delivered through networks (on-demand, interactive).

There is practically no other choice currently for an interactive information system than WWW framework based on IP networking. The WWW/IP infrastructure has great flexibility and enormous base of applications and it also has upgrade capabilities through plug-ins and software updates. Processing of WWW and IP data requires considerable system resources and processing power. One can conclude from this that for full WWW/IP processing a standard Web browser with operating system

providing complete system support and networking is necessary. For price sensitive consumer set-top box platform the only viable alternative is the use of Linux operating system and Mozilla Web browser which are free and publicly available with their respective source codes. On the hardware side a proper environment for running such system should be based on standard PC architecture which has sufficient processing power and support.

Cost effectiveness of this architecture can be ensured by using processors with lower speed than those in desktop PC's which are always based on latest technology. In other words, the "last year" processor which is unfit for the PC market, is perfectly cost effective solution for the set-top box. With the ever increasing processing power the PC processors can support complex multimedia applications in software.

The set-top box architecture based on the Lintel (Linux+PC) platform considered here is equivalent to a high-end workstation of few years ago. This architecture is capable of supporting all WWW applications running on desktop machines and also interactive multimedia applications for broadband services. It can be seen from this analysis that the Video-on-Demand (VoD) concept promoted in the past was in fact premature because the proper and cost-effective technology for its realization is only now becoming available. Also, proliferation of the Web brings huge range of new services and content fundamentally expanding the VoD concept.

3. Multimedia System Architecture

From the analysis above it can be seen that currently, server, client and network problems for broadband network services can be effectively solved. Broadband network set-top box terminals can be used for widely available broadcast digital TV services, e.g. from satellites plus variety of Web-based and on-demand services. In this way consumer broadband network service platform will become practical both from technical and market perspective. The main technical problem is platform architecture and integration to provide best QoS with minimized cost.

In this paper we are investigating issues related the architecture and implementation of a streaming multimedia client and its integration with PCs Linux platforms and Linux Set Top Box Systems. We made the distinction between the two by assuming that the PC has a Pentium III 500 MHz or higher, and no special MPEG-2 decoding hardware, and that the Set Top Box has a slow Celeron processor and hardware MPEG-2 decoder.

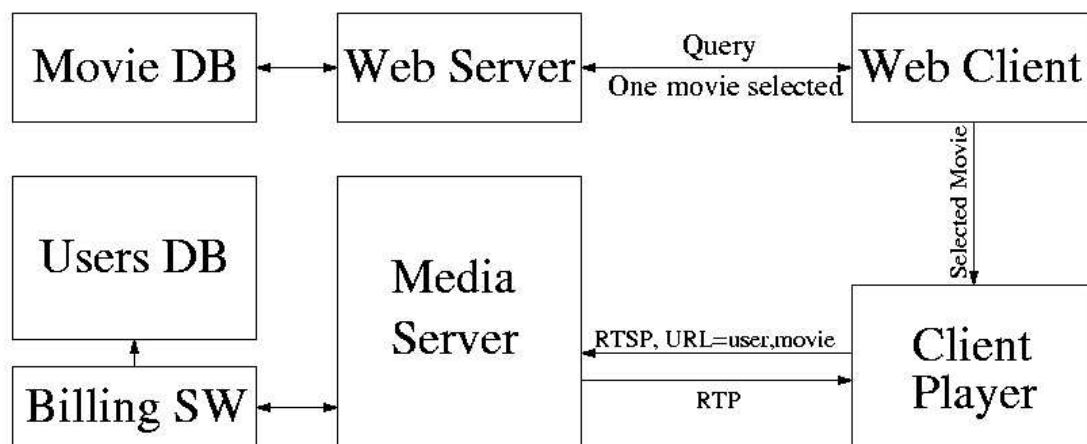


Figure 1: The general system architecture

The overall system architecture is presented in Fig. 1.

The user connects first with a web browser to a web server that can facilitate the selection of a movie. The web server should be in contact with a movie database, containing all the movies available for streaming. The user should select one movie that he or she wants to see, by performing several queries upon the movie database. The information about the selected movie is passed to the client player.

In the next step, the client player contacts the media server with the movie and authentication information. This is done by embedding the information into the URL. We assume that the control protocol between the client and the server is RTSP [5], so the URL will be sent to the server in the first RTSP command, a DESCRIBE request. The media server asks the billing software if the user having that authentication information can see the selected movie. If the user can see the selected movie, streaming is started.

There are several possibilities for the web browser to send the information about the selected movie to the client player. One of these possibilities is that the client player is a browser plugin, thus the playing of the movie can also be done inside the web browser (web page).

4. The Client Player Architecture

4.1. General Components Description

The client player can receive media streams from multiple sources: TV broadcast, network, local hard disk. Our architecture provides a uniform interface to these resources.

The main idea behind the player's architecture is presented in Fig. 2. There are three types of components, the main program, plugins and tools.

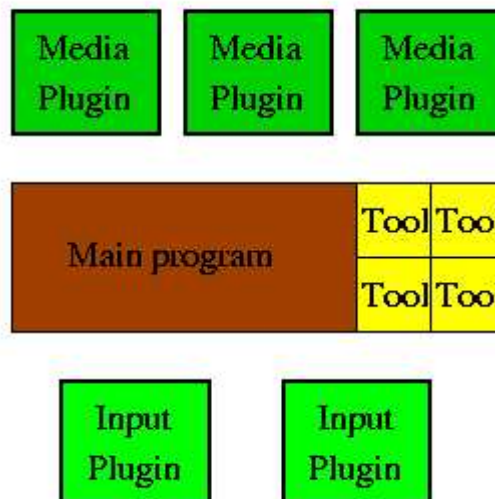


Figure 2: The general client architecture

The main program is the central unit of the player, it is responsible with keeping the state of the system and executing commands (like play or pause) that are sent from outside. The main program also commands the loading of some plugins, and informs them of the actual state of the system.

The tools are more or less stand-alone components which are linked together with the main program. Their purpose is to provide services to the main program or to the

plugins. A tool has a separate API and may use structures and information from the main program, but should not modify the main program's state.

The plugins are external dynamic link libraries that reside in a directory known to the main program. The needed plugins are loaded at runtime (using the `dlopen` Linux system function). We can split the plugins in two main categories, based upon their role in the player: there are input plugins, that interact with the user and send commands to the main program, and media plugins, that process the multimedia data through the chain from acquisition, demultiplexing, decoding to output/rendering. The media plugins also can be splitted into the above subcategories, according to their role in the media processing chain. But there is no fixed delimitation, an input plugin can also process data, a media plugin can also send commands to the main program (when it detects the end of stream, for example) and a media plugin can have several functions in the media processing chain (like demultiplexing, decoding and rendering).

A more detailed client architecture is presented in Fig. 3. The main program is splitted in few subcomponents and the relations between components and subcomponents are shown.

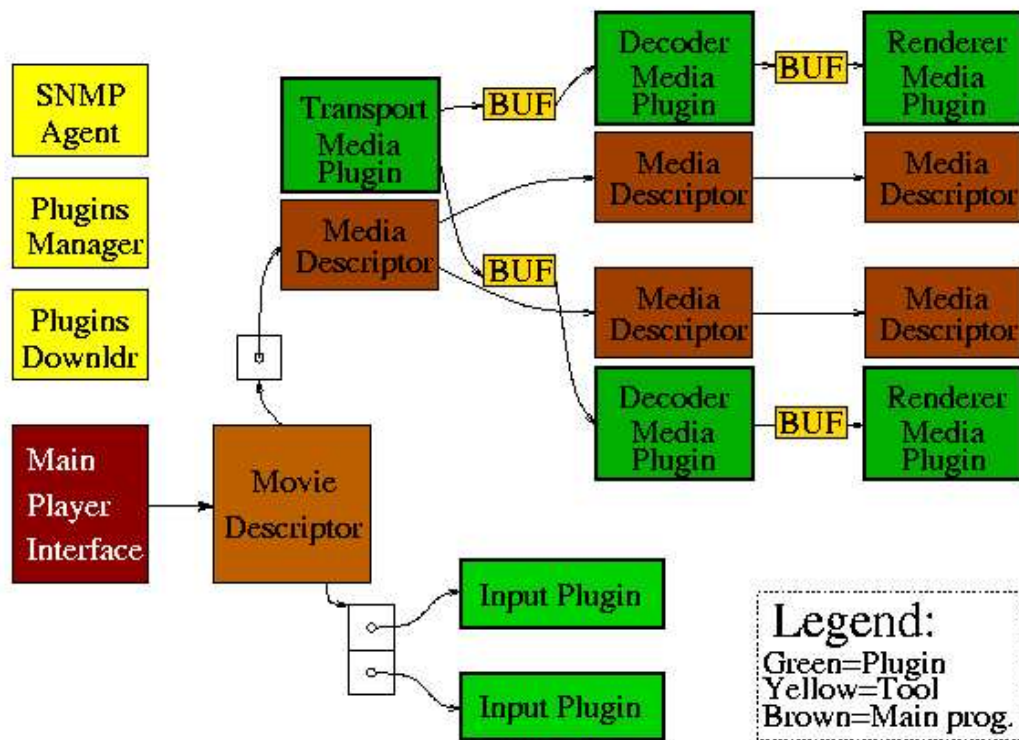


Figure 3: The detailed client architecture

The core of the player is the `MovieDescriptor` unit. It is implemented as a C++ class and contains information about the whole movie, both static (that does not change during the movie playback – description, duration, encoding type, geometry, number of frames per second, etc.) and dynamic information (that changes during movie playback – state of the system, playing position, viewing parameters, etc.). The `MovieDescriptor` unit also contains functions for changing the dynamic information – the state of the system. After the state of the system has been changed by an input plugin or by other external meaning, the new state is made known to all directly

registered plugins.

The MainPlayerInterface unit is also implemented as a C++ class and is meant as an interface of the MovieDescriptor unit to the outside world. It provides simple initialization and destruction methods and basic command (play, pause, stop) that then translates to the more complicated interface to the MovieDescriptor unit. The purpose of this unit is to ease the interconnection of the player with other software. We considered two possibilities here, when the player is implemented as a stand-alone program, or when the player is implemented as a Mozilla/Netscape plugin.

When implemented as a stand alone program, a main function has to be created, where a MainPlayerInterface unit is created and initialized. Then it can be checked if a filename or URL was provided when the executable was called, and the MainPlayerInterface unit is asked to open it. Then the main function should wait until the program finishes. When implemented as a Mozilla/Netscape plugin, a similar approach can be followed.

The MediaDescriptor unit contains information about the media processed by one media plugin (the URL of the movie, the type of media, the subcategory of the plugin, references to the next MediaDescriptor in the processing chain, references to the data pipe units between the current plugin and next plugins, a reference between the data pipe unit coming from the previous plugin). There is one MediaDescriptor associated with each media plugin. Input plugins do not have a MediaDescriptor unit associated with them. Each MediaDescriptor unit is responsible for command propagation. It receives commands from the previous plugin or from the MovieDescriptor and sends the command to its associated plugin and the next MediaDescriptors, that will do the same.

The player has several states: no stream, initializing, playing, paused, stopped, exiting. The transition between states is presented in Fig. 4.

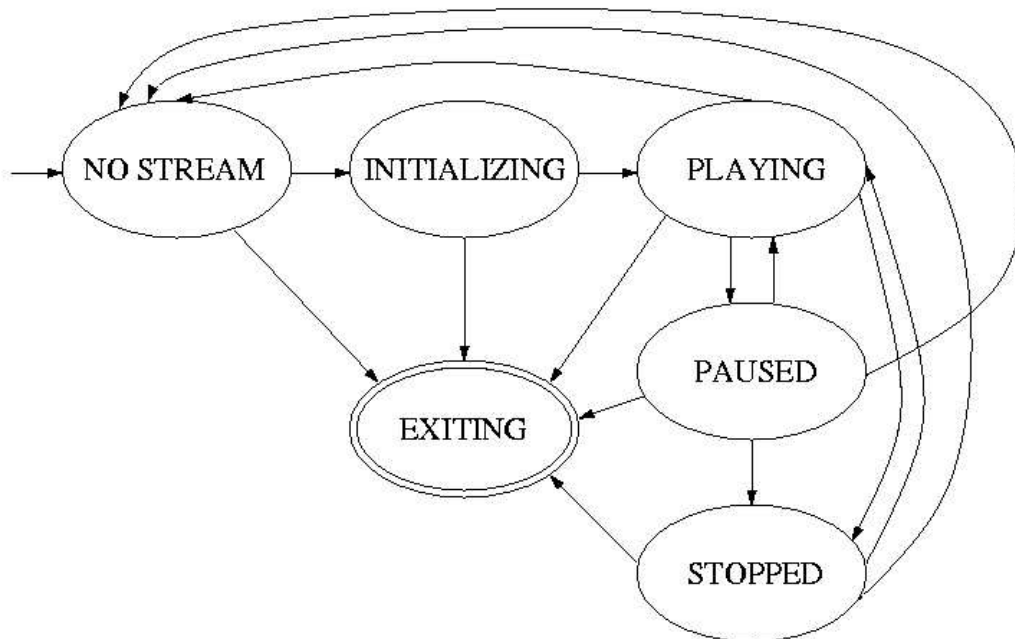


Figure 4: The player's states and the transitions between states

When the player is initialized, it automatically enters the no stream state. This means

that there is no open file or URL, and there are no media plugins loaded. However, all the input plugins are loaded (so that the user can open a file or URL, or close the program).

When a file or URL is open, the player goes to the initializing state. Here the MovieDescriptor unit tries to load a transport plugin able to handle that URL. For this purpose, a MediaDescriptor unit is created and filled with the URL of the movie. The plugin type of this MediaDescriptor is set to transport and the best suitable plugin is loaded (using the PluginsManager tool, explained later). The loaded plugin tries to complete the media chain, if itself it is not able to complete it. For this, it fills a MediaDescriptor unit with the information about what plugin has to be loaded next, and passes this information to the PluginsManager tool, to load the plugin. This continues until the whole chain is completed.

From the initializing state, the player enters in the playing state. A play command is automatically issued by the MovieDescriptor unit after the initialization finishes. The command is sent first to the input plugins, to eventually update their state. Then, the command is sent to the MediaDescriptor unit of the first plugin loaded, the transport plugin. This unit sends the command to its plugin, then propagates it to the next direct plugins, until the last plugins in the chain received the command.

The movie is now playing, and subsequent stop, play or pause may be issued at will. A command that will not change the state of the system is discarded (e.g. a stop command when the system is already stopped). The system can not go from the stopped state to the pause state. The system has a current playing time parameter, so play commands changing the current playing position can be issued. The system has also a playing speed, that can be fractional and also negative. A playing speed of one means normal playing, a fast forwarding effect can be obtained by playing the movie with increased speed (if the media plugins support this operation). A playing speed less then zero means reverse playing, that can be slow, normal, or fast, depending on the absolute value of the playing speed parameter.

The exiting state can be reached from any other state. This is also the final state of the player, it is not possible to go to other states from here. In this state, the plugins are stopped and unloaded. First, the input plugins are destroyed and then unloaded. Next, the exit command is issued to the first MediaDescriptor unit, that corresponds to the transport plugin. This unit stops its corresponding plugin, then issues the command to the next MediaDescriptor units, that will do the same. When the function issuing the command to the next MediaDescriptor units returns, the current MediaDescriptor destroys the next MediaDescriptor units, destroys its associated plugin and then returns. At the end, the MovieDescriptor destroys the MediaDescriptor unit associated with the first plugin (the transport plugin).

4.2. Plugins

A plugin for our player is a dynamic linking library that contains a class extending the generic plugin class defined in the main program. The plugin should contain more a C function that returns an instance of the plugin class, if the type of plugin corresponds to the type of plugin that it is wanted to be loaded.

The class should contains several member functions:

- The `IsYourMedia` member function has as input parameter the `MediaDescriptor` unit containing the information about what kind of plugin it is wanted to be loaded. The function should return an integer that states the degree of suitability of

that plugin with what it is desired to be loaded. A value of zero states that the plugin is not fit at all.

- The `Initialize4Loading` function should verify that all the conditions for the plugin to run properly are met (if the plugin depends on specific hardware to run, here is the place to verify that the hardware exists). If the running conditions are not met, the function should return a negative value.
- The `Initialize4Running` function initializes the plugin for use. When this function is called, it means that this plugin was already selected to run, from all the other available plugins. This function should initialize the current plugin and prepare everything for receiving data, processing it and sending it to next plugins.
- The `DestroyPlugin` function should prepare the plugin for unloading
- The `OnCommandReceived` function is called by the plugin's associated `MediaDescriptor` unit to announce a system state change. The plugin should take whatever actions it finds appropriate.
- The `OnDataReceived` function is called when the plugin received data from its previous plugin. The function specifies a data field and an information field. The two plugins involved in the data transfer should have an agreement upon the format and significance of the information field.
- The `SendData` function is called by the plugin in order to send data to on of its next plugins. A generic implementation is provided by the generic plugin class. The same data and information fields are specified.

4.3. Tools

We have considered 4 tools in the current architecture, but further needs may add more.

One of the most important tools is the `PluginsManager`. This tool is responsible for loading one or all plugins that meet certain criteria. First, all the plugins that are found in the plugin directory are loaded. Then, the function returning the plugin class instance is called. The plugins for which this function returns `NULL` are discarded. For the remaining plugins, the `Initialize4Loading` is called, and all plugins for which this function fails, are discarded. Finally, `IsYourMedia` functions is called. If we have to load a single plugin, we select the one for which this function returned the maximum result, if we have to load all plugins that fit, we load all plugins for which this function returned a result greater than zero.

Another important plugin is the buffering data pipe between two plugins. This is implemented as a circular buffer with a producer (the sending plugin) and a consumer (the receiving plugin). There are defined 3 possible consuming methods:

1. A pull method, where the consumer explicitly calls a `consume` function that returns a buffering element, consisting of a data part and an information part. The `consume` function blocks if there are no elements in the buffering system, until the producing plugin produces an element.
2. A push method, where a consumer callback linked to the `OnDataReceived` function is called when the buffering system is full.
3. A timestamp based method, where a consumer callback linked to the `OnDataReceived` function is called when the time registered in the timestamp associated with an element has come.

The buffering system allows several buffering instances to have common timestamp basis and delivery mechanism, to assure proper synchronization when needed. There is also embedded mechanism for updating the reference clock based upon received PCRs.

4.4. The software only decoding scenario

In this scenario we analyze the plugins creation and command flow for a player receiving an RTSP URL of an MPEG-2 transport stream. The decoding is performed entirely in software, and there are separate plugins for transport, demultiplexing, audio and video decoding, audio and video rendering.

When the RTSP URL is received, the MovieDescriptor unit is creating a MediaDescriptor unit and fills it with information (URL and transport plugin type). The PluginsManager tool is called to load the best available transport plugin, that is capable of opening an RTSP URL. We suppose there is only one such plugin, and it is loaded. The Initialize4Running function for this plugin will be called. Inside the function, the plugin tries to start the streaming of the MPEG-2 transport stream. It will send an RTSP DESCRIBE request to the server specified in the URL. If the response is OK and initialization data is received, the plugin may fill some of the static information fields from the MovieDescriptor unit. We suppose that the server is specifying in an SDP [6] attachment to the DESCRIBE request that it accepts unicast connections for multiplexed MPEG-2 transport stream, RTP-encapsulated [3], [4]. The plugin will issue a SETUP request for the specified stream. At this point the plugin knows that is going to receive a multiplexed MPEG-2 transport, that has to be demultiplexed, after it is extracted from the RTP encapsulation. The plugin fills in a MediaDescriptor unit for loading a demultiplexer plugin, capable of demultiplexing MPEG-2 (the media type is MPEG-2, plugin type is demultiplexing). Then the Initialize4Running function returns, and the associated MediaDescriptor function that propagates the initialization command will try to load a demultiplexing plugin. We suppose that there are two demultiplexing plugins available, but one needs specific hardware that is not available on this platform, so its Initialize4Loading function will return a negative value. The other demultiplexing plugin is loaded. We can assume that it is not able to decode the video and audio elementary streams, so it will also fill two MediaDescriptor units, having the decoding plugin type, and audio respective video media types. After the Initialize4Running function of the demultiplexing plugin returns, its associate MediaDescriptor unit will load the two decoding plugins. These two plugins will each load in the same manner corresponding audio and video rendering plugins.

When the whole chain finishes the initialization process, the initial function called by the MovieDescriptor unit returns. Here the unit will send the play command to the first MediaDescriptor, that will propagate it through the whole chain.

4.5. The hardware aided decoding scenario

In this scenario we analyze the plugins creation and command flow for a player receiving a file URL of an MPEG-2 transport stream. The decoding is performed entirely in hardware, from demultiplexing to rendering, and there is one transport plugin that is able to read the stream from the file and a plugin that is able to initialize the hardware decoder and to feed it with an MPEG-2 transport stream.

When the file URL is received, the MovieDescriptor unit is creating a MediaDescriptor unit and fills it with information (path and transport plugin type).

The PluginsManager tool is called to load the best available transport plugin, that is capable of opening that file. We suppose there is only one such plugin, and it is loaded. The Initialize4Running function for this plugin will be called. Inside the function, the plugin opens the file, and it discovers an MPEG-2 transport stream. It then fills a MediaDescriptor unit with information to load a MPEG-2 demultiplexing plugin. We suppose that the PluginsManager function finds two available demultiplexing plugins, one that uses the available hardware decoder and the other that is fully in software, and both are able to operate. The plugin that uses the hardware decoder should return a higher value of the IsYourMedia function in order to be loaded. This plugin is loaded and initialized. In its Initialize4Running function the hardware card should be initialized. This plugin has the functionality of the whole chain, so no further plugins are needed, so no further MediaDescriptor units are initialized and the Initialize4Running function returns. After the play command is issued, the transport plugin will read data from file and send it to the demultiplexing plugin, that will forward it to the hardware card.

5. Conclusions

In this paper we have described our system for streaming high-quality multimedia content. The architecture and implementation issues of our player were extensively presented. We are considering an open Lintel architecture, that allows our player to be easily extended with new features using plugins. Thus, new types of decoders, renderers and new transport means can be added without changing the whole program.

To test our player we have implemented a GTK GUI user input plugin, an RTSP/RTP/SDP transport plugin, a file transport plugin, an MPEG-2 demultiplexing plugin, MPEG-2 elementary video decoder plugin, MPEG audio decoder plugin, video rendering plugin (frame based) and audio rendering plugin. We show how all these plugins interoperate together, how they are loaded and how data and commands propagate through the whole chain.

6. References

- [1] Nokia Media Terminal, <http://www.nokia.com/multimedia/mediaterminal.html>
- [2] Florin Lohan, Prakash Sastry, Irek Defée, "Broadband network Set-Top Box System", Proceedings of PROMS 2000, pp. 257-263
- [3] H. Schulzrinne et al. "RTP: A transport protocol for real-time applications", Internet draft, Internet Engineering Task Force (IETF), July 14, 2000.
- [4] D. Hoffman et al. "RTP Payload Format for MPEG1/MPEG2 Video", Technical Report RFC 2250, Internet Engineering Task Force (IETF), Audio/Video Transport, January 1998
- [5] Schulzrinne et al. "Real time streaming protocol (RTSP)", Technical Report RFC 2326, Internet Engineering Task Force (IETF), Network Working Group, April 1998.
- [6] M. Handley et al. "Session description protocol (SDP)", Technical Report RFC 2327, Internet Engineering Task Force (IETF), Network Working Group, April 1998.
- [7] Barry G. Haskell, Atul Puri, Arun N. Netravali, "Digital Video: An Introduction to MPEG-2", Chapman & Hall 1997
- [8] "ISO-IEC 13818" - MPEG-2 standard, <http://www.iso.org>
- [9] Roberto Castagno, Serkan Kiranyaz, Florin Lohan, Irek Defée, "An Architecture Based on IETF Protocols for the Transport of MPEG-4 Content over the Internet", CD-ROM Proceedings of ICME 2000.