

# A GUI Client for Video Conference based on C++, Cross-Platform, Feature-Enhanced and Multiple-User

Sergio B. Villas-Boas<sup>1</sup>, Lisandro Lovisoló<sup>2</sup>

1- Escola Politécnica DEL / UFRJ, 2 - DETEL – FEN – UERJ  
sbvb@poli.ufrj.br, lisandro@uerj.br

**Abstract**—This paper describes the complete software engineering (methodology, design, coding, testing and deployment) used in the development a GUI client for a video conference system. It is a general-purpose system, but it is specially useful and convenient to use it is for distance learning, remote consulting and eHealth (electronic-enhanced health systems).

It has the characteristics below: it is native cross-platform (Windows and Linux, with possibility of other platforms); it is feature-enhanced (adding to audio and video, extra features are included to the client; the main extra features are: white board, chat, application sharing and file sharing) it is open-source (based on C++ from scratch, added with some C++ libraries that are open-source and cross-platform). The source code of this software is available to the public.

The client's GUI software architecture is described in detail in this work. It has several software tiers, conceived to allow effectiveness in design, easiness of coding and testing, good maintainability and some GUI features (such as skin).

This paper also discusses some sophisticated software issues needed to implement the communication among the client's tiers and the client's threads, which require synchronization due to different incoming streams.

**Index Terms**—C++, native cross-platform, video conference, streaming

## I. INTRODUCTION

The usage of video conference systems nowadays is ubiquitous. The number of video conference users rises quickly. There are numerous video conference software products available, for the many needs and the many existing platforms. However so much has been done, there's still need to investigate how to improve flexibility and add features to video conference systems. That's because innovations on the information technology market happen often, and users demand to update software accordingly.

Since the standardization of the H.221 [19], in year 1993, many digital TV and video conference systems were proposed for ISDN networks. In recent years a lot of video conference applications over the Internet have been proposed and deployed. There are some free-to-use but proprietary (not open-source) products such as Skype [13] (with video since 2006) and Windows Life Messenger [18] (with video since 2005), that are very popular. In addition, there are also popular commercial video conference products such as the ones by Cisco, known as webex [2] and Cisco Unified videoconferencing [1]. The IPTV [5] is a commercial video conference product with client-server architecture, to name a few.

The information technology environment in the beginning of the 21st century, on the segment of personal computers, is experiencing the following transformation (among others). The operating system Windows is gradually decreasing its overwhelming market share, loosing it mainly to Linux and Macintosh. This transformation presses the software providers to develop cross-platform solutions, that is, software that works on Windows, Linux and Macintosh, as well as for other platforms.

This pressure was happening to the IPTV [5] in 2007. The GUI client of their video conference product was Windows-only (devel-

oped with dependency of Windows-only libraries). New customers demanded IPTV to offer their solution also to Linux. To meet this demand, a mixed group was formed, for a project named "MultiTV" [29]. The private firm IPTV, joined funds with the Brazilian government (under an economical subvention open call, managed by FINEP [4]). Expertise was called from universities [22], [23] to adapt their existing software product for the new requisite of being native and cross-platform. To obtain the required cross-platform feature, the software had to be re-written from scratch, in cross-platform C++ added with open-source libraries such as wxWidgets [27]. The MultiTV software development work initiated in September 2007 and successfully finished in January 2009. The result of the MultiTV project (the open-source code) is published [12].

This paper is organized as follows. In Section II, the architecture of the original IPTV system is discussed together with some of its most relevant features (like the cloud of servers). The software engineering process is described from Section III to Section VII. Section III discusses the software development method employed. Section IV discusses the MultiTV's software design. Section V discusses the relevant aspects of the coding process. Section VI discusses the testing. Section VII discusses the deployment of the software. Section VIII closes the paper with some conclusive statements.

## II. SOFTWARE ARCHITECTURE AND FEATURES

The original windows-only software product IPTV had many characteristics, and it was a requisite that the new version of it ("MultiTV") be 100% compatible with the previous version, so that both versions could co-exist. In this section, there's a description of the software architecture and its main features.

One main software architect issue is to be considered for any video conference system. It is to define the system's architecture to be one-to-one (peer-to-peer) or one-to-many (broadcast, multicast or several unicast) [26].

The IPTV software was designed mainly to fit distance learning, for that purpose it incorporates the features described following. There are several independent "channels". Users are allowed to join individually to each channel. There's control based on login/password to allow or deny a given user request to join a channel. When a user joins a channel, he/she is always watching it, and can be also "collaborating" (producing audio and video for others to watch). There are 2 possible actors for the user: "channel coordinator" and "normal" (non-coordinator). A channel has 1..\* joined users (when there are no joined users, the channel is destroyed). Any of the users joined to a channel can be one of the channel coordinators. A channel coordinator has access to more controls (features) than normal users do. For instance: a channel coordinator can set and reset the "is collaborating" flag of a user, while normal users can not.

To accomplish these features, a smart and lean solution was proposed: to define a client-server architecture, and use a custom protocol based on the well established irc protocol [6]. The irc

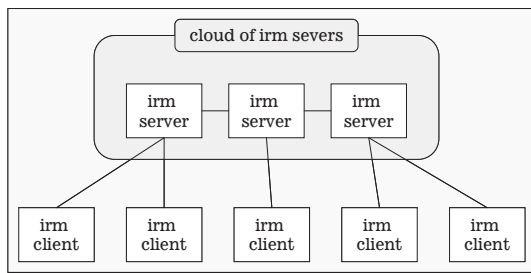


Fig. 1. IPTV and MultiTV Client-Server Software Architecture.

protocol is widely used since its birth in 1988 [7]. There exist several clouds of public irc servers, and many people exchange information using irc clients such as mirc [9]. During the design of the IPTV software, few modifications were introduced to the irc protocol, and a new derived protocol was proposed, named as irm (“m” of “media”). The irm protocol is basically a small extension of irc, that keeps compatibility with it, and allows also the exchange of “media packages”. These media packages are actually general purpose binary packages that carry information of audio, video and extended features of the video conference (such as chat and white board).

Therefore, the old IPTV client (and as a consequence the new MultiTV client) is an irm client (a superset and compatible irc client) that works with one server or a cloud of servers. The figure 1 shows the software architecture of the video conference system, showing a cloud of irm servers and several irm clients linked to the cloud.

The sole purpose of the server cloud is to make data flow, so that streams of video and audio reach the clients. There’s no data auditing, nor any data is saved in the servers.

The server cloud has no inherit contingency in its architecture, that is, if an incident happens during a video conference, (for instance, one of the servers in the cloud stop working), part of the clients are dropped of the channel.

The client-server architecture requires the server cloud to open one unicast stream connection with each client. This allows some feature improvements when compared to peer-to-peer architecture. For instance: multiple stream rates are allowed with a single media re-encoding in the server cloud. This benefits data flow optimization in the network.

The GUI client is responsible for capturing audio and video, encoding them using the selected codecs, and transmitting video, audio and extra information to the server. The GUI client also decodes the audio and video streams received from the server, as well as handling the extra information. The system is multi-collaborative, that is, 1..\* audio and video stream are allowed to be used at the same time. The system is multi-channel, that is, the server can handle several channels at the same time; there can exist as many as needed channels in the server; one channel is completely independent from other.

The GUI client works “per channel”. It is possible to run 2 or more GUI clients at the same time from the same computer, and each one works independently from the other. The irm server is multi-channel, that is, the server can handle several channels at the same time, and one channel does not interfere to some other channel.

#### A. IRM

As mentioned before, the irm protocol, derived from irc [6] is used for IPTV and for MultiTV. That is, both software are irm clients.

The irm protocol encapsulates audio and video bitstreams into the media package of irm. For the network, the client uses 2 types of

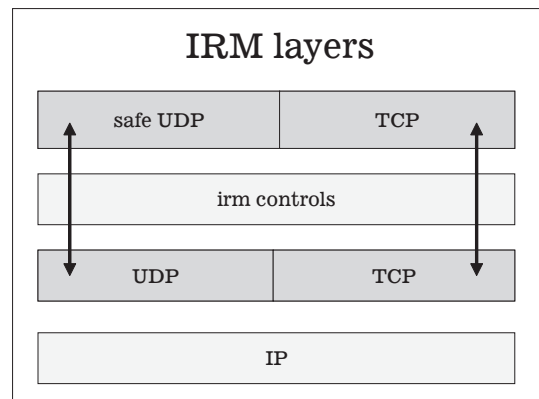


Fig. 2. IRM layers

connection

- TCP/IP.
- UDP, with package delivery assurance implemented over it, named “safe UDP”.

Safe UDP is ordinary UDP with a protocol on top of it that monitors the network and take corrective measures if package delivery fails. The safe UDP generally works better than TCP. The native mechanisms TCP for package delivery assurance use retransmission and bitrate control. These mechanisms don’t fit well for video transmission over public networks [26].

The irm protocol checks the available possibilities for the user’s network. If UDP is available, safe UDP is used. UDP might not be available (e.g. a firewall blocks it). If UDP is not available, but TCP is, than TCP is used. See figure 2.

#### B. Media encoding

In order to obtain maximum efficient in usage of bandwidth, the different media contents delivered must be encoded, that is compressed [24].

The delivered contents can be classified as follows.

- Video. Video content is encoded using open-source implementation of codec provided by the FFmpeg Library [3]. Most often the codec of choice is H.264 [20].
- Audio. Audio content is encoded using open-source implementation of codec provided by the FFmpeg Library [3]. Most often the codec of choice is Speex [14], which is based on the CELP [25].
- Application Sharing. The concept of “application sharing” means that a user’s desktop can be captured and sent as collaboration to a channel. By using this feature, any software running in an user’s computer can have its screen seen by all users in the channel. To get better performance for computer screens delivered to video-conference, a special codec was introduced. The “computer screen codec” works as following. It is a mix of DPCM+JPEG coding scheme, the same basis of most video coding tools. To begin, an entire frame of the desktop is sent. The next desktop image is compared with its predecessor at a pre-defined rate (of quite slow fps). Only the differences from previous frame are sent.

#### C. Video Renderization Strategies

When dealing with multiple video streams, one has to receive, decode and render the media. There are 2 strategies to accomplish it.

- “Push Video Renderization”. The software listens to the network interface, for each thread the incoming video packets are redirect to buffers. Each buffer decodes received video packets and render them to frames. Each thread is pushed to process by the received video packets. Such strategy is completely asynchronous. The buffer might become overload of packets, and in this case the solution is to drop one frame. As the video frames become ready, they are sent (“pushed”) to presentation.
- “Pull Video Renderization”. The software has an independent thread for each stream to be rendered. The presentation is set to a certain frame rate, and it requests a new frame from the thread when it is the time. When asked, the thread always gives a video frame to the presentation (that is, the presentation “pulls” from the rendering thread). The thread receives video frames from the network interface and processes them asynchronously from the presentation. If there’s no time to provide a new video frame to presentation, the previous frame is given again.

The “Pull Video Renderization” has some advantages over the “Push Video Renderization”. The “Push Video Renderization” has the ability to process different video frame rates. However, it does not allow independent tiers for rendering and presenting. Instead it requires a single code to implement both rendering and presenting. If CPU power sufficient for the frame rate defined, the renderization tier is works faster than presentation tier, and the user observers the video well. If CPU power not sufficient for the frame rate defined, the video frame rate in presentation diminishes, and the video turns out to be not smooth.

The “Pull Video Renderization” allows independent tiers for rendering and presenting. If CPU power sufficient for the frame rate defined, the renderization tier is works faster than presentation tier, and the user observers the video well. If CPU power not sufficient for the frame rate defined, eventually a video frame is lost, but the presentation tier keeps displaying with the the same rate, providing smooth video.

The MultiTV implemented the video rendering using the “Pull Media Renderization” strategy.

#### D. Audio and Video Synchronization

The approach of previous subsection can not be applied to audio. For video conference systems, audio is considered to have priority over video, because the loss of quality on audio may turn communication impossible.

Since audio have priority over video, video frames may be dropped but audio ones can never be dropped. Therefore, audio is decoded and played as soon as it arrives having a prioritized flow in the application.

If audio is not dropped and video is dropped, the stream loses audio and video synchronization. To avoid this, the following synchronization mechanism was implemented.

- 1) At a fixed rate of  $x$  (e.g. 15) times per second, the application thread will attempt to display the buffered video frames.
- 2) At this rate it senses the buffer-decoding thread of each video. This task verifies the existence of a new video frame and its time-stamp.
- 3) Priority is given to audio. That is, the audio is played whenever it arrives (at the correct timestamp) and video renderization attempts to follow the audio.
- 4) If the timestamp of the current video frame is closer to the the one of the current audio frame than the (not yet displayed) timestamp of the buffered video frame, the buffered video frame is dropped.

This mechanism makes the audio flow with priority and the renderization of video keeps trying to follow audio as well as possible.

### III. METHODOLOGY

The main objective of the MultiTV project was to re-write the existing software (IPTV) with the characteristic of being cross-platform. That required a “from scratch” approach, so that all Windows-only libraries such as MFC and directx had to be changed for cross-platform equivalents. Besides, the source code of the original software had been developed by IPTV team as a “first version”, that is, from a relatively unexperienced skill base. The experience gained from the real usage maintenance of the IPTV software, added by the expertise that came from the university allowed to write code with much better coding standards for the MultiTV software. Maintainability improvements also happened originated to the multi tier GUI architecture (see section “Design”).

The method used for software development was an agile method, a small variation of Scrum [11]. The first task was to build a development infra structure with the following features.

- 1) Setup a “data center”. We used one powerful computer and installed Ubuntu linux [16] on it. Over linux, we installed VMWare Server [17], and that allowed usage of as many virtual machines as we needed.
- 2) Install an other Ubuntu linux machine as virtual on the data center, that came to be known as “software engineering machine”.
- 3) In the “software engineering machine” setup a software source code repository. We used svn [15]. All members of the development team were given accounts to commit and checkout to the repository.
- 4) In “software engineering machine” setup bugtrack system. We used Mantis [8]. All members of the development team were given accounts on the bugtrack system. The clients of the project (owners of IPTV) were also given viewer access to the bugtrack system.
- 5) Setup another Ubuntu linux machine in the data center that came to be known as “linux build machine”. In this machine it was installed the gnu C++ compiler and all extra libraries used in the project. This machine was used to execute every day the scheduled task of checking out the entire code from the repository, rebuild it all, and report possible compilation errors to the team manager. After rebuilding all code, all automatic tests were executed, and its result was also sent to the manager.
- 6) Setup a Windows xp sp2 machine in the data center that came to be known as “windows build machine”. In this machine it was installed the Visual C++ 2005 and all extra libraries used in the project. This machine was used to execute every day the scheduled task of checking out the entire code from the repository, rebuild it all, and report possible compilation errors to the team manager. After rebuilding all code, all automatic tests were executed, and its result was also sent to the manager.
- 7) Setup the developer’s computers with dual boot (Windows and Linux), and all needed software (C++ compiler, extra libraries and tools).

The development team was composed by one development manager, 3 developers and one coordinator and one consultant. The coordinator’s task was to keep contact to the client, and provide the structural conditions to the team to work.

The client requested that the new software be a conceptual copy of the previous one, only being cross-platform. So, the requisites for the

development were defined as “the features of the existing software”.

The general plan for the project was to divide the total time of 15 months in 3 periods of 5 months, and use the periods as following.

- 1) Setup development environment, configure and test all extra libraries, develop test code of all key features for the system.
- 2) Integrate and test all features, one by one, into the product. By this time, a beta version must be available
- 3) Extra time to fine tune bugs, installation features and usability tests.

After the 1st period of 5 months, every 2 weeks, a new sprint (intermediate version) was developed and sent to the client for evaluation. The priority development tasks for the next sprint were negotiated after each delivered sprint. The low level tasks to be developed were introduced to the Mantis bugtrack system. The first sprint was delivered on April 8th, and the last sprint, number 22, was delivered on February 27th. All sprints are public and can be downloaded from MultiTV web page [29]. Every month, a development report was delivered to the client, who paid for another 1/15th of the project cost.

#### A. Bugtrack system

The bugtrack system works “issue oriented”. An “issue” (a task) is an indivisible piece of development effort, that is added and monitored using the bugtrack system. Each issue has an individual id number, and should be in one of the states shown in figure 3.

Each issue is assigned to a person. The manager assigns issues for himself and for the other developers. The team manager uses the bugtrack system to define who should do what, and sets priorities to the issues. The intermediate accomplishment reports delivered to the customer refer to the issues id’s. When the project is finished, the bugtrack system is given to the customer, storing the development history.

An issue is created in “new” state. The manager assigns it to a developer and changes state to “queued”. This is equivalent to saying “I request you to do this issue”. The developers should consult regularly the bugtrack system and see what the manager is requesting. When the developer is doing a given issue, he/she should change the state to “doing”. When it is finished, he/she should change it to “done”. The “done” state is also a queue to testing. Typically the tester is not the same person that did the issue. In MultiTV project, often the issues were tested by someone in the IPTV firm, outside the MultiTV team. If the test is ok, the issue goes to “tested”, to be latter sent to “closed”. If the test is not ok, the issue goes to “queued”, and assigned again to a developer.

The usage of the bugtrack system fits well for agile software development, when is typical that the priorities in development change fast. The customer expresses his/her will in the regular meetings, and this is converted to issues and its priorities. The development team is regularly steered by the customer to do what is most important at any time.

Some developers often work remotely, and the bugtrack system turns up to be very convenient. The developer logs to the bugtrack system, reads what he/she is expected to do and updates codes using the repository.

## IV. DESIGN

The design of the MultiTV client, more than keeping compatibility and implement all features of the old IPTV software, incorporated the new features as below.

- MultiTV client should isolate all possible parts of the code from the intricate user-interface-related technologic aspects of

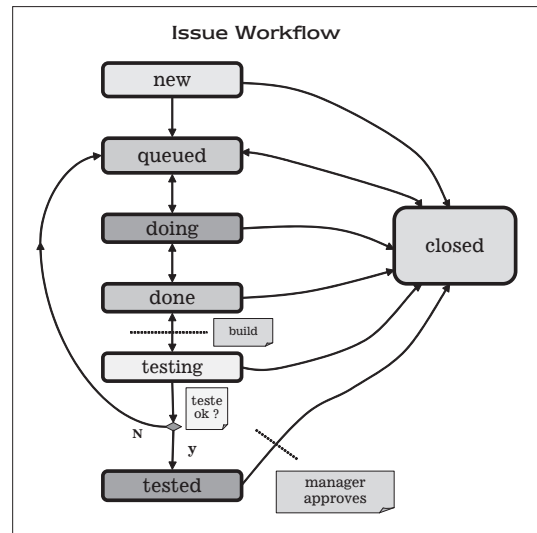


Fig. 3. Issue workflow in bug track system

the client. We chose to use wxWidgets [27] as the cross-platform GUI library. As it is explained below, the MultiTV was designed in a 5 tier architecture; tiers 1, 2 and 3 don’t have dependency of the the cross-platform GUI library of our choice (wxWidgets). It is possible write another client using tiers 1 to 3, and some other option for user interface different from the choice made.

- MultiTV should be isolated from the appearance, that is, be skinable. It should be easy to write a new skin to MultiTV, and the software should be able to use the a new skin without restarting the software. To implement this, we designed an abstract class in C++ to represent the skin. The code to implement the abstract methods of skin class exist in tier 5 (tier of skin). This tier is physically is a dll file (if Windows) or so file (if Linux). MultiTV can not execute if there’s no skin for it. After the software is running, by clicking to the interface the user can choose some other skin, even one that didn’t exist when the software started to run. The skin is loaded in execution time, and can be easily changed while the software running. A developer can write a new skin by writing a different implementation of the skin abstract class.
- MultiTV should be isolated from the language (idiom) shown in the screen, and be compatible with Unicode. That was accomplished by using the internationalization procedures recommended by wxWidgets.

To achieve the above mentioned requisites, the MultiTV was designed to have 5 tiers, as shown in figure 4.

Below it is explained the purpose of each tier.

- 1) This tier is named “helper”. It is the tier for very basic and general purpose features. This tier contains helper libraries, described below.
  - **Shared.** The Shared library was originally developed by the project’s client (the software owner). The helper features of that cross-platform C++ library include advanced TCP and UDP control, threads and mutex control, among other features. It was absolutely necessary that MultiTV

<sup>1</sup>In fact, during the development we used this characteristic of the software design to write another client based on tiers 1 to 3, with line command interface, for test and debug purposes.

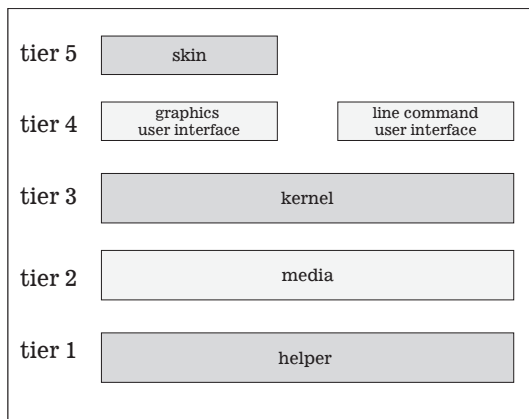


Fig. 4. The 5 tier MultiTV model

software depended on the shared library because the server uses the same library for the low level network communication.

- **FFMPEG [3].** The FFMPEG library main purpose is to implement the codecs to code and decode audio and video.
  - **Chat.** The chat library is contains code to handle a few functions for chat; this library was reused from the original IPTV software.
  - **Stub.** The stub library is contains code to handle a few functions for network handling permissions of users in the video conference channel; this library was reused from the original IPTV software.
  - **VBLib.** The VBLib library is distributed within a bundle of libraries named sbVBLibs [28]. It is a general purpose and cross-platform library in C++. This library was used for encapsulating the platform differences in loading dll's (if Windows) or so's (if Linux) of explicit link. This feature is used in MultiTV project to load the software's skin. VBLib was also useful because of the VBString class. The standard C++ string class can't be used in argument of functions that are loaded as explicit dll's or so's, so VBLib's class VBString was used instead.
- 2) This tier is named "media". It provides information partially processed to the tier above (named "kernel"). The media tier provides for the kernel tier, for each stream, decoded frames of audio and video that were received from the network. It also receives from kernel tier uncompressed frames of audio and video and sends them to be compressed and sent as collaboration streams. Code to handle user permissions and thread-related issues of media are defined of in the media tier.
  - 3) This tier is named "kernel". This tier works like a event-driven kernel of an operating system, processing irm events, stimulated by the tier above, that work like applications running in the operating system. When someone used the application, there are various use-cases like: enter channel, leave channel, set and reset "is collaborating" flag, start chat, end chat, send string to chat, etc. These use cases produce events that are handled by the kernel tier. The system is oriented to manipulate the virtual entities that exist in the server, like "channel" and "user". The condition of these entities can vary because some other user changed it, or because the network failed. All the processing of these cases happens in the kernel tier.
- The multi-tier architecture of MultiTV has an important soft-

ware maintainability aspect that is improved over the old IPTV software. When the user clicks or types something, he/she produces GUI events that have to be handled by the GUI software. In MultiTV, the GUI events are implemented in the "user interface" tier, while the irm events are implemented in the "kernel" tier. In IPTV software, the equivalent of these 2 tiers were intermingled, requiring code to be replicated equally in several places of the source file.

- 4) This tier is named "user interface". For MultiTV project, 2 applications were developed in this tier.
  - a) The main application, with Graphics User Interface (GUI). This application depended on the libraries below.
    - **wxWidgets [27].** wxWidgets is a C++ based library that allows cross-platform GUI development. By using this library, the GUI client is native an cross-platform; for each of the two platforms of interest - Windows and Linux - a build is required, producing a platform specific application; this is done from the same source code.
    - **OpenGL [10].** The OpenGL library is a well established industry foundation for high performance graphics. wxWidgets admits easily integration with OpenGL. The GUI software has a reason to use OpenGL is to solve the problem of fast resize of the display video frame. GUI software looks like a rectangle (see figure 5, where it appears maximized). The user expects to be able to resize the GUI software to any desired size. This means that the actual frame being displayed will have a size known in execution time. This display video frame has a refresh rate of the video conference (typically 15 fps). To resize what is in memory to the custom-defined size can be costly. We chose to solve this problem using OpenGL. An OpenGL rectangle is placed in perfect alignment to the 3D camera, and the rectangle's OpenGL texture is updated with the same frame rate of the video conference. By doing so, if the user's computer has OpenGL compatible hardware acceleration (this is quite common nowadays), that will used to help display quickly the resized video frames. The OpenGL library is not thread-safe. MultiTV software uses it to display simultaneously all collaborating video streams, that are handled in separated threads. Extra code had to be introduced in the main thread to emulate the thread safeness above OpenGL, thus allowing the threads handling the video streams to produce visible displays.
    - **OpenAL [21].** The OpenAL is a straightforward solution to produce cross-platform audio output and input. Audio frames can be easily input and output using OpenAL.
  - b) A test / debug application, with line command interface. This application is a very simple one, that converts to chars most signals from kernel tier.
- 5) This tier is named "skin". To provide options for the GUI client's appearance we chose to implement a plugin for appearance, that is the skin tier. At least one skin must exist, otherwise the user interface can't show itself. From one skin, the user can choose another skin that exists. That can be done without restarting the GUI client. Physically the skin is a dll (if Windows) or a so (if Linux). The skin tier is an implementation



Fig. 5. MultiTV GUI client in Linux, with 3 streams collaborating

of the methods of the abstract skin class.

## V. CODING

The code itself, as well as the information in the bugtrack system [8] were 100% done thinking and using the English language. That was done following strong recommendation of the project's coordinator (the first author of this paper), with enthusiastic support of the development team (remark that this development was done in Brazil, that has not English as its mother language). The customer [5] also requested the code to be done in English language.

Internationalization techniques were used to display any final user's language in the screen; since unicode compatible tools were used, any language can be exhibited in the screen, including eastern languages such as Japanese or Chinese.

The coding process took place throughout all 15 months of MultiTV project. The physical place the developers used to do most of the coding work was a laboratory in UFRJ [22]. The developers used computers to write code and committed the code to the project's repository in the laboratory's datacenter. Some code was developed in the developer's homes, in their private computers, and committed identically.

Most of the developers were students of computer engineer or similar undergraduate courses. These students were hired as part time developers, using the MultiTV's money provided by the client [5]. The development manager was a full time software engineer.

It was convenient to do most of the coding work inside the university, where the students spend their daytime, switching time with university tasks.

After the delivery of one most of the 22 sprints, the MultiTV's coordinator, the development manager and someone else of the development team went for a meeting with he client to discuss the priorities for the new sprint.

## VI. TESTING

There was 2 types of tests: automatic and manual tests.

Some of the automatic tests that were developed for irm features in the kernel tier, like the ones shown below.

- create user
- delete user
- send and receive a string in a chat

Since MultiTV software is GUI, most of the tests were manual and usability tests. These tests were done by the development team, and than confirmed by client's expert user. Should some GUI interface or usability feature be not satisfactory for the client (customer), that was reported in one of the inter-sprint meetings, and that became an issue to be solved for the next sprint.

The general aspects most tested in manual tests are listed below.

- Compatibility with existing irm server; the new version should be possible to use at the same time with the previous version of the client.
- Good usability of the GUI interface.
- work well in both Windows and Linux
- Do not consume too much CPU power, so that relatively slow computers could use the software.

### A. Delay

There was no formal specification for how much the video conference delay should be. But the delay of a video conference system is a consequence of the system architecture, the buffers and the signal processing.

Tests showed delay of 0,3 seconds.

### B. Jitter

Jitter is the variation of the delay for delivery of packets. The jitter level is a characteristic of a network. Tests were made using video conference in a satellite based network. The delay remained the same (around 0,3s). The high jitter of the satellite based network produced no noticeable effects.

### C. Audio quality

Audio packets have priority over video packets. This means that when relatively narrow bandwidth is available, the audio remains as intelligible as possible, while the video loses quality. To prioritize audio over video is used in most video conference systems, including the one presented in this paper.

The MultiTV software allows selection of bandwidth to be used. The narrowest bandwidth that can be selected is 33K Bps.

The tests using speex codec showed that users can have audio quite intelligible conversation using any bandwidth equal or greater 33KBps.

### D. Video quality

The video tests were done with frame rate of 15 fps, and frame resolution of  $360 \times 240$ .

The video quality is highly affected by the selected bandwidth. Our tests used H264 codec. Using 33KBps, the quality is very poor, and the video becomes extremely boxed.

As expected, the video quality improves as bandwidth increases. For bandwidth of 256K, the produces video with quality comparable to an analog VHS videocassette.

Tests were made with bandwidth of 512K and 1M Bps. The video quality improves with this increase, but the improvement is not very noticeable.

### E. Consumption of CPU power

The CPU consumption tests were done with frame rate of 15 fps, and frame resolution of  $360 \times 240$ .

In this subsection, it is presented the CPU consumption in some cases. The test computer is has CPU Intel core 2 duo, 1G RAM, and video card with OpenGL implemented in hardware.

In the tests, "encoding" means that the local camera is being captured, encoded and sent as stream to the video conference. "decoding" means that a video stream is being received from the network, decoded and presented in the GUI software.

Operating System	1 encoding 1 decoding	0 encoding 2 decoding
Windows	37%	19%
Linux	33%	17%

As expected, results show that encoding requires considerable CPU power. Linux version worked slightly better than windows version.

#### F. Compatibility

The Windows version of MultiTV works on the most popular versions of Windows. We tested it to work on Windows 98, Windows XP and Windows Vista.

The Linux version of MultiTV works fine on Ubuntu Linux. Unfortunately there's nothing near a "standard Linux", that is, a flavor of Linux that is considered to be the most important one, in the sense that a software like MultiTV should be primarily developed and tested to it. Other than Ubuntu Linux, we tried RedHat Linux, and it worked. It is known that in some old distributions of Linux MultiTV does not work. From the beginning of MultiTV project, the requisite was that we choose one decent and well known flavor of Linux, and develop the software that runs well on it. We choose Ubuntu, for its good usability and well acceptance by the market. It is not the scope of this work to attempt to solve the numerous compatibility issues among the Linux flavors.

#### G. Scalability

This work is about the development of the GUI client. The issues concerning how to scale the video conference to numerous clients concern about the server.

It is beyond the scope of this work to discuss strategies to produce video conferencing for numerous clients. So no tests were produced in this work concerning scalability of MultiTV.

In our tests, we used up to 4 collaborating GUI clients connected to a single server (only one server in the cloud). The host server is a quadricore xeon Intel CPU with 8G RAM running Ubuntu Linux and VMWare server; running over it in virtual mode, the actual MultiTV server is another Ubuntu Linux with 1G RAM. The server CPU usage remained very low during all tests.

### VII. DEPLOYMENT

The deployment requisites for the MultiTV are listed below.

- Develop an easy to use installer for the Windows version of it.
- Solve compatibility issues for a few flavors of Linux (Ubuntu, RedHat, etc.) where MultiTV was expected to run.
- The client decided to use a customized version of ffmpeg [3]. That required some special adaptations in the code to avoid conflict with the standard ffmpeg that could have been installed in the user's computer.

To regularly backup of the virtual machines in the datacenter was also considered a deployment task of the team. When the project ended, the client (customer) received the datacenter computer with all virtual machines installed on it.

### VIII. CONCLUSION

The private firm IPTV [5], who owns a software product for video conference, received demands from its customers to offer video conference as cross-platform (Windows and Linux). Theirs original product was Windows-only, written in C++ with deep rooted dependency on Windows-only libraries and technologies.

To attend customers demand, a mixed group was formed named MultiTV [29], calling expertise from universities [22], [23], partially funded by the Brazilian government, by Finep [4]. The MultiTV

group re-wrote from scratch the GUI client for video conference, using C++ added with open-source cross-platform libraries. The new MultiTV client kept compatibility with the previous version, so that both versions can be used simultaneously. The software development work initiated in September 2007 and successfully finished in February 2009. The result of the MultiTV project (the open-source code) is published [12].

This paper described the complete software engineering (methodology, design, coding, testing and deployment) used to develop the MultiTV software.

### IX. ACKNOWLEDGEMENTS

We would like to acknowledge FINEP [4], for funding the MultiTV project, reported in this paper.

We would also like to acknowledge the excellent development work done by the MultiTV team, whose members are listed below. Guilherme C. De Lello (development manager) André Cotrim (developer), Daniel D. Gouvea (developer) Vinicius H. dos Santos (developer), Ivan C. S. Lopes (developer), Daniel Dantas (developer).

### REFERENCES

- [1] Cisco unified videoconferencing. <http://www.cisco.com/en/US/products/hw/video/ps1870/>.
- [2] Cisco webex. <http://www.webex.com/>.
- [3] Ffmpeg library. <http://ffmpeg.org/>.
- [4] Finep home page. <http://www.finep.gov.br/>.
- [5] Iptv home page. <http://www.ip.tv>.
- [6] Irc. <http://irchelp.org/>.
- [7] Irc history. [http://www.irc.org/history\\_docs/jarkko.html](http://www.irc.org/history_docs/jarkko.html).
- [8] Mantis bug track system. <http://www.mantisbt.org/>.
- [9] Mirc. <http://www.mirc.com/>.
- [10] Opengl. <http://www.opengl.org/>.
- [11] Scrum. <http://www.scrumalliance.org/>.
- [12] Sergio barbosa villas-boas. <http://code.google.com/p/multitv/>.
- [13] Skype home page. <http://www.skype.com>.
- [14] Speex project home page. <http://www.speex.org/>.
- [15] Subversion. <http://svn.collab.net/>.
- [16] Ubuntu linux. <http://www.ubuntu.com/>.
- [17] Vmware server. <http://www.vmware.com/products/server/>.
- [18] Windows live messenger. <http://people.live.com>.
- [19] International Telecommunication Union. Line transmission of non-telephone signals - frame structure for a 64 to 1920 kbit/s channel in audiovisual teleservices - ITU-T Recommendation H.221. March 1993.
- [20] International Telecommunication Union. Advanced video coding for generic audiovisual services - ITU-T Recommendation H.264. May 2003.
- [21] Creative Labs. Openal. <http://connect.creativelabs.com/openal/>.
- [22] Federal University of Rio de Janeiro. Ufrj. <http://www.ufrj.br/>.
- [23] University of Rio de Janeiro State. Uerj. <http://www.uerj.br/>.
- [24] K. Sayood. *Introduction to Data Compression*. Morgan Kauffman Publishers, San Francisco, USA, 2 edition, 2000.
- [25] M. R. Schroeder and B. S. Atal. Code-excited linear prediction (celp): high-quality speech at very low bit rates. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 10:937 – 940, 1985.
- [26] Wes Simpson. *Video Over IP: A Practical Guide to Technology and Applications*. Media Technology Professional Series. Focal Press, 1st edition, 2006.
- [27] Julian Smart. wxwidgets: C++ library for cross-platform gui. <http://www.wxwidgets.org/>, since 1992.
- [28] Sergio B. Villas-Boas. sbvbblibs: Cross-platforms c++ libraries by sbvb. <http://code.google.com/p/sbvbblibs/>, August 2008.
- [29] Sergio B. Villas-Boas. Multitv project home page. <http://www.svbv.com.br/cgi-bin/index.cgi?p=18>, June 2009.