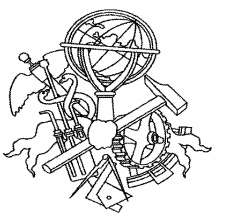
SOLUÇÃO DOMÉSTICA PARA TELEVISÃO DIGITAL

Rui Carlos Neves da Silva



Departamento de Engenharia Electrotécnica

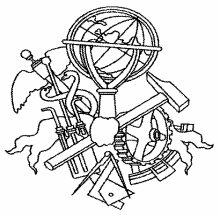
Instituto Superior de Engenharia do Porto

2008

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Rui Carlos Neves da Silva, Nº1030381, 1030381@isep.ipp.pt

Orientação científica: João Miguel Queirós Magno Leitão, jml@isep.ipp.pt



Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

7 de Dezembro de 2008

Agradecimentos

Uma vez que a elaboração de uma Tese de mestrado implica, acima de tudo, aplicação, trabalho e disponibilidade, não posso deixar de agradecer a todos os amigos, familiares e, principalmente, namorada, que ficaram privados da minha companhia para que eu me pudesse dedicar ao desenvolvimento da Tese aqui apresentada.

Ao Professor Doutor João Miguel Queirós Magno Leitão, orientador da dissertação, agradeço o apoio, a disponibilidade e a criatividade, que se revelaram contribuições valiosas para o trabalho desenvolvido.

A todos os colegas e professores que, ao longo de cinco anos, me ajudaram a adquirir conhecimentos importantes e necessários para a elaboração desta Tese.

Resumo

A *Internet Protocol Televison* (IPTV) é uma tecnologia desconhecida da maioria dos portugueses, embora já seja usada por alguns cidadãos. Esse desconhecimento e a necessidade de conhecer a arquitectura da IPTV, levaram a um estudo de uma rede IPTV, apresentado neste documento, onde é também demonstrado um protótipo de uma solução doméstica de IPTV. Uma solução doméstica típica de IPTV utiliza um *HomeGateway* para receber os vários tipos de dados, e uma *box* junto a cada televisor para tornar possível visualizar numa TV os conteúdos recebidos. A solução aqui proposta, comparativamente às soluções oferecidas no mercado, inclui um servidor na casa de cada cliente. Este servidor é colocado entre as *boxes* de cada TV e a *HomeGateway* e é o responsável por receber conteúdos vídeo do servidor central do serviço e enviá-los para as *boxes*.

O protótipo implementado divide-se em duas partes: servidor (o servidor local) e cliente (as *boxes*). O servidor foi desenvolvido é capaz de receber vários pedidos de ligação e geri-los independentemente; para o cliente implementaram-se as seguintes funcionalidades, usando apenas *software* livre: interacção por parte do utilizador através de um comando, alteração do volume, corte do som, paragem em tempo real, guia de programação, *Video on Demand* (VoD), agendamento de gravação e reprodução de conteúdos gravados.

Na elaboração deste protótipo foi usado um *media player,* o VLC, para se poder ler e enviar vídeos através de uma rede IP. Para o desenvolvimento desta aplicação foi usada a linguagem de programação C e alguns pacotes de bibliotecas auxiliares que são: *Libvlc*, para a criação de uma janela VLC; Xlib, para permitir aceder ao servidor de janelas X; GTK, para o desenvolvimento do interface gráfico; Libxml, para permitir parcelar um ficheiro XMLTV.

Para testar o protótipo desenvolvido foi utilizado um repositório de conteúdos local.

Palavras-Chave

IPTV, Linux, *shell-script*, Apache, VideoLan, Xlib, GTK, libxml, XMLTV, LIRC.

Abstract

The IPTV is a technology unknown by most of the portuguese, although being used by some. This unknowledge and the need to know the architecture of IPTV, led to a study of an IPTV network, herein presented, In this document is also demonstrated a prototype of an IPTV domestic solution. This solution, compared to the others offered in market, includes a server in each customer's home. This server is placed between the TV boxes and the HomeGateway and is responsable to send to the boxes the video content received from the service's central server.

The implemented prototype is divided into two parts: server (the local server) and client (the boxes). For the server it was developed an TCP / IP server capable of receiving several connect requests and independently manage them; using only free software, for the client was implemented the following features: user interaction through a remote control, volume change, mute, real time stop, programming guide, VoD, schedule recording and recorded content playback.

To be able to read and send videos through an IP network, in this prototype was used a media player, the VLC. For the development of this application was used C programming language and some assistant library packages, who are: Libvlc to create a window VLC; Xlib, to allow access to the X window server; GTK, for the graphical interface development; Libxml, to extract info from a XMLTV file.

In order to test the developed prototype it was used a local content repository.

Keywords

IPTV, Linux, *shell-script*, Apache, VideoLan, Xlib, GTK, libxml, XMLTV, LIRC.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ADSL – Asymmetric Digital Subscriber Line

ASM – Any Source Multicast

CO – Central Office

DSLAM – DSL Acess Multiplexer

ECM – Entitlement Control Messages

EMM – Entitlement Management Messages

ES – Elementary Stream

GC – Graphics Context

GPL – General Public License

HD – High Definition

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IGMP – Internet Group Membership Protocol

IP – Internet Protocol

IPTV – Internet Protocol Televison

ISP – Internet Service Provider

LEO – Local End Office

MPEG – Moving Picture Experts Group

MTU – Maximum Transfer Unit

OSI – Open System Interconnection

PIM – Protocol Independent Multicast

PS – Program Stream

PSI – Program Specific Information

PT – Portugal Telecom

RT – Remote Terminal

RTP – Real-time Transportation Protocol

SHE – Super Head End

SI – Service Information

SSH – Secure Shell

SSM – Source Specific Multicast

STB – Set-top Box

TCP – Transmission Control Protocol

TCP/IP – Transmission Control Protocol/Internet Protocol

TS – Transport Stream

UDP – User Datagram Protocol

VLS – Video Lan Server

VLC – Video Lan Client

VOD – Vídeo On Demand

VoIP – Voice on IP

VSO – Vídeo Serving Office

1. INTRODUÇÃO

Este documento apresenta o trabalho realizado para a disciplina de Tese e Dissertação, do Mestrado em Engenharia Electrótecnica e de Computadores, mais especificamente para a Tese "Solução Doméstica para TV Digital".

1.1. CONTEXTUALIZAÇÃO

Este projecto surgiu do desejo de realizar um trabalho que permitisse conhecer e trabalhar numa tecnologia inovadora. Assim, a escolha deste tema implicaria o estudo de uma tecnologia recente, com conceitos novos, que pertence a uma área atractiva como é a das redes. Quando esta Tese foi escolhida, as soluções de IPTV ainda não estavam muito divulgadas entre a população portuguesa e ainda demonstravam algumas limitações, limitações essas, que com um maior investimento, podiam ser emendadas. Entretanto, devido a um maior investimento, principalmente na publicidade, a MEO, da Portugal Telecom (PT), ganhou força sendo actualmente a solução de IPTV com mais sucesso em Portugal. Assim, decidiu-se criar uma solução diferente, mas capaz de tirar o máximo partido desta tecnologia mesmo com recursos limitados.

1.2. OBJECTIVOS

Devido à fraca divulgação e documentação desta tecnologia, principalmente em Portugal, primariamente será objectivo o conhecimento dos protocolos por detrás de uma rede IPTV, assim como a constituição e arquitectura de uma rede destas e, com este conhecimento, propor uma solução alternativa.

Uma vez proposta uma solução, desenvolver um protótipo capaz de avaliar a sua eficácia.

Por fim, pretende-se acrescentar à solução básica criada, quer funcionalidades existentes quer não existentes.

É também objectivo que o protótipo desenvolvido consiga ser operado num *hardware* de baixos recursos.

1.3. CALENDARIZAÇÃO

Tendo em vista o cumprimento dos objectivos propostos, foi necessário dividir a criação desta solução por etapas. A forma como essas etapas estão definidas pode ser visualizada no Diagrama de Gant presente na Tabela 1, onde se realçam as fases de: estudo das soluções existentes no mercado, estudo das redes IPTV, projecto do protótipo da solução, desenvolvimento dos vários módulos da solução e testes/validação.

1.4. ORGANIZAÇÃO DO RELATÓRIO

No primeiro capítulo deste documento é feita uma introdução, na qual são apresentados os objectivos e a organização do relatório.

Uma vez que a elaboração desta Tese envolveu fases de estudo e fases de desenvolvimento da solução, optou-se por colocar nos primeiros capítulos os estudos realizados. Assim, no segundo capítulo é apresentada uma definição de IPTV, os protocolos envolvidos, e as arquitecturas existentes no mercado português enquanto que no terceiro é apresentada a constituição de uma rede de IPTV.

De forma a se ter uma ideia sobre que tecnologias usar num protótipo a desenvolver, no quarto capítulo são apresentados alguns testes preliminares que foram realizados.

No quinto capítulo é apresentada uma proposta de solução a implementar, de forma a se poder contornar as limitações das soluções de IPTV disponíveis no mercado.

Para demonstrar a validade desta proposta desenvolveu-se um protótipo funcional dividido em duas partes; um servidor e um cliente. Assim, no sexto capítulo é demonstrada a forma como se implementou o servidor, enquanto que no sétimo é apresentado o modo como se implementou o cliente.

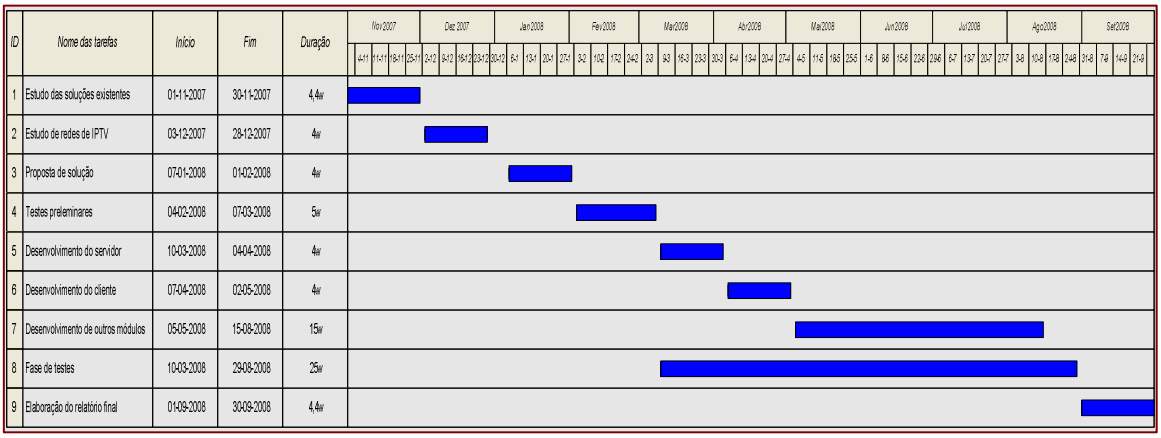
No capítulo oito são usados *screenshots* com o intuito de apresentar o protótipo desenvolvido.

Já no nono capítulo, são apresentados os desenvolvimentos passíveis de serem realizados caso se deseje dar continuidade à Tese aqui apresentada.

Com o intuito de validar a solução proposta, no décimo, e último capítulo, são apresentadas as conclusões.

Para finalizar, são apresentados três anexos, o primeiro sobre o *media player* VLC, o segundo sobre o pacote de bibliotecas XLIB e o terceiro sobre o pacote Libxml.

Tabela 1 Calendarização do projecto



2. *INTERNET PROTOCOL TELEVISON* (IPTV)

Como o próprio nome indica, a IPTV faz uso do *Internet Protocol* (IP) para distribuir televisão digital através de uma rede privada. A IPTV tornou-se numa tecnologia que permite às companhias de telecomunicações a distribuição dos convencionais canais de televisão através de um rede de I*nternet*. Como tal, grandes investimentos têm sido feitos quer na infra-estrutura das redes de *Internet*, quer nas *set-top boxes* necessárias para descodificar o conteúdo recebido.

2.1. DEFINIÇÃO DE IPTV

A *Internet Protocol Television* consiste em distribuir canais de televisão*, vídeo-on-demand* (VOD) ou outros vídeos, através de uma rede privada. Como a IPTV apenas requer o uso do protocolo IP como mecanismo de entrega, os vários tipos de dados podem ser distribuídos pela *Internet* ou por redes IP privadas. Segundo a *Iternational Telecommunication Union*, a IPTV consiste num conjunto de serviços multimédia tal como televisão, música, vídeo, texto e imagens distribuídos por uma rede IP. Como se pode ver com esta definição, esta tecnologia proporciona uma gama elevada de serviços já existentes no mercado e de potenciais serviços a introduzir neste mesmo mercado. **N**um

ponto de vista mais simples, pode-se dizer que a IPTV é usada para oferecer serviços equivalentes, ou melhores, dos que são oferecidos pela tradicional televisão analógica ou pela televisão por cabo, usando para tal uma rede IP. Outra característica importante de referir é que, contrariamente à TV por cabo (por exemplo), na qual todos os canais são distribuídos ao mesmo tempo, na IPTV só são distribuídos os canais que o consumidor selecciona naquele instante.

2.2. DIFERENÇA ENTRE IPTV E *INTERNET* TV

Contrariamente ao que muitas vezes é pensado, IPTV e *Internet* TV não significam o mesmo. Apesar de ambas as tecnologias usarem o protocolo IP para distribuir vídeo, a *Internet* TV usa a *Internet* como meio de entrega do vídeo, enquanto que a IPTV usa redes privadas e dedicadas como meio para distribuir o vídeo. Para além desta principal diferença, estes dois serviços também diferem no que diz respeito ao alcance geográfico, ao *hardware* e *software* necessário e aos custos inerentes. Assim, a IPTV é distribuída localmente (rede privada) e o utilizador necessita normalmente de uma *set-top box* para descodificar o conteúdo dos vídeos distribuídos pela operadora. Já a *Internet* TV, pode ser distribuída para todo o mundo, necessitando apenas de um PC para aceder aos vídeos. Devido ao facto da *Internet* TV apenas necessitar de um *software freeware* para aceder e reproduzir os canais de TV e de a IPTV necessitar de um contrato com um fornecedor do serviço, o custo da *Internet* TV é menor. No entanto, há que ter em conta que para se ter acesso à *Internet* e assim fazer uso dos *softwares* de *Internet* TV, também é necessário ter um contrato com um fornecedor do serviço de *Internet*.

2.3. MERCADO IPTV

Quando uma empresa decide criar um serviço de IPTV, tem que ter em conta os custos necessários para poder criar esse serviço. Conforme foi dito anteriormente, as redes de IPTV distribuem continuamente para os utilizadores múltiplas *streams* através de redes privadas e estes vêem o conteúdo das *streams* numa televisão normal. Apesar deste serviço parecer simples, são necessárias bastantes tecnologias e equipamentos para providenciá-lo. Assim, passam-se a enumerar alguns destes elementos:

- Canais de televisão
- Rede IP e seus componentes

- *Set-top-box*
- *Electronic Programming Guide*
- Servidores de conteúdos

Aliados aos custos de equipamento referidos anteriormente, estão outros não relacionados com a IPTV, nomeadamente custos de marketing, de pessoal, de suporte ao cliente, de manutenção da rede, etc.

Provavelmente é a ausência de conhecimentos precisos, no que diz respeito ao lucro real e ao lucro expectável, que faz com que a IPTV seja um produto que ainda está numa fase de crescimento no mercado.

Normalmente, os fornecedores optam por propôr aos utilizadores a aquisição de pacotes de canais a preços diferentes ou pacotes *Triple/Quadruple Play*. Estes últimos, referem-se a serviços de televisão, telefone, *Internet* e/ou telefone móvel, assim como a subscrição apenas de canais específicos, serviços de *Vídeo on Demand*, gravação programada de programas de televisão específicos e serviços de TV interactiva.

No que diz respeito ao número de subscritores, não existem dados precisos. Segundo a Parks Associates, em 2011, o número de subscritores de serviços IPTV em todo o mundo ultrapassará os sessenta milhões. Segundo esta mesma fonte, entre 2006 e 2007 o número de subscritores aumentou de 4,56 milhões para 10,85 milhões. [6]

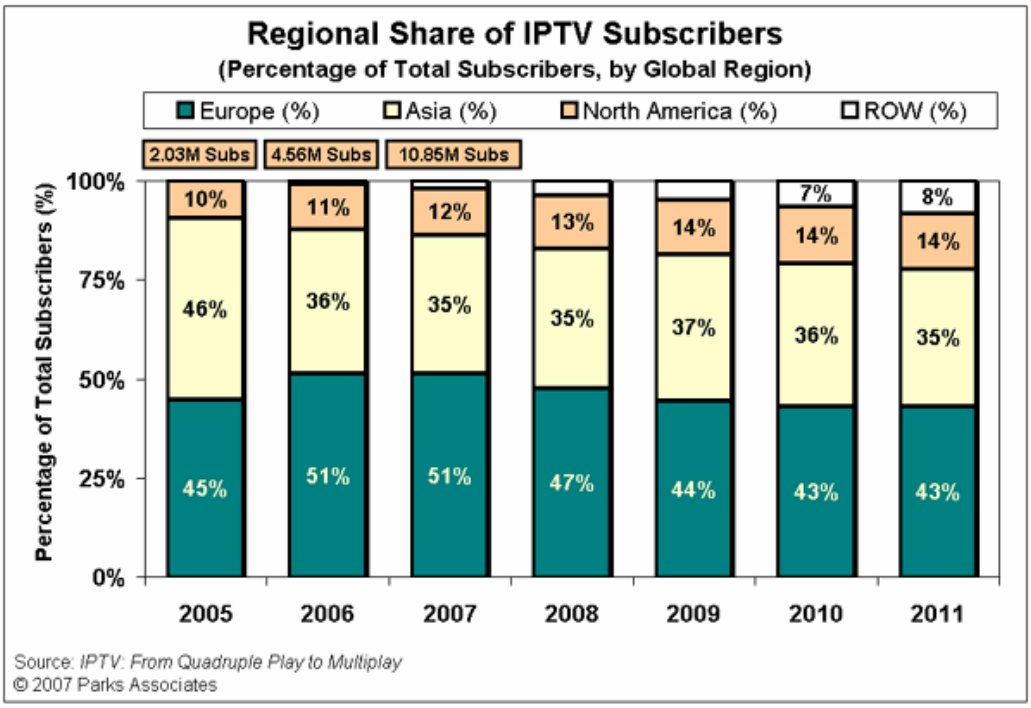


Figura 1 Número de subscritores e sua distribuição por continentes [6]

Num artigo publicado em 11 de Março de 2008[7], a ABIresearch valida a previsão exposta anteriormente e apresenta uma nova previsão até os finais de 2013. Segundo essa previsão, no fim do ano 2013 haverá mais de 90 milhões de subscritores dos serviços de IPTV em todo o mundo.

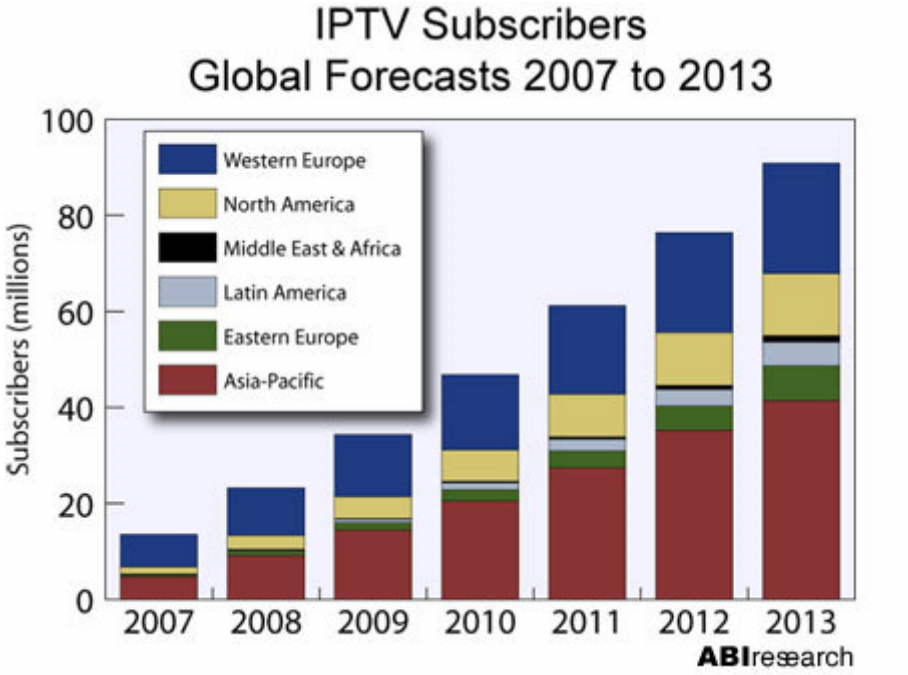


Figura 2 Previsões de número de subscritores [7]

Não existem dados suficientes que permitam contabilizar o impacto da IPTV em Portugal, no entanto, estatísticas mais recentes dão conta de um grande crescimento do número de subscritores do serviço da PT, o MEO. Segundo esses dados, no final do primeiro trimestre de 2008, o MEO contava com 47 mil subscritores, número esse que, graças a um grande investimento na publicidade deste serviço, aumentou para 100 mil no fim de Junho de 2008, ou seja, num trimestre, a PT conseguiu melhor do que duplicar o número de assinantes.[8]

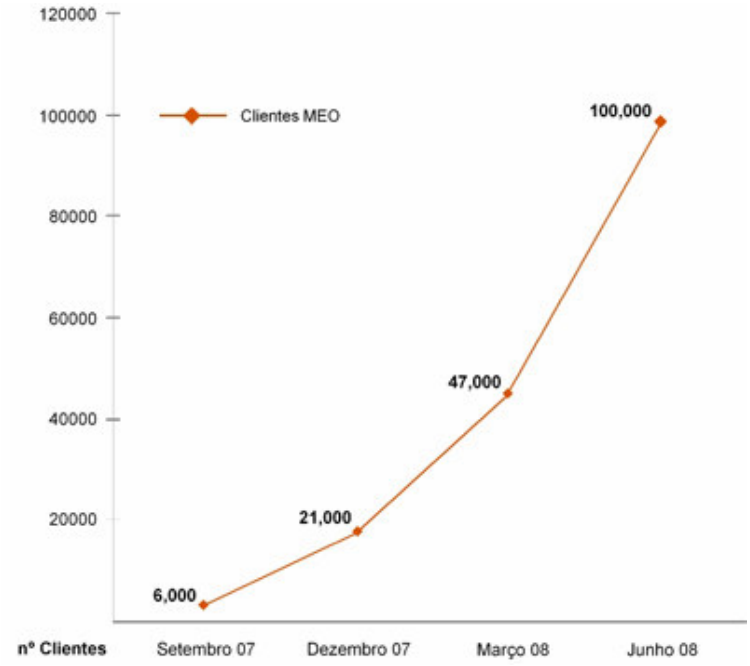


Figura 3 Número de assinantes do serviço MEO da PT [8]

2.4. SOLUÇÕES DE IPTV EM PORTUGAL

Quando esta Tese começou a ser desenvolvida, a IPTV e as suas soluções, em Portugal, estavam ainda pouco desenvolvidas e divulgadas. Neste momento, existem no mercado português duas soluções disponíveis: o MEO da PT e o Smart TV da Clix.

O serviço Smart TV, da Clix, encontra-se pouco documentado, contudo sabe-se que é constituído por um *Modem/Router* que recebe os serviços e distribui o serviço de voz para um telefone, a *Internet* para um computador e o serviço de TV para uma *TVBox*. Por sua vez, a *TVBox* é operada através de um Controlo Remoto que permite ao utilizador visualizar os conteúdos e serviços recebidos pela rede. Segundo a Clix, estas *boxes* vêm já preparadas para a Alta Definição e oferecem som *Dolby Surround*[43]. De salientar ainda,

que a Clix disponibiliza um máximo de três *TvBox*, mas este número pode ser menor, dependendo da largura de banda disponível na zona da intalação. [9]

O MEO rege-se pelos mesmos princípios do Smart TV, ou seja, faz uso de um *Router/Modem* (chamado *HomeGateway*) que recebe os vários serviços e os distribui. Para transformar o sinal recebido em conteúdo que possa ser visualizado, é usada uma MeoBox (semelhante às *TvBox*). Tal como o serviço disponibilizado pela Clix, o MEO usa também um telecomando para aceder e operar as opções oferecidas por este serviço.

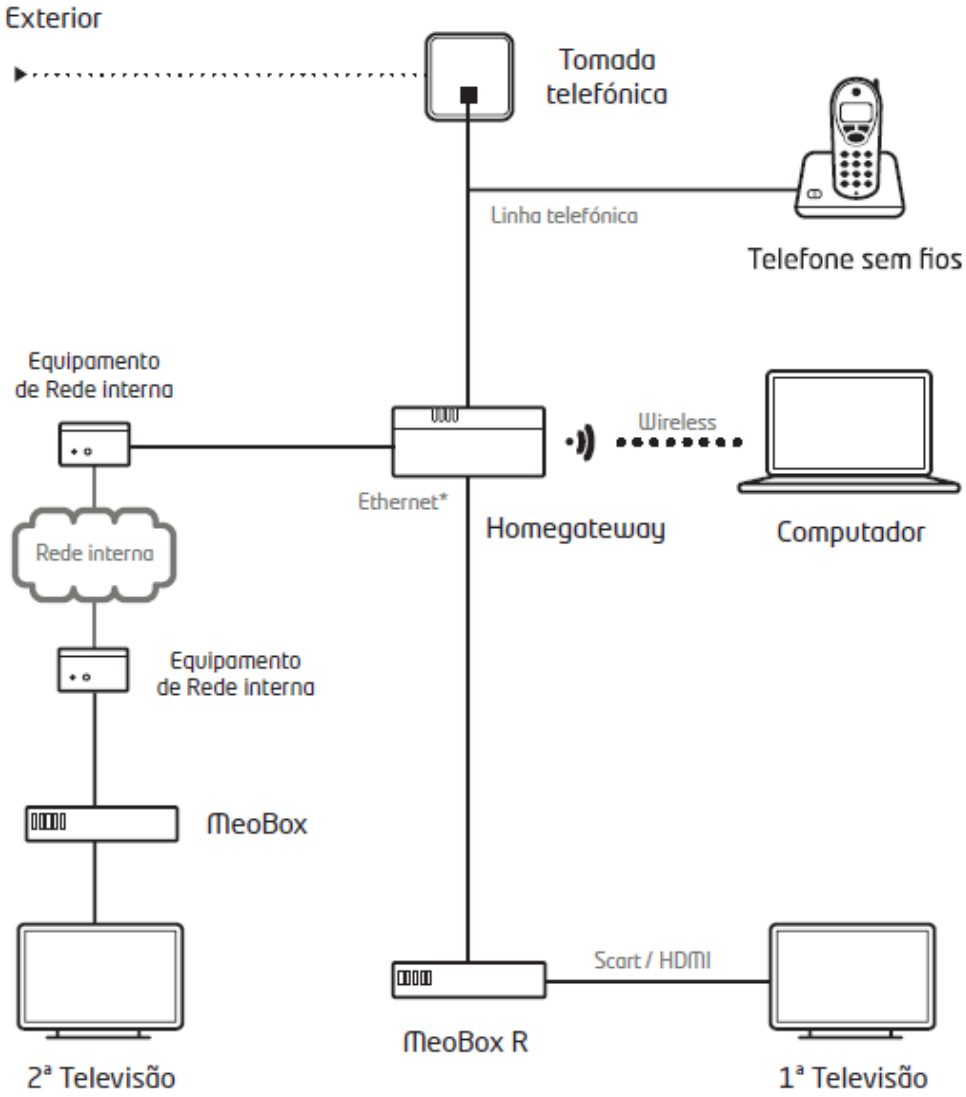


Figura 4 Arquitectura das soluções de IPTV da MEO[11]

Relativamente a figura anterior, salienta-se que o sinal recebido do *Internet Service Provider* (ISP) chega à casa do cliente através de uma ligação *Asymmetric Digital Subscriber Line* (ADSL).

No entanto, a PT disponibiliza outras opções para a arquitectura das soluções de IPTV através do uso do equipamento Coax, do equipamento PLC, ou do equipamento Wireless. Estes equipamentos permitem interligar a MeoBox à HomeGateway. O equipamento Coax é uma solução que permite a transmissão de dados através da rede coaxial, o equipamento PLC permite a transmissão de dados através da rede eléctrica e o equipamento Wireless usa a tecnologia sem fios.

Já no que diz respeito aos serviços, quer a Clix quer a PT, conseguem fornecer um *Triple Play,* sendo que as principais diferenças se situam nas opções/serviços disponíveis. A MEO dispõe de um número de serviços maior que o SmartTv; enumeram-se assim os serviços oferecidos pela MEO:

- Ver um canal
- Barra com título do programa que está a ser visualizado
- Paragem em tempo real
- Guia de TV
- Lista de canais favoritos
- Compra de canais
- Gravação
- Visualização de gravações
- Alugar um vídeo
- Bloquear visualização de canais e conteúdos
- Bloquear compras e alugueres
- Bloquear conteúdos para adultos

Uma das principais limitações destes serviços prende-se com o facto dos fornecedores limitarem o número *set-top-box* disponíveis para cada cliente (a Clix só permite três por exemplo). Esta deve-se ao facto de cada *set-top-box* permitir ao utilizador ver um canal, independentemente do que as outras *set-top-box* da mesma casa estão a ver, e de a largura de banda necessária para a recepção de cada canal de IPTV ser elevada quando comparada com a largura de banda disponível. Ou seja, o que limita o número de *boxes* disponibilizadas a cada utilizador é a largura de banda existente.

Outra das desvantagens que se pode apontar aos serviços de IPTV é o facto de poderem haver pacotes perdidos na rede e assim se peder alguma qualidade.

2.5. PROTOCOLO IP NA IPTV

O protocolo IP é o protocolo mais usado no mundo para ligar computadores em rede. É usado na IPTV, sendo a "base" desta tecnologia.

Devido à sua importância, este protocolo é usado nos modelos de referência de telecomunicações *Open System Interconnection* (OSI) e *Transmission Control Protocol/Internet Protocol* (TCP/IP). Para se conhecer estes modelos, devem ser consultadas as referências [2] e [3].

O protocolo IP é usado na camada *Internet*, no caso do modelo TCP/IP e na camada de Rede (*Network*), no caso do modelo OSI. Requer a atribuição de um endereço único a cada computador da rede, de forma a este poder ser identificado. Os endereços IP são fáceis de reconhecer devido ao seu formato especial, formato esse chamado *dotted decimal* que consiste numa série de quatro números separados por pontos. Um número destes representa um número de 32 *bit* que está subdividido em quatro números de 8 *bit*.

Este protocolo também define as metodologias de roteamento entre múltiplas redes. Como os pacotes podem chegar fora da ordem de envio, as camadas superiores têm a função de os reorganizar. É nesta camada que é inserido um cabeçalho IP (*IP header*) à mensagem, sendo que nesse cabeçalho se destacam os endereços IP do destinatário e do emissor.

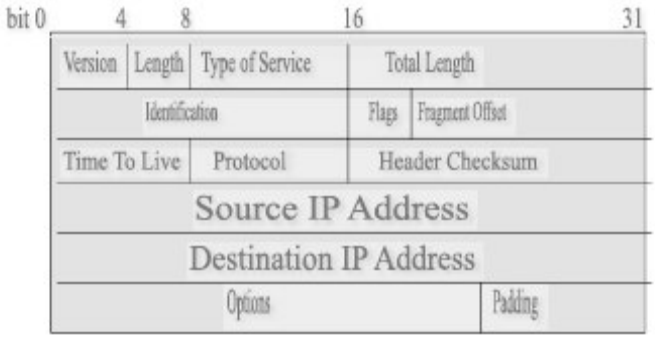


Figura 5 Cabeçalho IPv4 [4]

Uma descrição mais completa do protocolo IP pode ser encontrada nas referências [2] e [3].

2.5.1. MULTICASTING

Um conceito chave das redes IP para aplicações de IPTV é o *multicasting*. Existem dois significados que esta palavra pode ter: por um lado, *multicasting* pode ser usado para definir a distribuição simultânea de vários vídeos através de um único canal digital, por outro, esta expressão também pode significar a distribuição de uma única *stream* para vários clientes simultaneamente.

No primeiro caso, é possível ter num único canal digital vários canais de vídeo, sendo que cada canal de vídeo ocupa uma porção da largura de banda do canal digital.

No segundo caso, uma *stream* vídeo é enviada simultaneamente para múltiplos clientes. Através do uso de protocolos especiais (*Internet Group Membership Protocol* (IGMP), por exemplo) são feitas cópias da *stream* vídeo para todos os clientes de um grupo dentro da própria rede. Assim, todos os clientes que pretendem aceder a um canal *multicast* recebem o mesmo sinal ao mesmo tempo.

Nas soluções IP *unicasting* cada vídeo é enviado para apenas um cliente e, se vários clientes querem ver o mesmo vídeo, a fonte tem de criar diferentes *streams unicast* para cada cliente. Contrariamente, no IP *multicasting,* uma *stream* vídeo pode ser enviada para múltiplos utilizadores. Estas diferenças podem ser observadas na Figura 6. Através do uso de protocolos especiais, a rede tem a possibilidade de fazer cópias das *streams* vídeos e, assim aliviar a fonte de vídeo, uma vez que as cópias passam a ser concebidas na rede. Estas cópias são feitas apenas nos pontos de roteamento necessários, porque os protocolos especializados permitem à rede reconhecer os pacotes *multicast* e encaminhá-los para os múltiplos destinos. Para tal, os pacotes *multicast* têm endereços especiais reservados para *multicasting*.

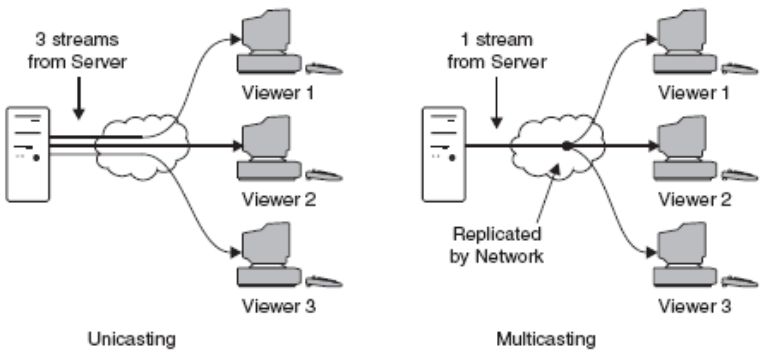


Figura 6 Exemplo de *Unicasting* e *Multicasting*

O uso desta técnica permite uma redução significativa dos custos da construção de uma rede IPTV, pois deixa de ser necessário uma grande largura de banda entre o servidor de aquisição de vídeo e um servidor de alojamento dos vídeos.

Relativamente à distribuição de *streams broadcast* de televisão para os clientes, é maioritáriamente usado *multicasting*. Assim, para mudar de canal, basta o cliente juntar-se a um *multicast*. No entanto, nem todos os equipamentos permitem *multicast.* Em caso negativo, deverá ser enviada para cada cliente uma *stream* única, o que requer uma grande largura de banda. Caso o equipamento permita *multicasting,* apenas será necessário uma cópia de cada canal *broadcast,* sendo reduzida assim a largura de banda necessária.

2.5.2. IGMP

Como já referido, o IGMP é um protocolo que permite aceder a grupos *multicast*. Uma vez que cada canal de televisão é um grupo de *multicast*, sempre que o utilizador muda de canal, é necessário sair do grupo actual e juntar-se ao novo grupo. Existem actualmente três versões deste protocolo, sendo que neste momento apenas é usada a segunda e a terceira versão, pois a primeira não permite comunicar a saída de um grupo. Sendo estes protocolos de enorme importância numa arquitectura IPTV, entendeu-se ser necessária uma breve explicação

O uso deste protocolo envolve o uso de dois dispositivos, um cliente (IGMP *host*) e um *router multicast* (IGMP *router*). Enquanto que o cliente envia mensagens para se juntar e para sair de um grupo *multicast*, o *router* responde aos pedidos de junção e saída de um grupo e determina se deve ser feito *forwarding* dos grupos *multicast*. Para receber grupos *multicast,* o *router* IGMP usa um protocolo de *multicast*, o *Protocol Independent Multicast* (PIM) ou *flooding* estático.

O IGMP providencia quatro funções básicas:

- JOIN, indica que um cliente deseja juntar-se a um grupo *multicast*;
- LEAVE, indica que um cliente já não deseja pertencer a um grupo *multicast;*
- QUERY, permite ao *router* perguntar a determinado cliente a que grupos pertence. Isto torna-se importante para o caso de um cliente se ter juntado a um grupo sem ter saído de outro;
- MEMBERSHIP REPORT, indica a um cliente a que grupos este pertence.

Como já foi referido, actualmente são usadas duas das três versões deste protocolo, sendo que a mais usada nas arquitecturas de IPTV é a segunda versão, que suporta *Any Source Multicast* (ASM). Numa rede destas, os clientes IGMP especificam o grupo a que se querem juntar e recebem todo o conteúdo correspondente a esse endereço *multicast*, independentemente de quem lho está a enviar. A maior diferença entre a segunda e a terceira versão do IGMP está no facto do uso de *Source Specific Multicast* (SSM) ser assegurado pela terceira versão. Como o nome indica, quando o SSM é usado, o cliente IGMP especifica o endereço que quer escutar e apenas recebe dele, prevenindo assim a recepção de tráfico indesejado, que pode vir, por exemplo, de outro cliente IGMP.

Para entender melhor as diferenças do uso de cada uma das versões do protocolo, deve ser consultada a referência [5].

2.5.3. PROTOCOLOS DE TRANSPORTE

Outra camada que merece destaque nos modelos de camadas TCP/IP e OSI é a camada de transporte, que é definida para permitir aos utilizadores manterem uma conversação. Para tal, existem dois protocolos ponta-a-ponta: o *Transmission Control Protocol* (TCP) é orientado às conexões, garante entrega e providencia controlo de erros e de fluxo, enquanto o *User Datagram Protocol* (UDP) não é orientado às conexões, não garante entrega e não providencia controlo de fluxo nem de erro. Nesta camada de rede é inserido um cabeçalho TCP ou UDP (TCP/UDP *header*) que identifica as aplicações que comunicam entre si através do uso de número de porto de destino e número de porto de origem.

Apesar de não ser tão usual, na camada de transporte pode também ser usado o *Real-time Transportation Protocol* (RTP), concebido para aplicações multimédia em tempo real, tal como aplicações voz e vídeo na *Internet*. Assim, este tem o intuito de ser usado em aplicações nas quais o tempo de entrega dos pacotes não pode exceder um certo limite. Para tal, o RTP foi desenvolvido adicionando ao UDP bastantes funções do TCP, mas sem usar algumas funções (do TCP) inúteis para o seu propósito. Uma das principais funcionalidades deste protocolo prende-se com o facto deste permitir *multicasting,* que é um método bastante eficiente para distribuir vídeo e voz através da *Internet*. O RTP é usado em conjunto com o protocolo UDP (UDP/RTP) para permitir que o UDP contenha uma ligação ponta-a-ponta que consiga detectar condições de erro, tais como:

- Determinação de pacotes recebidos fora de ordem

- Detecção de pacotes duplicados
- Determinação de pacotes perdidos
- Determinação de pacotes com um tamanho incorrecto[4]

2.5.4. MPEG VÍDEO *STREAMS*

Para perceber o funcionamento por detrás de uma *set-top-box*, é necessário entender a diferença entre um *codec* e um *container format.*

Um *codec* é um algoritmo de compressão usado para reduzir o tamanho de uma *stream*. Para tal, são usados *codecs* para vídeo e para áudio. Entre os vários tipos de *codecs* destacam-se: MPEG-1, MPEG-2, MPEG-4, Vorbis, DivX.

Já um *container format*, contém uma ou várias *streams* que já foram codificadas pelos *codecs.* Dentro de um *container format*, as *streams* podem ser codificadas usando diferentes *codecs*. No entanto, esta prática pode gerar erros, pois alguns *codecs* são incompatíveis. Entre os vários tipos de *containers*destacam-se: AVI, OGG, MOV, ASF.

Para descodificar uma *stream*, o VLC, por exemplo, desmultiplica (*demuxes*) a *stream*, ou seja, lê o *container format* e separa as componentes de áudio, vídeo e, se existirem, as legendas. Seguidamente, estas componentes são transferidas para os *decoders*, que realizam um processamento matemático para descomprimir as *streams.*

O *codec* usado na solução apresentada nesta Tese é o *Moving Picture Experts Group* (MPEG). Existem várias versões deste *codec*: MPEG-1, MPEG-2, MPEG-4.

Para além de *codec*, o MPEG é também um *container format.* Tal como para o *codec*, existem vários tipos deste *container* MPEG*: Elementary Streams* (ES), *Program Stream* (PS), e *Transport Stream* (TS). Para entender estes formatos, recorre-se ao exemplo da reprodução de um vídeo MPEG num DVD. Neste caso, o *container format* MPEG é composto por diversas *streams* elementares (ES), vídeo, áudio, etc.. Este tipo de *streams* é aglomerado numa única *stream*, a *stream* de programa (PS). No entanto, caso seja pretendido fazer *streaming* de um vídeo numa rede, é necessário formatar estas *streams* para outro formato, o *Transport Stream* (TS).

Analisa-se então o modelo de camadas para uma rede IPTV que faz uso do *codec* MPEG-TS (*transport stream*), comparando-o com o modelo OSI.

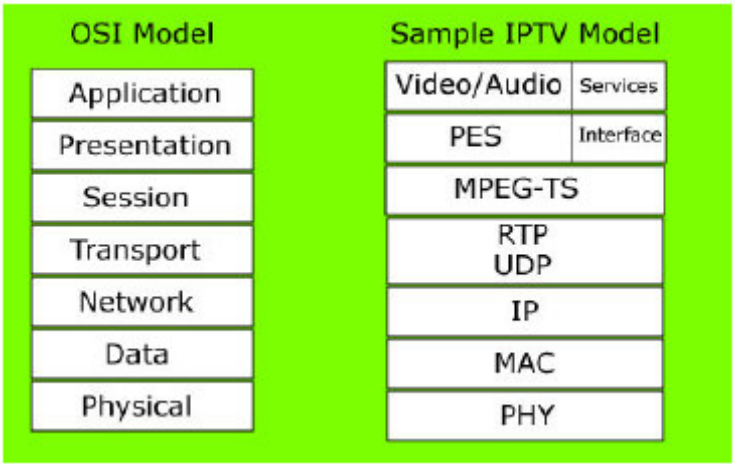


Figura 7 Modelo OSI IPTV usando MPEG-TS

Como é normal na análise de um modelo destes, este pode ser dividido em duas partes: as camadas relativas ao serviço e as camadas relativas ao transporte. Assim, nesta divisão, acima da camada relativa ao *codec* MPEG-TS, inclusive, temos o serviço e abaixo desta temos o transporte.

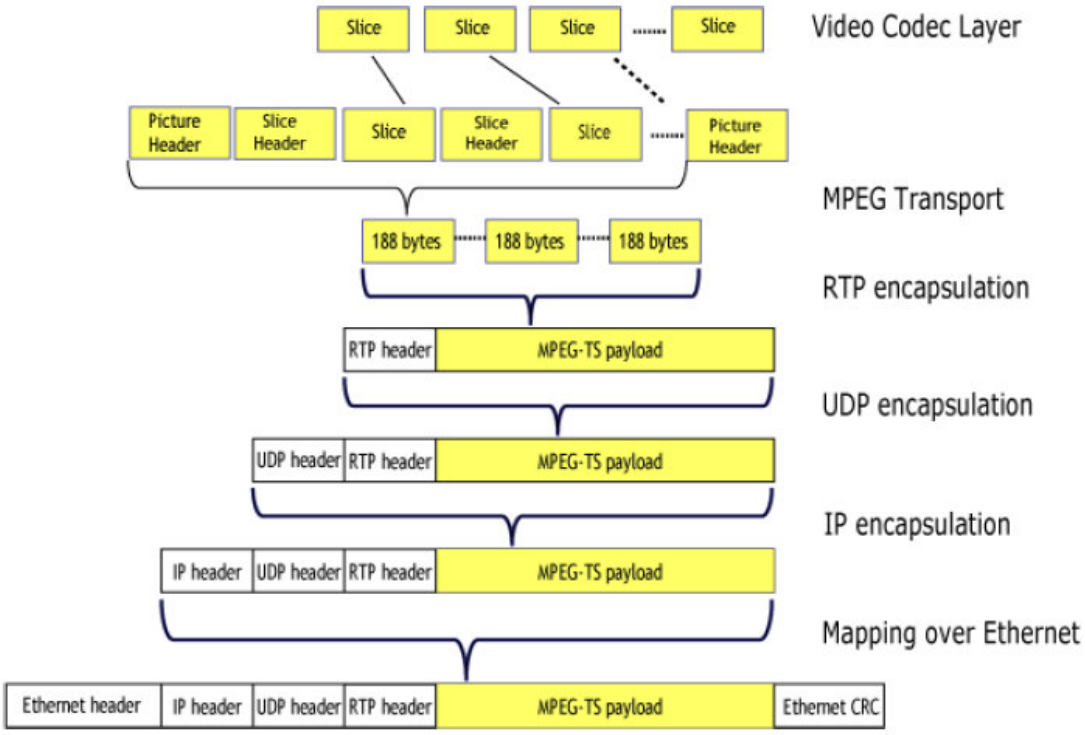


Figura 8 Niveis de inserção de cabeçalhos

A figura anterior permite observar a forma como os vários níveis interagem na construção de uma *stream* MPEG, independentemente de se tratar da parte relativa ao serviço ou ao transporte. Assim, na parte relativa ao serviço, um sinal vídeo de qualquer formato é colocado num *encoder* para o digitalizar e comprimir (neste exemplo, usando MPEG-2 e gerar uma *Elementary Stream* (ES) de vídeo e de áudio). Seguidamente, o *encoder* processa ou faz *slice* (desliza) da ES, de forma a gerar uma *Packetized Elementary Stream* (PES). Cada PES contém uma imagem e um cabeçalho relativo ao *slice*. O passo seguinte é multiplexar individualmente cada PES numa *Transport Stream* (TS). Uma vez que neste exemplo é usada uma *Ethernet,* a *Maximum Transfer Unit* (MTU) é definida como 1560 bytes e o tamanho da PES definido em 188 bytes, o que permite o transporte de até 7*188 bytes como uma MPEG-TS.

Esta parte inicial diz respeito à parte do serviço e, normalmente, inclui a adição de *Service Information* (SI), *Program Specific Information* (PSI), *Entitlement Control* (ECM) e *Entitlement Management Messages* (EMM); usados principalmente para canais encriptados, que normalmente obrigam o utilizador a uma assinatura especial (SportTv por exemplo). Usualmente, esta encriptação é aplicada à MPEG *payload* e não aos cabeçalhos, ou seja, o conteúdo é recebido na mesma mas pode não ver o conteúdo. Finalmente a PES é multiplexada em MPEG-TS.

Os níveis seguintes dizem respeito à parte da transmissão e servem para preparar o envio da MPEG-TS pela rede.

3. CONSTITUIÇÃO DE UMA REDE DE IPTV

Na Figura 9 é possivel ver uma ilustração simples da constituição de uma rede IPTV. Como se pode verificar, a rede está dividida em três partes: *Vídeo Serving Office* (VSO), *Local End Office* (LEO) e a casa do utilizador. O VSO é o responsável por juntar as várias fontes de vídeos e convertê-las em *streams* IP, que são enviadas para o LEO, sendo este responsável por juntar (neste caso) os sinais vídeo, dados (*Internet*) e voz (telefone), de forma a poder transmiti-los na rede IP. Já em casa do consumidor, o sinal recebido é distribuído para os diversos serviços: *Internet*, telefone e televisão. No entanto, antes de ligar a televisão, o sinal de vídeo necessita de passar por uma *set-top box* (STB) para ser descodificado.

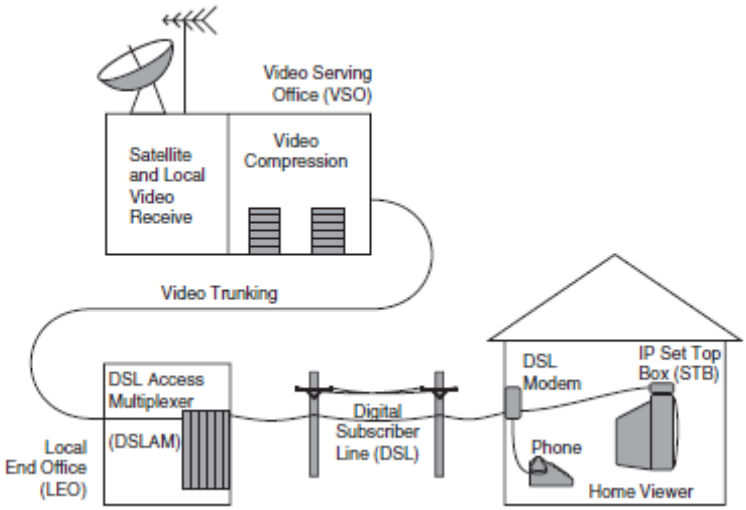


Figura 9 Rede IPTV típica [42]

3.1. ARQUITECTURA TÍPICA DE UM SISTEMA DE IPTV

A arquitectura de uma rede IPTV tem bastantes semelhanças com a de uma rede telefónica, acrescentada de outros equipamentos e funções. Na figura seguinte são apresentados mais pormenorizadamente os sistemas fundamentais da constituição de uma rede IPTV.

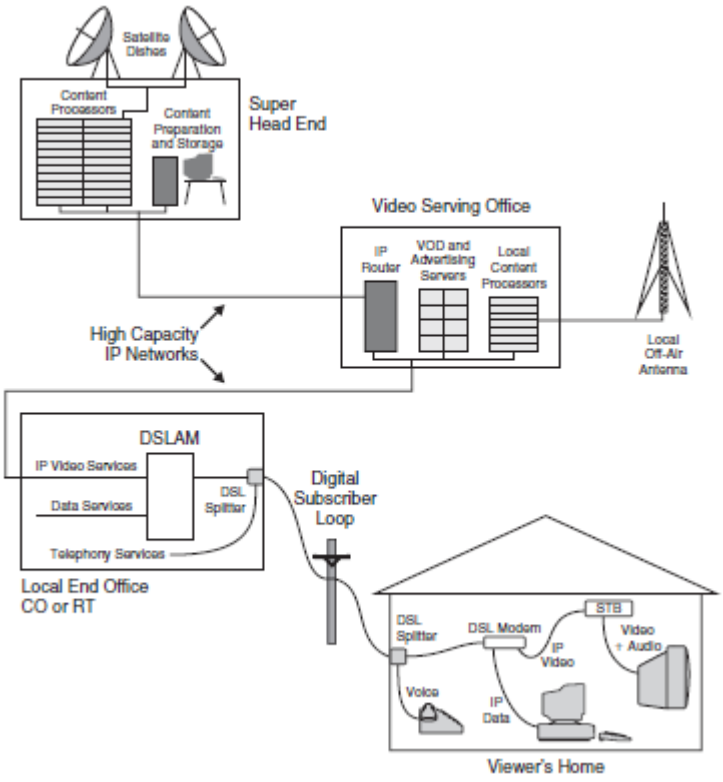


Figura 10 Arquitectura pormenorizada de uma rede IPTV[42]

3.1.1. SUPER HEAD END (SHE)

É responsável por obter os conteúdos dos fornecedores de programas, convertê-los para um formato apropriado de forma a circular na rede IPTV e por enviá-los para o VSO. Para além destas funções, também deve estar preparado para armazenar os vídeos adquiridos, de forma a estes estarem disponíveis no serviço de *Vídeo On Demand*.

O conteúdo obtido pode tratar-se apenas de programas de TV ou de canais de TV de diversos tipos de conteúdos (desporto, política, generalistas, etc.). Estes canais podem vir de diversas fontes, podendo recorrer-se a receptores de satélites ou a uma rede terrestre de transporte de vídeo.

Cada canal adquirido necessita de ser codificado para uma forma padrão (*bit rates standard*, formato do pacote *standard*, tecnologia de compressão *standard*) de maneira a que o equipamento do cliente (a STB) entenda o sinal de vídeo recebido e a forma como o deve tratar. O tipo de vídeo a formatar pode ser um qualquer (MPEG, MPEG-2,MPEG-4), mas, depois de compactado, tem de ser um pacote IP que contenha o conteúdo de vídeo

comprimido para o formato para o qual a rede IPTV foi desenhada e que permita ao cliente receber e entender o que lhe foi enviado.

Seguidamente, os sinais de vídeo que acabaram de ser comprimidos devem ser colocados num rede que os leve até ao VSO. Normalmente estas redes são terrestres porque, devido ao facto de serem necessárias larguras de banda bastante elevadas para suportar centenas de canais, seria caro obter através de serviços de satélite. Para prevenir uma falha que corte o serviço televisivo, por vezes, é construído um segundo SHE a uma distância elevada do primeiro.

No que diz respeito ao conteúdo a ser distribuído por VoD, este pode ter origem em cassetes de vídeo, DVDs ou ficheiros da *Internet.* O conteúdo deve também ser distribuído com ficheiros de dados enviados através de uma rede IP. Outros conteúdos devem ser transmitidos através de *links* de *broadcast* em tempo real, requerendo um mecanismo para receber os sinais de vídeo e guardá-los para outros processamentos. Tal como nos canais de televisão, estes vídeos também necessitam de ser codificados para um formato pré-definido e capaz de ser entendido pelas STB, mas os resultados da codificação em vez de irem para o *Advertising Server* vão para o VoD *server*. No entanto é necessário adicionar informações que descrevam o vídeo: tamanho, nome, descrição, etc. A maioria dos passos são idênticos. O envio dos conteúdos VoD para os servidores VoD, localizados no VSO, é normalmente realizado na forma de ficheiro, através de uma normal rede de dados.

3.1.2. VIDEO SERVING OFFICE (VSO)

O VSO é o responsável por distribuir os serviços para uma determinada área geográfica. Cada VSO recebe conteúdo de vídeo do SHE e é responsável por distribuir todo este conteúdo em tempo real para todos os *Central Office/Remote Terminal* (CO/RT) daquela região. É também aqui que se situam os servidores de VoD e outros servidores que distribuem conteúdos especializados para os subscritores.

Uma das funções que o VSO tem, é o processamento de um conteúdo específico de uma região. Este conteúdo pode ter origem numa *Local Off-Air Antenna* ou em outras fontes, como instituições governamentais, instituições públicas ou outras. Tal como acontecia no SHE, este conteúdo necessita de ser convertido para o formato padrão requerido para circular na rede IPTV.

Neste local, é também possível alterar o formato *standard* para um formato diferente. Por exemplo, um vídeo que havia sido recebido no formato *standard* MPEG-2 pode ser convertido para o formato MPEG-4.

As *streams* de IPTV são criadas no VSO e consistem em pacotes que serão enviados para os COs/RTs. O número de *streams* que são necessárias gerar pelo VSO varia consoante o nível de sofisticação do equipamento remoto. Para um equipamento remoto simples, o VSO necessita de gerar uma *stream* para todos os clientes activos, já para um equipamento remoto sofisticado capaz de duplicar os pacotes de vídeo de saída, apenas é necessário gerar uma *stream* por cada canal *broadcast*. No segundo caso, o equipamento remoto faz tantas cópias quanto as necessárias para satisfazer todos os clientes activos. Como se pode verificar, o segundo caso, trata-se de um exemplo claro de *multicast*, anteriormente explicado.

Tal como já foi referido, os servidores VoD situam-se no VSO e são responsáveis por criar as *streams unicast* e enviá-las para cada subscritor, que vê o conteúdo. Também no que diz respeito ao VoD, o VSO é responsável por responder rapidamente aos comandos *pause, foward, rewind, play, stop*, pressionados pelo cliente aquando da visualização de um filme.

Publicidade, uma fonte de rendimentos para os fornecedores deste serviço, pode ser inserida neste local (VSO) desde que com aprovação legal. Também com o intuito de rentabilizar o produto, é possível tirar partido da interactividade que uma rede IPTV pode proporcionar, ou seja, promover votações, jogos interactivos, compras, etc. Para garantir esta interactividade, os servidores VSO necessitam de *software* para processar os comandos pressionados pelos clientes.

É também aqui necessário arranjar meios de garantir que apenas as STB autorizadas recebem o serviço, ou seja, garantir que apenas os clientes que pagam pelo serviço conseguem ter acesso às funcionalidades disponíveis.

Por fim, deve ser garantido que as ligações aos COs/RTs têm uma capacidade suficiente para fornecer o serviço com qualidade.

3.1.3. CENTRAL OFFICE/REMOTE TERMINAL

Os COs contêm equipamento telefónico enquanto que os RTs, que costumam ser construídos no subsolo, contêm sistemas que conectam as linhas dos subscritores e as ligações digitais ao CO mais próximo. Em ambos, o equipamento pode ser instalado de forma a distribuir IPTV através de circuitos DSL. Dentro de uma CO/RT, estas linhas são ligadas ao DSLAM como se pode ver na figura anterior

Os sinais IPTV são normalmente distribuídos pelo VSO através de uma rede IP com grande largura de banda, geralmente através do uso de fibra-óptica.

Basicamente, a função de um DSLAM é actuar como um *switch Ethernet* e conectar o fluxo de vídeo que chega do VSO às linhas DSL, saindo para cada subscritor. Para tal, o DSLAM examina o endereço IP de todos os pacotes que chegam e envia-os pelo circuito DSL que o conecta ao subscritor que tem esse endereço IP.

Através do uso da tecnologia de *multicasting*, o DSLAM é capaz de tirar uma única *stream* do VSO e copiá-la de forma a satisfazer múltiplos clientes que acedem simultaneamente ao mesmo canal. Como já foi referido, se o DSLAM não for capaz de usar esta tecnologia, têm de ser criadas *streams* individuais para cada cliente, o que ocupa uma maior largura de banda.

O DSLAM deve também estar ligado aos sistemas telefónicos existentes no CO/RT. Um *splitter* DSL ou uma ponte híbrida são usados para permitir, quer ao equipamento DSL quer ao equipamento de telefone, partilhar um único par de fios de cobre que chega à casa de cada subscritor. Dentro do CO/RT, uma perna do *splitter* está ligada ao equipamento de processamento de telefone de voz, enquanto que a outra perna está ligada ao DSLAM para tratar da ligação de dados a alta velocidade e dos sinais de vídeos.

Todos os diferentes serviços podem partilhar a largura de banda da ligação de alta velocidade, oferecida ao serviço do consumidor pela linha DSL. Convém no entanto referir que o serviço de vídeo IPTV é uma componente, enquanto que o serviço de dados a alta velocidade para acesso à *Internet* é outro. Este tráfico pode ser separado pelo DSLAM e conectado a um *router* de dados para ser processado no CO. Esta informação é importante, por exemplo, para se saber os consumos no acesso à *Internet*. Também serviços como

Voice on IP (VoIP) podem ser distribuídos através de saídas separadas do DSLAM, se este for configurado apropriadamente.

3.1.4. CONSUMIDOR

Um dos ambientes mais difíceis para um operador de IPTV é a casa do consumidor, o que se deve ao facto dos dispositivos IPTV necessitarem de uma fonte de energia, uma localização física e uma rede ligada de forma a ligar uma ou mais STB no interior da casa. Algumas tecnologias já foram desenvolvidas para esta rede, como por exemplo, cabo coaxial, par entrelaçado, entre outros.

Em cada casa é instalado um modem DSL de forma a ser possível receber sinais digitais a alta-velocidade do circuito DSL e converter os dados noutras formas que permitam a outros dispositivos a leitura destes dados. Este dispositivo pode encontrar-se integrado numa *home gateway* ou separado desta.

Para separar os sinais requisitados pelos telefones dos sinais de dados a alta velocidade processados pelo modem DSL, é necessário usar um *splitter* DSL. Como referido anteriormente, uma das pernas dos *splitter* é ligada a um equipamento de telefone enquanto a outra perna é ligada ao *modem* DSL. De salientar que muitas vezes o *splitter* está integrado no *modem* DSL.

Alguns operadores instalam um *home gateway* para controlar e comunicar com as múltiplas STB. Este dispositivo pode também servir para gerir a rede caseira, de forma a garantir que a navegação na *Internet* a partir de um PC não compromete a alta prioridade do tráfico de vídeo existente na rede.

As STB são as responsáveis por fornecer muitas das funcionalidades dos sistemas de IPTV. Elas descodificam os sinais de vídeo digital que chegam, produzem o ambiente gráfico no ecrã, permitem mudanças de canal, entre outras funções. De salientar que sem STB adequadas, o sistema de IPTV torna-se ineficaz.[42]

4. TESTES PRELIMINARES

Uma vez conhecida a arquitectura dos sistemas de IPTV, é importante realizar testes por vários motivos:

- Para se poder saber o que é melhor para a solução antes de se começar a desenvolver a solução final
- Caso o que está a ser testado seja o melhor para a solução, tem-se uma base de trabalho
- Para aprender a trabalhar com algo que pode vir a ser necessário no futuro

4.1. COMUNICAÇÃO ENTRE DOIS PCS

O passo inicial nos testes foi a colocação de dois PCs (PC1 e PC2) a comunicarem entre si numa rede privada. Para tal, foi necessário um cabo *ethernet* cruzado e de dois PCs em ambiente Linux, de forma a permitir o envio e recepção de dados. O passo seguinte foi a atribuição de um endereço IP aos dois PCs, de forma a coloca-los na mesma sub-rede. Para tal, na linha de comandos foram usados os seguintes comandos:

```
No PC1: #/sbin/ifconfig eth0 172.16.12.1 netmask 255.255.255.0 // como
root

No PC2: #/sbin/ifconfig eth0 172.16.12.2 netmask 255.255.255.0 // como
root
```

Para garantir que a comunicação pode agora ser realizada, ambos os PCs devem poder "pingar" entre si.

```
No PC1: #ping 172.16.12.2
No PC2: #ping 172.16.12.1
```

4.2. REPRODUÇÃO DE UM VÍDEO

Importante também, é a selecção de um reprodutor de vídeo robusto e capaz de reproduzir *streams* recebidas numa rede. Apesar de, no início, se ter apontado para um programa bastante robusto, o MythTv, acabou por se seleccionar o VideoLan. Tal selecção deve-se ao facto do MythTv ser um programa que já possui características que deveriam ser desenvolvidas ao longo desta Tese e do VideoLan ser uma excelente escolha para a reprodução de *streams* de vários tipos, além de poder ser operado na linha de comandos. O estudo das características do VideoLan que levaram à sua escolha são mostradas no Anexo A. Assim, reproduzir um vídeo usando o VideoLan é possível usando quer o seu interface gráfico, quer a linha de comandos. Para reproduzir um simples vídeo usando a linha de comandos:

```
%vlc /home/utilizador/vídeo.mpg
```

4.3. VER UM VÍDEO USANDO *SECURE SHELL* (SSH)

Esta etapa serve para permitir ao PC2 ver um vídeo alojado no PC1. Para tal, usou-se o protocolo X através do SSH. No entanto, para ser possível ver um vídeo presente noutro PC, foi necessário editar o ficheiro de configuração do SSH (normalmente presente em `/etc/ssh/sshd_config`). Nesse ficheiro foi importante activar as propriedades: `X11 Forwarding`, e `Permit User Environment.`

Seguidamente foi necessário reiniciar o SSH e, posteriormente, no PC2 usou-se o seguinte comando para ter acesso a linha de comandos do PC1:

```
%ssh –X user@172.16.12.1
```

Caso não se obtenha sucesso, pode ser necessário exportar a variável de ambiente DISPLAY do PC1 para o PC2.

Quando já se estava na linha de comandos do PC1, bastou correr o comando:

```
%vlc /home/utilizador/vídeo.mpg
```

4.4. VER O QUE ESTÁ A SER REPRODUZIDO NUMA *WEBCAM*

Para simular o que está a ser reproduzido por um canal de televisão em tempo real, pode ser usada uma *webcam*, uma vez que esta permite reproduzir o que esta está a gravar em tempo real.

Usando uma simples *webcam* de ligação USB, foi necessário configurá-la, ou seja, criar um *device driver*. Para a criação deste dispositivo, foi usado um módulo existente para este propósito, o gspca.ko. Depois deste módulo ser inserido, é criado um *device driver*, normalmente em `/dev/video0`.

Apesar de existirem programas específicos para reproduzir o que está a ser gravado por uma *webcam*, o VideoLan também tem essa capacidade. Assim, usou-se o seguinte comando para reproduzir o conteúdo de uma *webcam*:

```
%vlc v4l://v4l-vdev=”/dev/video0” :v4l-adev=”/dev/dsp” :v4l-norm=3
:v4l-frequency=-1
```

Para reproduzir no PC2 a *webcam* ligada ao PC1 usando o protocolo X, foi usado o comando SSH para, como já se viu, no PC2 aceder à linha de comandos do PC1. Uma vez na linha de comandos do PC1, bastou utilizar o comando anterior.

O uso do VLC permite transmitir o que está a ser gravado pela *webcam* do PC1, através da utilização de diversos protocolos, entre eles estão o UDP, o RTP ou o *Hypertext Transfer Protocol* (HTTP). Estas possibilidades podem ser consultadas no Anexo A.

Por UDP por exemplo:

```
No PC2: % vlc –vvv udp:

No PC1: % vlc  –vvv
v4l:/dev/video0:norm=secam:frequency=543250:size=640x480:channel=0:adev
=/dev/dsp:audio=0–
sout’#transcode{vcodec=mp4vacodec=mpga,vb=3000,ab=256,vt800000,keyint=8
0,deinterlance}:std{Access=udp,mux=ts,url=172.16.12.2}’ ttl 12
```

Para gravar no disco o que está a ser gravado pela câmara:

```
vlc v4l:// :v4l-vdev="/dev/video0" :v4l-adev="/dev/dsp" :v4l-norm=3
:v4l-frequency=-1 --sout
'#transcode{vcodec=mp1v,vb=1024,scale=1,acodec=mp2a,ab=192,channels=2}:
duplicate{dst=std{access=file,mux=mpeg1,dst="/local/a/gravar/nome"}}'
```

4.5. SERVIDOR

A utilização de um servidor é importante, quer se use o SSH para ver um vídeo alojado no servidor, quer se use uma transmissão HTTP, RTP ou UDP para reproduzir um vídeo alojado no servidor.

4.5.1. APACHE

No caso de se usar a comunicação com o protocolo X, é vantajoso possuir um servidor Apache. Com este servidor pode-se armazenar, com alguma segurança, os conteúdos a reproduzir: vídeos, filmes, som. Para utilizar o servidor Apache, que é um servidor HTTP, bastou instalá-lo e aceder a este. No entanto, para uma correcta utilização, convém editar o seu ficheiro de configuração, o `httpd.conf`, cujas propriedade "endereço de escuta", "porto de escuta", "directório raiz" e "cliente" foi necessário alterar.[1]

Para aceder ao servidor, bastou abrir um *browser* e digitar o endereço e o porto de escuta. Exemplo:

```
http://172.16.12.1:81
```

4.5.2. SERVIDOR TCP/IP EM C

No caso de se usar o VLC, quer para enviar quer para receber as *streams* vídeo, é necessário ter um servidor capaz de transmitir/receber os pedidos. Tal foi possível com a criação de um servidor TCP/IP tradicional, no qual foram usados *sockets*. Passa-se assim a enumerar os passos exigidos na criação de um servidor.

Criar um *socket*:

```
sock=socket(AF_INET,SOCK_STREAM,0);
```

Editar estrutura do *socket*:

```
struct sockaddr_in me; //criar estrutura
int len=sizeof(me); //tamanho da estrutura
bzero((char*)&me,len);//zerar estrutura
me.sin_family=AF_INET;//Família
me.sin_addr.s_addr=htonl(INADDR_ANY);//Endereço
me.sin_port=htons(8450);//Porto
```

Ligar o *socket*:

```
bind(sock,(struct sockaddr *)&me,len);
```

Escutar pedidos de ligação:

```
listen (sock,5);
```

Aceitar a ligação:

```
int novo;

novo=accept(sock,(struct sockaddr *)&me,&len);
```

Ler o que é enviado:

```
Char datasize;

Char linha [81];

Read(novo,&datasize,1);//determinar o tamanho do que falta receber

Read (novo,linha,datasize)//Ler os dados
```

4.5.3. CLIENTE TCP/IP EM C

Para poder comunicar com o servidor, foi necessário criar um cliente capaz de se ligar ao servidor e de enviar dados, cujos passos imprescindíveis na sua criação se encontram de seguida.

Criar *socket*:

```
Strcuct sockaddr_in target;

Int sock, len=sizeof(target);

Struct hostent *server;

Sock=socket (AF_INET,SOCK_STREAM,0);
```

Editar estrutura do *socket*:

```
server=gethostbyname(argv[1]); //nome do servidor na linha de comandos

bzero((char*)&target,len);

target.sin_family=AF_INET;

target.sin_addr=*(struct in_addr*) *server->h_addr_list;//nome escrito
na linha de comandos

target.sin_port=htons(8450);
```

Ligar o *socket*:

```
connect(sock,(struct sockaddr *)&target,len);
```

Escrever:

```
char datasize,linha[81];
datasize=strlen(linha)+1;//tamanho a enviar
write(sock,&datasize,1);//envia tamanho a enviar
write(sock,linha,datasize);//envia
```

4.6. INTERFACE GRÁFICO

De forma a integrar as opções desejadas e permitir ao utilizador ter acesso às funções disponíveis para esta solução, torna-se imprescindível a criação de um interface gráfico. Este interface gráfico tem de conseguir embeber uma janela que demonstre o que está a ser reproduzido pelo VLC.

4.6.1. GAMBAS

O Gambas é um programa *freeware,* semelhante ao *Visual Basic*. Ou seja, é um programa que permite uma programação de alto nível orientada a objectos mas fazendo uso de outras bibliotecas.

Uma característica interessante deste programa, é que tem uma função que permite embeber janelas existentes no ambiente gráfico, `embEmbedder_Embed`. No entanto, para as embeber, tornou-se necessário conhecer o seu ID, para o qual se recorreu à linha de comandos e procurou o ID correspondente à janela com determinado nome (já conhecido previamente). Assim, na linha de comandos, correu-se o seguinte comando:

```
%xwininfo –name “nome_da_janela” |grep “Window id”
```

Usando esta chamada de sistema dentro de um programa Gambas, gravando o valor obtido para uma variável e chamando a função `embEmbedder_Embed` com esta variável, é possível embeber a janela VLC.

4.6.2. BROWSER

Outra das opções proporcionadas pelo VLC é o facto deste poder ser embebido num *web browser*. Os *browser*s suportados são: Mozilla, Firefox e Safari. Para ter acesso a esta funcionalidade foi instalado o *plugin* mozilla-plugin-vlc e um dos *browsers* suportados. O uso deste *plugin* baseia-se na criação de páginas HTML com as funções essenciais para embeber o VLC. Para testar este *plugin*, programou-se em HTML usando um exemplo presente na referência [33].

4.6.3. LIBVLC

Esta opção usa uma biblioteca, a Libvlc.h, para operar e disponibilizar as várias opções do VLC. Esta biblioteca é escrita em C, o que permite uma programação a um nível mais baixo que os dois exemplos anteriormente ilustrados. Para testar esta API na reprodução de um vídeo alojado em `/home/vídeos` desenvolveu-se o seguinte código:

```
#include <stdio.h>

#include <stdlib.h>

#include <vlc/libvlc.h>

//Trata uma excepção

static void quit_on_exception (libvlc_exception_t *excp) {

   if (libvlc_exception_raised (excp)) {

      fprintf(stderr, "error: %s\n",
libvlc_exception_get_message(excp));

      exit(-1);

   }

}

int main(int argc, char **argv) {

   libvlc_exception_t excp; // excepção

   libvlc_instance_t *inst; //Instancia VLC

   int item;

   char *filename = "/home/videos/2004-10-30_pl_arsenal-southampton_2-
2_van_persie.wmv"; //Nome do ficheiro

   libvlc_exception_init (&excp);//Inicializar uma excepção

   inst = libvlc_new (argc, argv, &excp);//Criar nova instancia

   quit_on_exception (&excp);//Em caso de excepção sai
```

```
    item = libvlc_playlist_add (inst, filename, NULL, &excp);
//adicionar item à playlist, retornando o seu ID

    quit_on_exception (&excp);

    libvlc_playlist_play (inst, item, 0, NULL, &excp); //Reproduzir
playlist, com ID=item

    quit_on_exception (&excp);

    sleep (10000000);

    libvlc_destroy (inst);//Destruir a instancia criada

    return 0;


}
```

O estudo ilustrado no Anexo A demonstra, entre outros, o modo de empregar esta biblioteca, assim como as suas principais funções.

4.7. XMLTV

A XMLTV representa um conjunto de utilitários que permitem trabalhar com listas de programas. Assim, trabalham com listas guardadas num formato baseado no XML, o XMLTV. A ideia deste projecto é separar a parte de fundo da parte de topo, ou seja, separar a parte de obtenção das listas (fundo) da forma como estas são mostradas ao utilizador (topo).[18]

De forma a ser possível extrair a informação presente no ficheiro XMLTV, foi necessário perceber a estrutura deste tipo de ficheiros.

Assim, o ficheiro XMLTV tem o nó `<tv>` como nó raiz, que dentro de si tem os sub-nós `<channel>` e `<programme>`.

No primeiro dos sub-nós está presente a propriedade `id` e o sub-nó `<display-name>`. Na propriedade `id` são definidos os identificadores de cada canal, que são usados no processo de obtenção da programação. Já no sub-nó `display-name`, está presente o nome do canal que tem a propriedade `id` igual à presente no nó superior.

O segundo sub-nó, `<programme>`, é o responsável pela programação. Neste sub-nó estão presentes as propriedades `start`, `stop` e `channel`. A propriedade `channel` identifica o canal a qual esta programação se refere (usa o `id` do sub-nó `<channel>`); a

propriedade `start` identifica a hora a que o programa começa e a propriedade `stop`, a hora a que o programa termina. Também no sub-nó `<programme>` está presente um sub-nó, o `<title>`, que tem uma propriedade que identifica a língua da programação (lang) e o nome do programa de tv referente às propriedades anteriormente definidas.

Exemplo:

```
<tv source-info-url="http://tvcabo.pt" source-data-
url="http://www.tvcabo.pt/Televisao/Programacao.aspx" generator-info-
name="XMLTV" generator-info-url="http://membled.com/work/apps/xmltv/">

...

<channel id="AXN.tvcabo.pt">

    <display-name>AXN</display-name>

  </channel>

..

<programme start="20080825000400 +0100" stop="20080825015200 +0100"
channel="AXN.tvcabo.pt">

    <title lang="pt">SÃ³ Tu</title>

    <category lang="pt">Film</category>

  </programme>

</tv>
```

A figura seguinte demonstra a organização do ficheiro XMLTV.

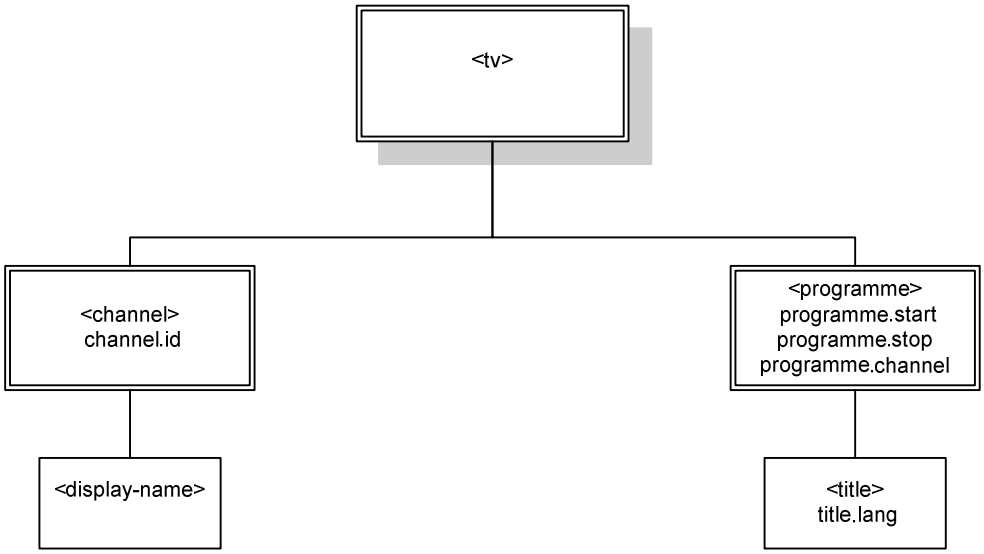


Figura 11 Organização do ficheiro XMLTV

4.7.1. TVXB

Trata-se de uma aplicação para usar em ambiente Windows, que "rouba" a programação demonstrada num *site* e, posteriormente, cria um ficheiro XMLTV com a programação obtida. Para correr esta aplicação em ambiente Linux, teve que se instalar e usar o emulador de Windows, o Wine. Contudo, anteriormente, foi necessário editar um ficheiro de configuração, no qual foi importante referir a página *web* à qual este aplicativo vai buscar a informação e o número de dias para os quais se deseja a programação.

4.7.2. XMLTV-DRUID/GSHOWTV

Tal como o Tvxb, este programa "rouba" a programação demonstrada numa página *web*. No entanto, este programa é usado em ambiente Linux. Depois de instalado, este programa disponibiliza um interface gráfico que permite a sua configuração. Nessa configuração foi escolhido o país para o qual se deseja a programação, a hora a que, diariamente, o programa deve obter de forma automática a programação e, para todos os canais disponíveis, escolher se se deseja a sua programação.

Já o Gshowtv é um interface gráfico para o xmltv-druid. Neste interface a programação é organizada numa grelha hora/canal.

4.7.3. XSLTV

Este programa consiste num ficheiro *Hypertext Markup Language* (HTML), que lê um ficheiro XMLTV e apresenta a programação numa grelha. Para permitir a leitura desta página HTML, foi necessário o uso de um *browser*.

4.8. LIRC

O Lirc é um pacote que permite descodificar e enviar sinais infra-vermelhos a partir de muitos controladores remotos.

A parte mais importante do Lirc é o seu demónio (aplicação que corre em segundo plano) que descodifica os sinais IR recebidos pelo *device driver* e providencia a informação num *socket*. O sistema de janelas X deve ser configurado de modo a usar o controlo remoto como um dispositivo de entrada; para tal os eventos X devem ser enviados para as aplicações desejadas.

Com vista à utilização do comando Toshiba MCE Remote Control-PX1246R-1ETC, tornou-se necessária a instalação deste pacote, assim como a sua correcta configuração. Deste modo, uma vez que o controlo remoto a ser usado não consta na lista de comandos auto configuráveis, durante a instalação deste pacote optou-se por uma instalação padrão. No entanto, com esta instalação, o comando não foi configurado, mas foi criado um *device driver* em `/dev/lirc0`. Foi portanto possível configurar o controlo remoto editando o ficheiro `lircd.conf`, que se encontra em `/etc/lirc`. Esta edição poderia ter sido realizada usando as indicações que constam na página oficial deste pacote. Contudo, usou-se um ficheiro de configuração deste comando, que se encontra disponível na *Internet* [16]. Para se obter uma completa configuração deste controlo remoto, foi necessário seguir as indicações presentes no ficheiro, que indicam que este controlo remoto tem dois códigos por botão [17].

```
# this config file was automatically generated
# using lirc-0.8.2(default) on Sun Jan 27 13:32:13 2008
#
# contributed by Frank Reimann, Berlin (capital city of Germany)
#
# brand:                       Toshiba
#
# model no. of remote control:  Toshiba MCE Remote Control (X10),
#                                 model no. PX1246E-1ETC, model type OR23E,
#                                 Universal Remote Control
#
```

```
# devices being controlled by this remote:
#                       RF USB receiver: X10 Wireless Technology, Inc.
#                       Manufacturer: X10 WTI (Made in China by X10)
#                       USB Vendor Id: 0bc7,
#                       USB Product Id: 0006,
#                       Revision Number:  1.00
#                       Toshiba MCE Remote Control Receiver,
#                       model no. PX1246E-1ETC, model type: CM23E
#
# Hint:
# This remote control has two codes per button (see toggle_bit_mask below; XOR).
# A button toggles between two states like a toggle switch every time it is
# pressed.
# The state of a key changes everytime it is pressed and the appropriate code
# for the new state is send to the receiver.
# If both codes are assigned to the same button name then this button behaves
# like a push-button. Both button codes are accepted for the same button name
# when option toggle_bit_mask is enabled.
#
# One more tip:
# You should add a line with "options lirc_atiusb repeat=15" to the distribution
# specific module parameters file (like /etc/modules.conf
# or /etc/modprobe.conf) for an accurate key recognition. Consult the
# documentation of your distribution for further information concerning module
# parameters.
# Two or more button codes are sent in case default value "repeat=10" is used
# and when a button is pressed once. You can try greater values for slower
# button code repetition.
#
# More driver options (module parameters) and additional information are shown
# with command "modinfo lirc_atiusb".
begin remote

  name  Toshiba_PX1246E-1ETC
  bits           16
  eps            30
  aeps           100

  one             0     0
  zero            0     0
  pre_data_bits          8
  pre_data            0x14
  post_data_bits         16
  post_data           0x0
  gap              235969
  toggle_bit_mask    0x80800000


      begin codes

          STANDBY               0x5782
          RED                   0x0732
          YELLOW                0x0934
          GREEN                 0x0833
          TELETEXT              0x6B96
          BLUE                  0x0A35
          BACKSPACE             0x75A0
          INFO                  0x042F
          UP                    0x6F9A
          LEFT                  0x729D
```

```
        OK                  0x739E
        RIGHT               0x749F
        DOWN                0x77A2
        TIMESHIFT           0x6D98
        GUIDE               0x0631
        LIVE                0x719C
        DVD                 0x5984
        VOLUME+             0x5E89
        VOLUME-             0x5D88
        SYSTEM-LOGO         0x709B
        CHANNEL+            0x608B
        CHANNEL-            0x618C
        MUTE                0x5580
        PREVIOUS            0x76A1
        NEXT                0x78A3
        REWIND              0x79A4
        PLAY                0x7AA5
        FAST-FORWARD        0x7BA6
        RECORD              0x7CA7
        STOP                0x7DA8
        PAUSE               0x7EA9
        1                   0x628D
        2                   0x638E
        3                   0x648F
        4                   0x6590
        5                   0x6691
        6                   0x6792
        7                   0x6893
        8                   0x6994
        9                   0x6A95
        ASTERISK            0x0C37
        0                   0x6C97
        HASHMARK            0x0D38
        CLEAR               0x0530
        ENTER               0x0B36

    end codes
end remote
```

4.9. CRIAÇÃO DE JANELA COM TEXTO DESEJADO USANDO XLIB

Para elaborar esta janela foram usados os conceitos estudados no Anexo B. Nesta aplicação são usadas as bibliotecas `X11/Xlib.h` e `X11/keysym.h`. Para começar, estabeleceu-se uma ligação ao servidor X usando a função `XOpenDisplay`. O passo seguinte foi a definição do tamanho da janela. Para tal, inicialmente, obteve-se o identificador do ecrã e as dimensões deste ecrã usando o seguinte código:

```
screen_num = DefaultScreen(display);//identificador do ecrã

display_width = DisplayWidth(display, screen_num);//largura do ecrã

display_height = DisplayHeight(display, screen_num);//altura do ecrã
```

Conhecendo estes valores, procedeu-se à definição do tamanho da janela, sendo a largura da janela igual à largura do ecrã e a altura igual a um quinto (1/5) da altura do ecrã.

Para criar uma janela com estas características usou-se a função `XCreateSimpleWindow()`. Esta função recebe os seguintes parâmetros:

- O *display*
- A janela raiz, obtida usando `RootWindow (display,screen_num)`
- A posição da janela, coordenadas x e y. Neste caso ambas valem zero (0) para posicionar a origem da janela no canto superior esquerdo do ecrã.
- A largura e a altura definidas anteriormente
- A largura do bordo da janela. Neste caso vale dois (2).
- As cores de *foreground* e *background*. Neste caso é usada a cor branca para *background* e preta para *foreground*. Para tal, são usadas as funções `BlackPixel(display, screen_num)` e `WhitePixel(display, screen_num)`.

Passou a ser necessário mapear a janela para que esta seja lançada no ecrã, ou seja, é necessário pedir ao servidor para mapeá-la (usando `XMapWindow()`) e de seguida obrigar este a ler os pedidos pendentes com a função `XFlush()`.

Com o intuito de editar o visual da janela, criou-se uma estrutura GC (`XGCValues`) e um Gc (*graphics context*) a zero, usando `gc=XCreateGC(display, janeladesejada, 0, &values)`. Os valores da estrutura GC foram editados de forma a se obter o estilo de desenho desejado. Inicialmente definiu-se para o GC a cor do fundo e da frente, usando as funções `XSetForeground(display,gc,BlackPixel (display,screen_num))` e `XSetBackground (display,gc,WhitePixel (display,screen_num))`. Para definir o estilo das linhas a desenhar usaram-se as seguintes características:

- `int line_style = LineSolid;` para o estilo da linha a desenhar
- `int cap_style = CapButt;` para o estilo dos limites das linhas
- `int join_style = JoinBevel;` para as junções
- `unsigned int line_width = 2;` para a largura das linhas

Para aplicar este estilo de linha ao GC, foram necessárias as funções:

```
XSetLineAttributes(display, gc,line_width, line_style, cap_style,
join_style);
```

No que diz respeito ao GC, por fim, definiu-se como estilo de preenchimento, o preenchimento sólido (`XSetFillStyle(display,gc,FillSolid)`)

De forma a desenhar o texto desejado na janela, é necessário definir o nome da fonte (da letra), o local onde o desenhar, o que desenhar e por fim desenhar. Assim, definiu-se o nome da fonte e o tamanho dos seus caracteres. A fonte envolvida é a helvética, de tamanho 24 (`char* font_name = "*-helvetica-*-24-*"`) . Para determinar o tamanho dos caracteres desenhados com esta fonte, é necessário somar a parte ascendente à descendente (`font_height = font_info->ascent + font_info->descent;`). O primeiro parâmetro recebido por esta aplicação é o texto que se quer escrever. Para desenhar o texto desejado foi usado:

```
text_string = argv[1];//recebe texto
string_width = XTextWidth(font_info, text_string, strlen(text_string));
x = 1;//para alinhar à esquerda
y = (win_height + font_height) / 2;
XDrawString(display,win,gc,x,y,text_string,strlen(text_string));
```

4.10. *HIGH DEFINITION* (HD)

Uma tecnologia bastante em voga e que merece bastante atenção por parte dos fornecedores de serviços de televisivos é a alta definição. Exemplo disso é o aparecimento de canais nacionais em HD, a SportTv HD por exemplo. A alta definição consiste na reprodução de vídeos a uma resolução superior à usada pelos sistemas tradicionais (PAL, NTSC). Para ser possível fazer uma comparação, um sistema NTSC tem uma resolução de 720x480p, enquanto que uma transmissão em HD pode atingir as 1920x1080p. [21][22]

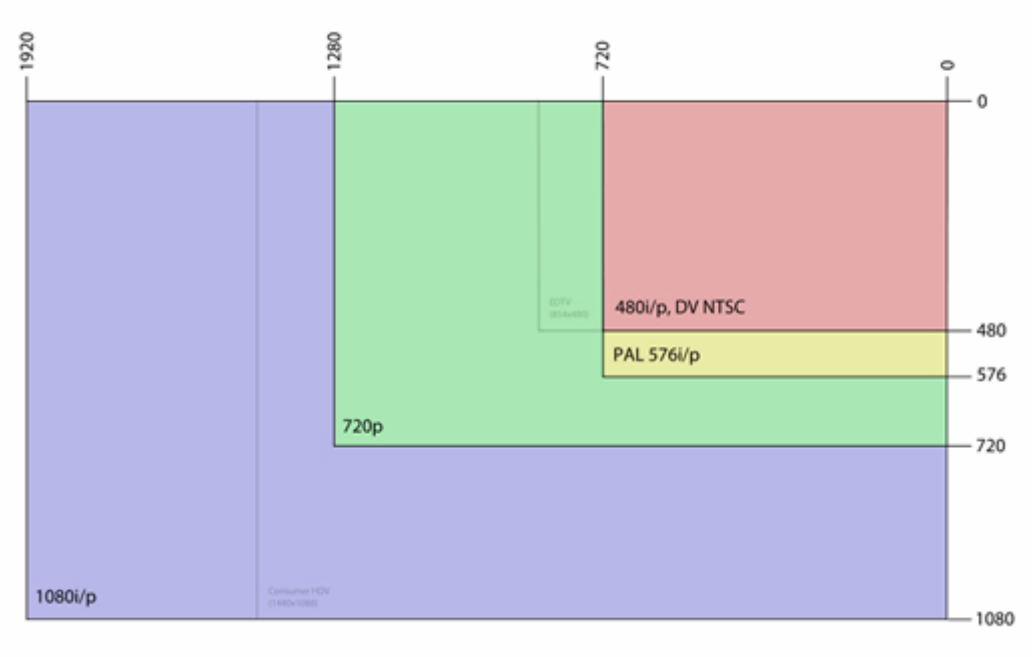


Figura 12 Resoluções dos vários formatos de TV [19]

Para satisfazer as exigências necessárias na reprodução de conteúdos em HD, algumas operadoras necessitaram de mudar o seu equipamento de recepção de conteúdo televisivo (a Zon TvCabo, por exemplo, lançou a ZonBox)[20].

Pelos motivos acima apresentados, tentou-se primeiramente averiguar a capacidade do VLC para reproduzir vídeos no formato HD, para seguidamente averiguar a sua capacidade de transmissão deste tipo de vídeos.

Para iniciar esses testes necessitou-se de vídeos no formato HD. Esses vídeos foram descarregados da *Internet* da referência [23]. Usaram-se dois vídeos HD com diferentes resoluções:

- 1280x720p, 29.97fps - Space Shuttle Backflip
- 1920x1080i, 29.97fps - Space Shuttle STS117 Launch

Usando um PC Intel Centrino Duo 1.5GHz e 2 Gb, conseguiu-se concluir que o VLC consegue reproduzir perfeitamente ambos os vídeos HD.

Para testar as capacidades do VLC em fazer *streaming* de vídeos HD, usou-se mais uma vez a mesma máquina a enviar em *loopback* (para ela mesma). Usou-se a seguinte configuração:

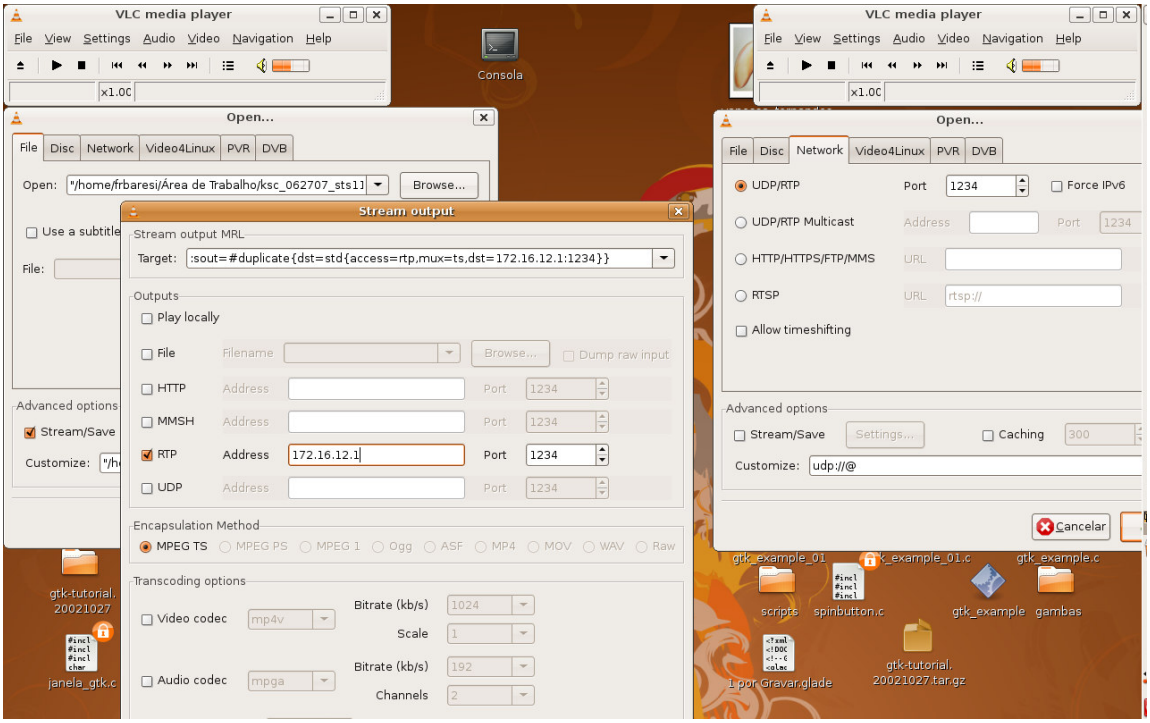


Figura 13 Definições para fazer *streaming* de video HD

Usando esta configuração, conseguiu-se uma visualização perfeita de ambos os vídeos, como pode ser comprovado pela imagem abaixo, que mostra o vídeo a ser rerproduzido.

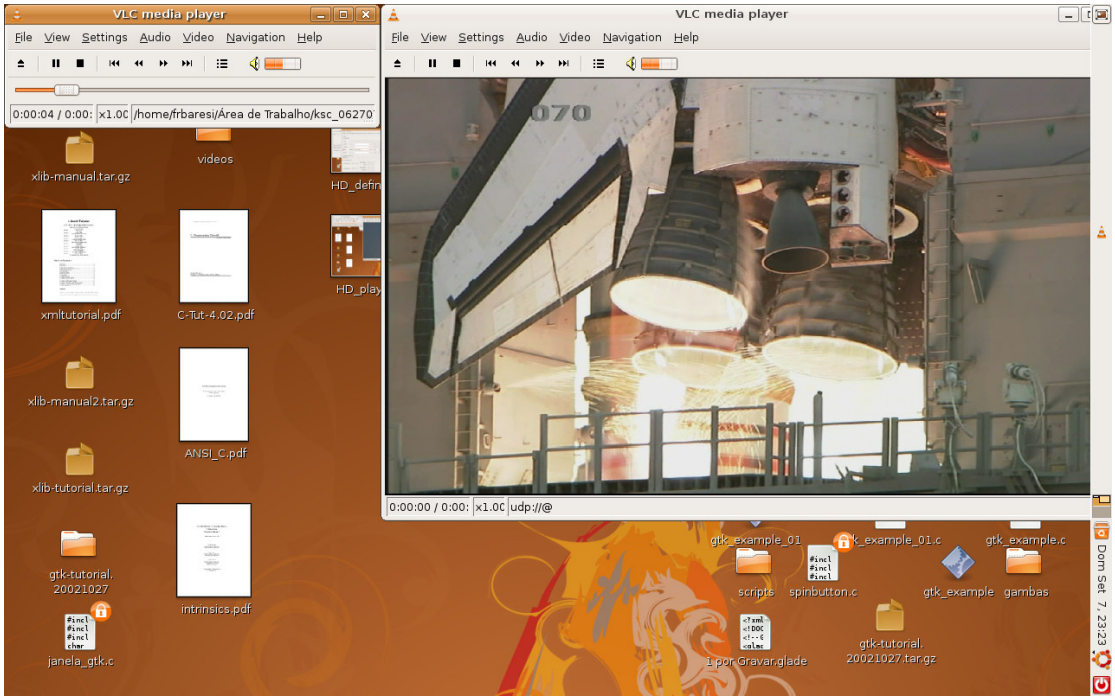


Figura 14 Visualização de video HD usando *streaming*

No entanto, ao testar a reprodução dos vídeos HD para um PC AMD K7 com um processador de 500MHz e 128Mb de memória RAM, não se conseguiu uma correcta visualização dos vídeos. Para o vídeo com maior resolução, a imagem bloqueia, não havendo um "refrescamento" da imagem do ecrã enquanto o vídeo está a ser reproduzido. Para o vídeo de menor resolução, a imagem "arrasta-se" havendo um refrescamento da imagem a cada 2 a 3 segundos.

Isto indiciava que para um *hardware* de recepção melhor, os vídeos seriam reproduzidos correctamente. Para testar tal pensamento, testou-se o envio do vídeo para um PentiumIV a 2.80GHz com 512 MB de RAM. Como seria de esperar, os vídeos correram na perfeição.

Para finalizar os testes relativos à capacidade do VLC, achou-se importante testar a gravação de um vídeo HD para o disco. Apesar de, neste caso, se tratar de fazer uma cópia do ficheiro que está a ser reproduzido, este teste permite testar a gravação para disco de um canal HD que possa estar a ser reproduzido usando o VLC. Após ter sido feito esse teste, como seria de esperar, pois já se tinha visto que o VLC conseguia enviar *streams* do video HD, o VLC consegue gravar sem problemas conteúdo HD que está a ser reproduzido.

4.11. VoD

Um serviço bastante utilizado é o VoD, que é sabido, permite ao utilizador parar, avançar, ou retroceder o vídeo sempre que assim o desejar. No entanto, este serviço consome bastantes recursos na rede (largura de banda devido a se tratarem de ligações *unicast*) e no servidor VoD, se estiverem bastantes utilizadores a aceder ao serviço. Convém ainda salientar que um serviço destes faz uso de HTTP *streaming*.

Para a implementação de uma solução destas ser possível usando o VLC, é necessário ter a correr do lado do servidor VoD um servidor Web (Apache por exemplo) que contém os ficheiros de vídeo desejados. Para explicar como um cliente pode aceder ao servidor VoD, supõe-se que o servidor tem um endereço IP 172.16.12.1 e os vídeos estão contidos na pasta "vídeos" do servidor Apache. Uma vez configurada a solução do lado do servidor, para o utilizador aceder aos ficheiros contidos no servidor, basta usar na *shell* o seguinte comando

```
%vlc http://172.16.12.1:8080/videos/nomedovideo.mpg
```

4.12. GTK

Uma vez que a janela produzida usando apenas Xlib não é visualmente apelativa nem dinâmica, optou-se por criar uma janela usando GTK, pois este pacote de bibliotecas permite uma programação a um nível superior.

O GTK é um *toolkit* que permite a criação de interfaces gráficos mais apelativos usando linguagem C e uma programação orientada a objectos. Este *toolkit* foi usado, pois permite trabalhar com o sistema de janelas X a um nível pouco superior que o Xlib e, ao mesmo tempo, criar interfaces gráficos com maior qualidade e facilidade do que os que podem ser obtidos usando Xlib.

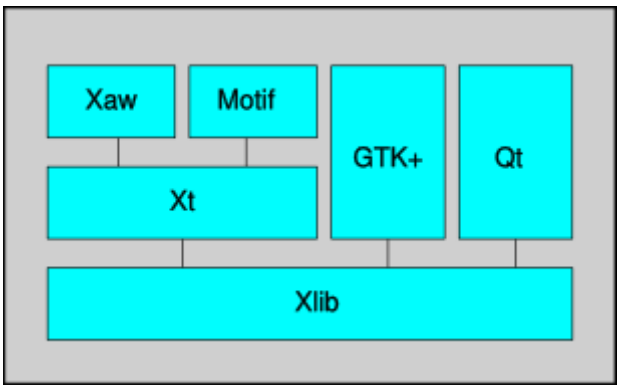


Figura 15 Xlib/GTK

Como se pode ver na figura acima, o GTK opera directamente com o Xlib sendo que, por esse motivo, os princípios que regem a programação usando GTK são semelhantes aos usados na programação apenas com a biblioteca Xlib. Por esse motivo, e uma vez que o mais importante no estudo feito para a Xlib (Anexo B) era compreender o funcionamento do sistema de janelas X, não foi feito nenhum estudo relativo à programação em C usando GTK. Assim, conforme se for usando o GTK, os princípios da sua programação vão sendo explicados mas tendo sempre como base as informações presentes nas referências. No entanto, por se tratar de um princípio importante para a programação usando GTK, refere-se que para ser possível trabalhar dentro de uma janela GTK é necessário ir dividindo esta usando caixas verticais e/ou horizontais. [28][29][30][31].

5. SOLUÇÃO A IMPLEMENTAR

Ao serem conhecidas as soluções de IPTV existentes no mercado, saltam à vista algumas limitações, principalmente o facto destas restringirem o utilizador quanto ao número de *set-top-box* disponibilizadas. Como já foi dito, o que limita o número de *boxes* é a largura de banda porque os fornecedores de serviço apenas disponibilizam um número de *boxes* igual ao número de canais que um cliente pode ver em simultâneo. Esse motivo leva a delinear uma solução que não limite o número de *set-top-box*, o que é possível introduzindo na casa de cada cliente um servidor responsável por distribuir os conteúdos de vídeo para cada *box* situada junto à TV.

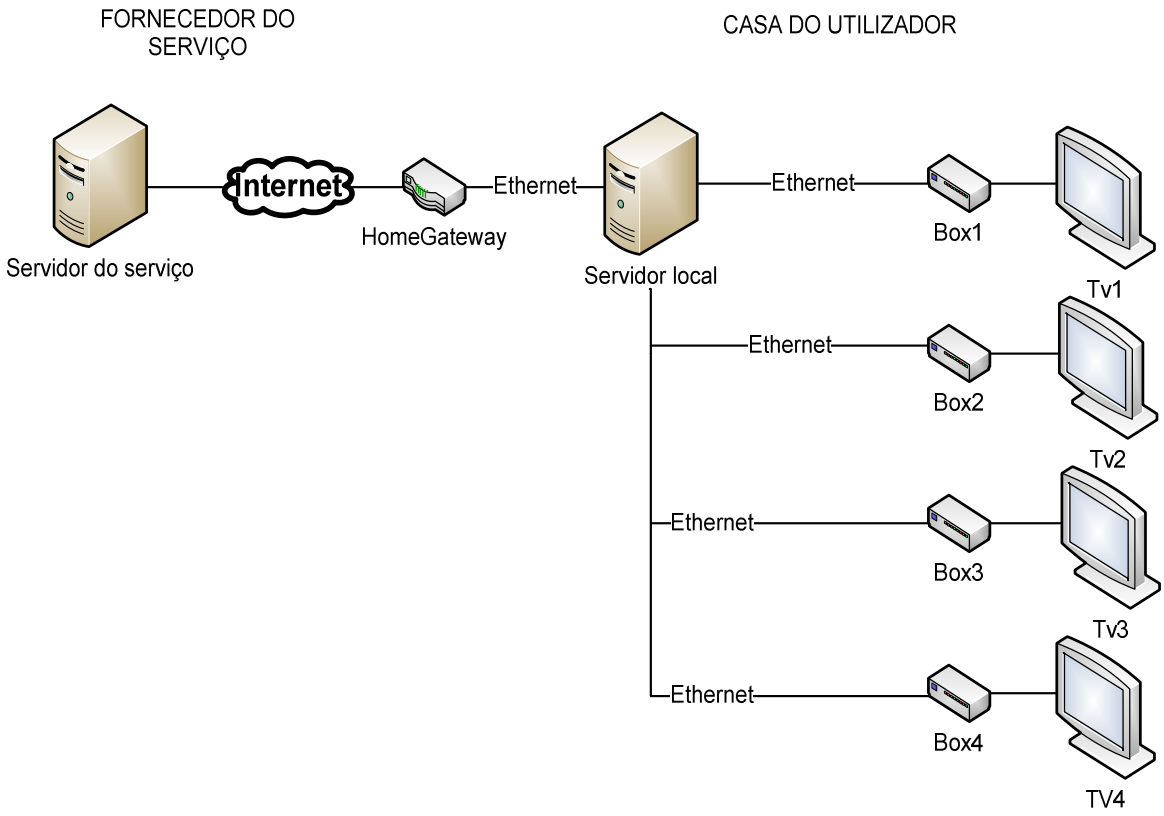


Figura 16 Arquitectura a implementar

O facto de ser usado o servidor local, ilustrado na figura anterior, proporciona uma capacidade de tratar e gerir os conteúdos recebidos e distribuí-los para as várias boxes. Com esta gestão, torna-se possível contornar as limitações impostas pelo fornecedor de serviço.

Supõe-se agora um caso em que a limitação no número de boxes é dois e estão a ver TV na mesma casa dois utilizadores em televisões diferentes. Se um terceiro utilizador quer ver TV numa terceira televisão, é lhe dada essa oportunidade, desde que este veja o mesmo canal de TV que um dos outros dois está a ver, ou que veja conteúdos gravados no servidor local. Tal, é permitido, colocando o servidor local a distribuir os canais usando endereços de multicast privados, ou seja, este junta-se ao endereço multicast do canal pedido por um utilizador e disponibiliza-o num endereço multicast privado.

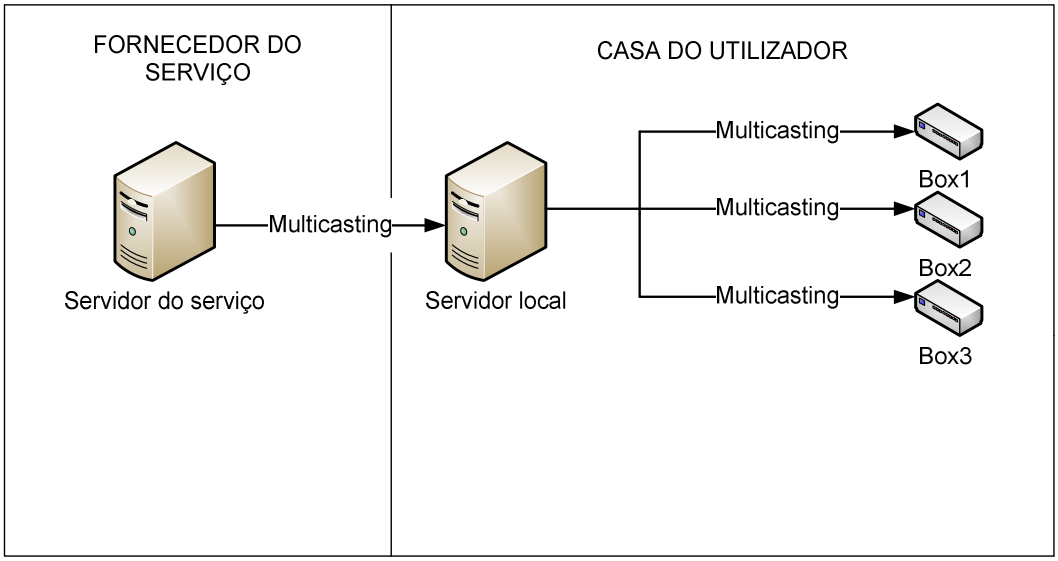


Figura 17 Tipo de ligações para transmitir vídeo

Entre os benefícios no uso do servidor local, destaca-se a possibilidade de haver uma "interacção" entre as *boxes*, ou seja, se a *box*1 colocar a gravar um conteúdo, a *box*3 pode aceder a este, pois esta gravação fica alojada no servidor local. Com uma arquitectura igual à oferecida no mercado, um conteúdo gravado por uma *box* não pode ser acedido por outra *box*.

Como é de prever, a adição do servidor local permite a criação de outros serviços e oferece novas possibilidades. O uso deste servidor local permite:

- Colocar conteúdos (vídeos pessoais, músicas, etc.) no servidor para que os utilizadores nas várias *boxes* lhes possam aceder
- Criar no servidor local um servidor de VoD
- Colocar o servidor local a gerir os pedidos do utilizador ao servidor do serviço
- Disponibilizar nas várias *boxes* acesso à *Internet*

Pelos motivos anteriormente referidos, decidiu-se projectar e implementar a parte relativa à "casa do utilizador", ilustrada na Figura 16.

Tendo em conta os testes preliminares, é importante seleccionar, entre as hipóteses testadas, as que vão ser usadas no desenvolvimento desta solução. Assim, neste ponto é definido o *software* a utilizar quer nas *boxes* quer no servidor local.

Antes de mais, convém salientar que se optou por *software* livre para não haver custos inerentes ao *software* utilizado.

De modo a haver uma comunicação entre cada *box* e o servidor local, é necessária uma ligação servidor-cliente, sendo o servidor local o servidor dessa ligação e cada uma das *boxes* o cliente. Assim, para ligar o servidor aos clientes, é fundamental definir uma rede privada.

Para promover a comunicação entre cliente e servidor, será usada a ligação TCP/IP desenvolvida anteriormente, pois permite uma fácil comunicação entre servidor e cliente e um envio de dados mais consistente quando comparada com a comunicação usando o protocolo X. Com este modelo servidor-cliente, para enviar um vídeo recebido pelo servidor para o cliente, é usado o VLC que, através do uso do protocolo RTP, envia *streams* do canal a reproduzir usando o MPEG-TS.

O interface gráfico a usar, faz uso do pacote Libvlc associado aos pacotes de bibliotecas Xlib e GTK. Tal escolha deve-se ao facto de, assim, ser possível uma programação a um nível mais baixo, ao mesmo tempo que se garante a possibilidade de receber interacções do utilizador. Comparativamente ao *plugin* para um *browser*, a grande vantagem é que este pacote não se necessita de um *browser*, que, se fosse necessário, se tornaria "pesado" para a aplicação. Comparando o uso do Libvlc com o uso do Gambas, salta à vista que o facto de ser necessário "capturar" a janela desejada através do seu nome, é uma desvantagem. Também o facto do Gambas ser programado a um nível mais elevado, que o GTK e que o Xlib, constitui uma desvantagem, pois torna o código menos eficiente.

No que diz respeito aos programas que criam ficheiros XMLTV, o facto do xmltv-druid não necessitar do emulador Wine para operar, constitui uma grande vantagem. Para visualizar a programação guardada nos ficheiros XMLTV e, de forma a criar um guia com a estrutura desejada, decidiu-se desenvolver um interface gráfico usando o pacote de bibliotecas GTK.

Para haver uma comunicação entre o utilizador e as *boxes* é usado um telecomando com receptor de infravermelhos que é configurado usando a biblioteca Lirc.

Ilustra-se então um protótipo da solução desejada, indicando o *software* usado em cada constituinte desta.

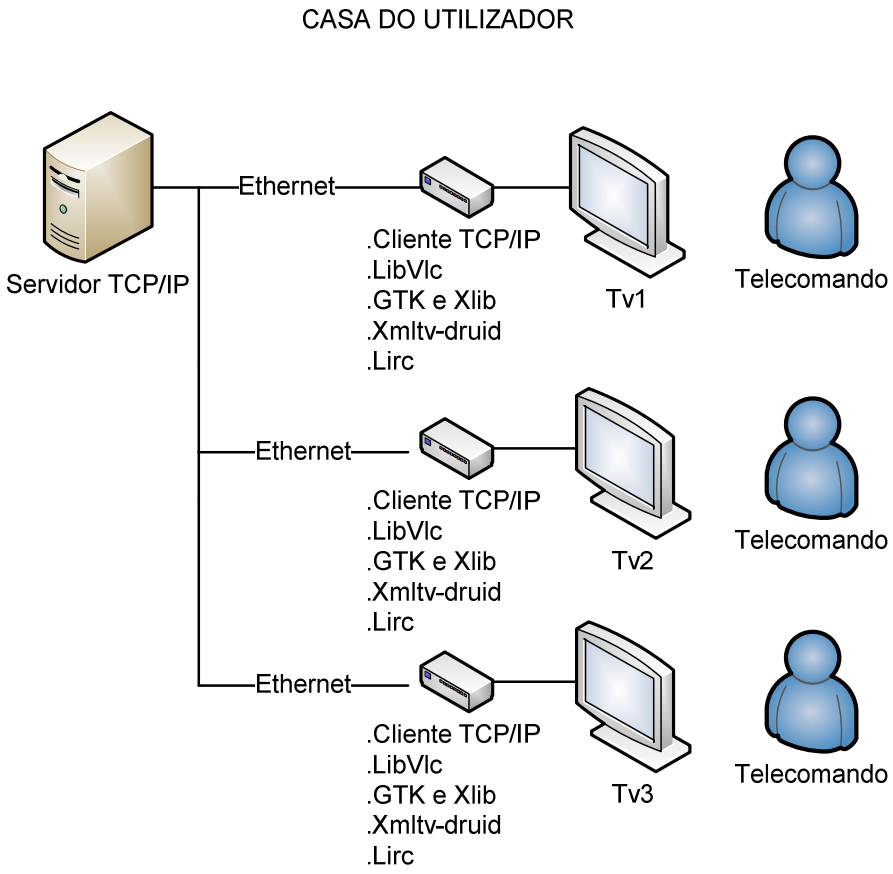


Figura 18 Protótipo funcional

6. IMPLEMENTAÇÃO DO SERVIDOR

A missão deste servidor é receber vídeos do servidor central de serviço e enviá-los para o cliente que os pediu. No entanto, este servidor deve também, a pedido do utilizador, guardar vídeos no disco.

6.1. HARDWARE

Segundo a documentação relativa ao VLC, um Pentium 100MHz com 32 MB de memória deve ser suficiente para enviar uma *stream* para a rede. No entanto, o envio de *streams* de vários vídeos alojados num disco duro é limitado pelo disco duro e pela velocidade da ligação à rede e não pela velocidade do processador. Por estes motivos, deverá ser usada uma máquina com grande capacidade de armazenamento em disco e um bom desempenho em comunicações TCP/IP. De salientar ainda que a capacidade de processamento não é crítica.

O protótipo de servidor foi implementado com base num computador do tipo PC, com um processador Intel Centrino Duo 1.5GHz e 2 Gb de memória RAM. Uma vez a correr a

aplicação relativa ao servidor, o PC poderia perfeitamente continuar a realizar outras tarefas.

Visto que o servidor não necessita de interface gráfico, numa solução final, poderia ser utilizado um computador sem placa gráfica, de forma a reduzir o custo deste *hardware*.

6.2. SOFTWARE

Uma vez que a missão deste servidor é receber vídeos do servidor central de serviço e enviá-los para o cliente que os pediu, o protótipo de servidor foi implementado usando como sistema operativo a distribuição Linux Ubuntu. Esta escolha deve-se ao facto do *hardware* seleccionado funcionar simultaneamente como servidor e computador pessoal.

6.2.1. SERVIDOR TCP/IP

De forma a satisfazer os objectivos deste servidor, desenvolveu-se um servidor TCP/IP capaz de receber os pedidos efectuados pelo utilizador, que, uma vez recebidos, são tratados de forma a serem disponibilizados ao utilizador que os pediu. Assim, este servidor deve receber o número de canal pedido pelo utilizador, abandonar o endereço *multicast* a que está ligado, juntar-se ao endereço *multicast* representativo do canal pedido e fazer *streaming* RTP do que está a ler no endereço *multicast.*

Caso o utilizador deseje ver um vídeo previamente gravado, o servidor deve sair do endereço *multicast* a que está ligado, abrir o vídeo com o nome recebido e, em vez de o reproduzir, fazer *streaming* para o cliente usando o protocolo HTTP, para permitir ao utilizador as opções de: parar, reproduzir, retroceder, avançar.

Quando um pedido de gravação é recebido, deve usar na linha de comandos o comando `crontab` para agendar o início e o fim da gravação.

No desenvolvimento deste servidor foram usadas as bases do servidor TCP/IP criado durante os testes preliminares (4.5.2). Por esse motivo não será explicado o código necessário para criar um servidor TCP/IP capaz de receber dados. Passa-se por isso a explicar o procedimento necessário para a construção deste servidor.

Criou-se então um *socket*, editou-se a sua estrutura, ligou-se o *socket* e aguardou-se por pedidos de ligação. A partir deste momento, criou-se um ciclo infinito para aceitar os

pedidos de ligação. Ao ser aceite um pedido de ligação, é criado um processo filho, que recebe o identificador do *socket*, para tratar dessa ligação. Assim, é possível ter mais do que um cliente (TV) por servidor, uma vez que é criado um processo para tratar de cada cliente.

Em cada processo filho são lidos os dados enviados pelo cliente TCP/IP (uma estrutura) e consoante o valor presente no identificador, o servidor TCP/IP trata da forma correcta os dados presentes no outro elemento da estrutura.

```
struct my_msg{

int elem;// identificador do tipo de dados (numero de canal, nome de
vídeo contido no servidor)

char frase[SIZE];//dados a enviar

};
```

Assim, definiu-se que, caso o inteiro `elem` seja igual 1, os dados presentes no elemento `frase` representam o canal que o utilizador deseja ver; se é igual a 2, os dados representam as opções de gravação (hora e dia); já se for igual a 3, os dados representam o nome do vídeo gravado que o utilizador deseja ver. Este mecanismo permite, no futuro, adicionar funcionalidades a esta solução, bastando para tal acrescentar ao servidor um ciclo referente ao caso da variável `elem` ser igual a X.

Para mudar de canal, é lido o conteúdo da variável `frase` e o número que lá é lido é concatenado a um endereço *multicast* incompleto, ou seja, como não se sabe qual o endereço *multicast* que, na realidade, corresponde a um determinado canal, definiu-se, para efeitos de simulação, que o endereço *multicast* correspondente aos vários canais disponíveis é dado por:

```
233.1.1. + Número do canal

Os números dos canais são definidos pela ordem alfabética dos canais.
Por exemplo, o canal CNN é o número 12 e o seu endereço multicast é
233.1.1.12.
```

Para colocar a gravar o que está a ser reproduzido num determinado dia a uma determinada hora, o servidor lê o conteúdo da variável `frase` para, posteriormente, poder colocar a gravar. Quando se deseja gravar, a variável `frase` tem o seguinte formato:ddmmaaaahhmmhhmmc; ou seja, dia de gravação, mês, ano, hora e minutos de início de gravação, hora e minutos de fim de gravação e número do canal a gravar. Assim, para colocar a gravar, é necessário parcelar o conteúdo recebido e colocar a gravar usando

o comando Linux `crontab`. Depois de parcelado o ficheiro, é aberto o ficheiro de configuração do `crontab`, que necessita de ser editado para agendar tarefas. Este comando permite o agendamento de tarefas periodicamente (como é o caso da obtenção do ficheiro XMLTV), ou singularmente. No ficheiro de edição desta agenda de tarefas estão presentes seis campos:

- Minuto (0-59)
- Hora (0-23)
- Dia do mês (1-31)
- Mês (1-12)
- Dia da semana (0-6, sendo domingo 0 e segunda feira 1)
- Comando a executar

Caso um dos campos (excepto o relativo ao comando a executar) for preenchido com um asterisco, esse campo vale "todos os valores possíveis". Por exemplo, se o campo dos minutos contiver um asterisco, o comando é executado todos os minutos de determinada hora, dia, mês e dia da semana. Assim, depois de parcelada a informação relativa a uma pedido de gravação no dia 07 do mês 06, às 05 horas e 04 minutos e parar às 07 horas e 17 minutos do mesmo dia, o ficheiro `crontab` deverá conter:

```
04 05 07 06 * ./gravar

17 07 07 06 * ./parar_gravação
```

Assim, é necessário abrir o ficheiro de configuração no modo de escrita e escrever no fim deste (para não apagar as outras tarefas agendadas) duas linhas: uma para agendar o início de gravação, e outra para agendar o fim de gravação [24].

```
fl=fopen("/home/user/crontab","a");

…

fputs(linha,fl); //linha contém a configuração de início de gravação a
por no ficheiro

fputs(linha,fl); //linha contém a configuração de fim de gravação a por
no ficheiro

system("crontab /home/User/crontab"); //aplica o ficheiro editado ao
comando crontab
```

Para iniciar a gravação é chamado um pequeno *script*, que consiste em chamar o VLC e colocá-lo a gravar o canal desejado para um ficheiro alojado no directório dos ficheiros vídeo e com um nome definido pelo formato:`nrcanal:ddmmaaa_hhmm_ate_hhmm`

Para parar a gravação, é chamado um pequeno *script* que lê o ID de processo (PID) do VLC que iniciou a gravação e, usando esse PID termina a gravação do VLC correspondente.

Para abrir um vídeo gravado, é usado o conteúdo da variável `frase`. Uma vez que esta variável contém o nome exacto do vídeo desejado, o servidor apenas necessita de concatenar esse nome ao directório que contém os vídeos e colocar o VLC a fazer *streaming* desse vídeo. Neste caso é usada uma ligação *unicast*

Exemplo:

```
system("vlc /home/user/vídeos/12:12072008_1255_ate_1530 --sout
'#transcode{vcodec=mp2v,vb=1024,scale=1,acodec=mp2a,ab=192,channels=2}:
duplicate{dst=std{access=rtp,mux=ts,dst=172.16.12.1:1234}}'&");
```

O fluxograma seguinte representa o funcionamento deste servidor.

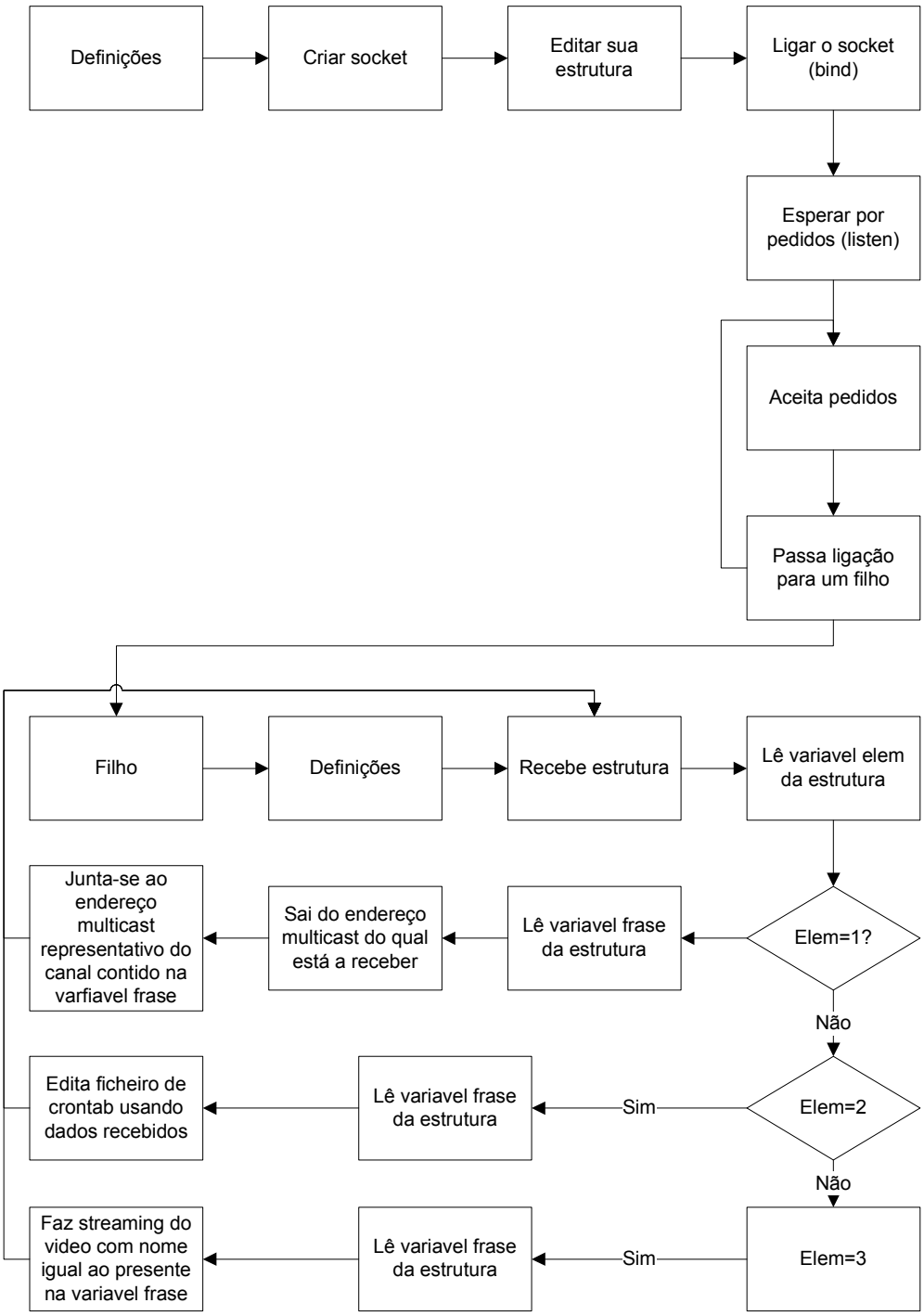


Figura 19 Fluxograma representativo do funcionamento do servidor TCP/IP

7. IMPLEMENTAÇÃO DO CLIENTE

A função do cliente é receber um vídeo, enviá-lo para o ecrã, correr uma aplicação de controlo e permitir ao utilizador o uso de um comando.

7.1. HARDWARE

Durante a implementação do protótipo, foi usado um computador do tipo PC, um AMD K7 com um processador de 500MHz e 128Mb de memória RAM. Embora este computador tenha um desempenho positivo, um computador com uma capacidade de processamento um pouco maior, permitiria um bom desempenho.

Uma vez que o cliente não necessita de armazenar conteúdos e que os vídeos que o utilizador deseja gravar são gravados no servidor, numa solução final poderia ser usado um computador sem disco.

7.1.1. COMANDO

De forma a permitir ao utilizador interagir com a aplicação, é usado um telecomando. O telecomando usado tem um receptor de infravermelho que é ligado ao computador do cliente. O comando usado é um: Toshiba MCE Remote Control-PX1246R-1ETC.

Figura 20 Toshiba MCE Remote Control-PX1246R-1ETC

7.2. SOFTWARE

Apesar de se ter pensado em desenvolver um sistema embebido, que arrancasse a partir de uma *pen* USB e corresse a aplicação relativa ao cliente, optou-se por instalar, no computador relativo ao cliente, uma distribuição Linux leve e robusta, o Xubuntu. Esta opção por uma distribuição Linux do tipo *desktop,* deveu-se ao facto de assim se poder continuar a usar o computador relativo ao Cliente como computador pessoal. Tal opção, torna-se ainda mais vantajosa, devido ao facto do computador pessoal poder ser colocado junto a uma TV e continuar a ser operado usando dispositivos sem fios. Ou seja, o utilizador pode correr o protótipo da solução de TV Digital sempre que quiser e interagir com esta usando o comando IR ou usar o computador como sistema operativo e interagir com este usando um rato e um teclado *wireless*.

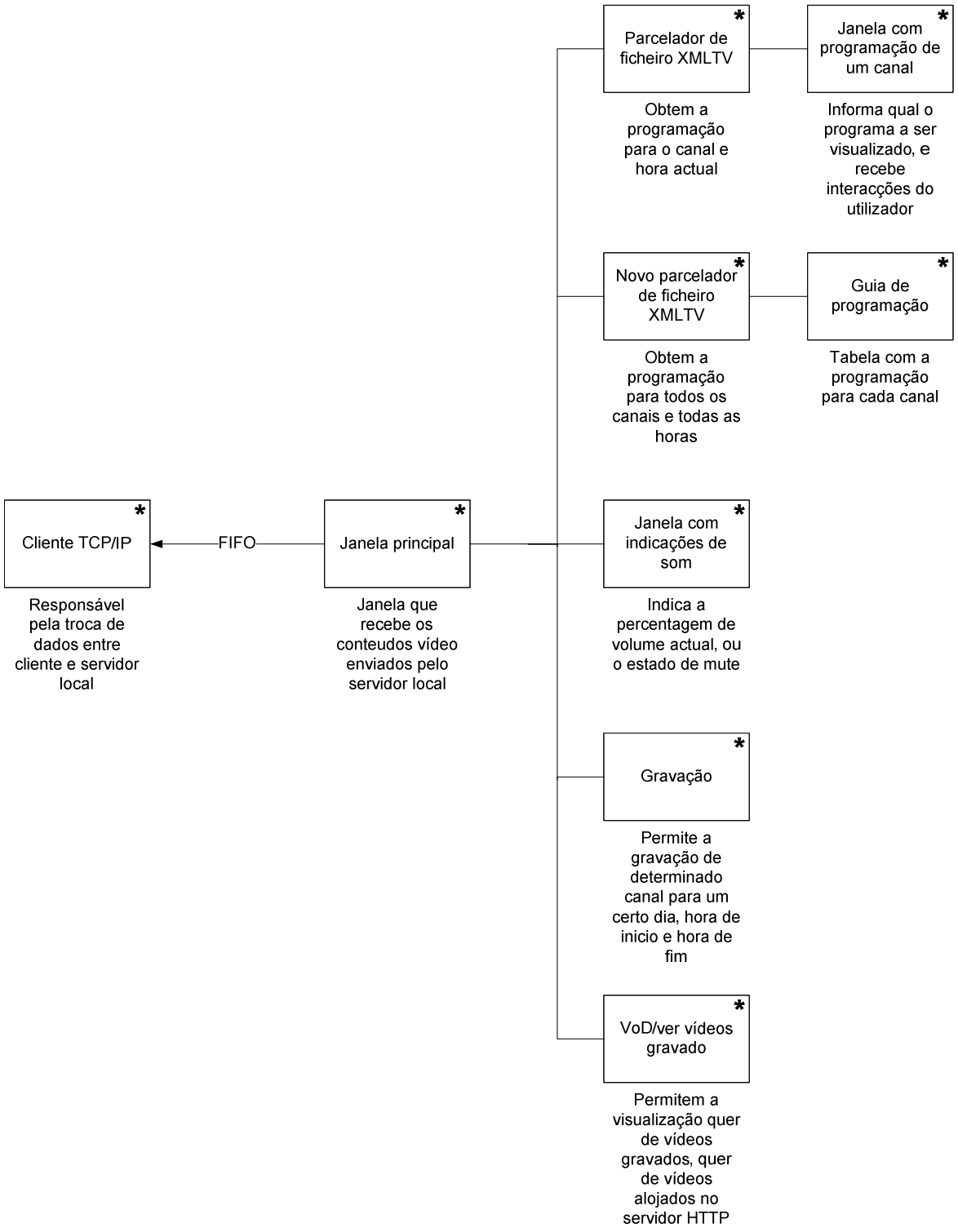


Figura 21 Organização dos vários módulos de software desenvolvidos para o cliente

7.2.1. JANELA PRINCIPAL

Esta janela é a responsável pela reprodução do que está a ser enviado pelo servidor. Para tal, esta janela é criada usando a biblioteca `libvlc.h`, pois esta biblioteca permite usar algumas das propriedades do VLC. Para poder configurar janela e captar algumas das interacções do utilizador é usada a biblioteca `XLib.h`. Caso o utilizador prima uma tecla do comando, esta deve ser lida e tratada; se a tecla premida for para alterar o volume, para cortar o som, ou para fazer uma pausa a tecla é tratada localmente; se a tecla premida for para uma mudança de canal, para colocar a gravar, para ler um vídeo gravado ou para ver a programação é usada outra janela para tratá-las.

Para desenvolver esta janela, foi usada como base a janela criada com a Libvlc nos testes preliminares, na secção 4.6.3

Para a janela ocupar o ecrã todo, foi necessário, após colocar o vídeo a reproduzir, saber a entrada que está a ser reproduzida e colocá-la em *fullscreen.* Para tal, foi acrescentado o seguinte código:

```
Inp= libvlc_playlist_get_input (inst, &excp);

Libvlc_set_fullscreen (inp, 1, &excp);
```

De forma a permitir a interacção do utilizador com esta janela foi usada a biblioteca Xlib (documentada no Anexo B). Assim, para detectar as interacções do utilizador, é necessário detectar as teclas pressionadas por este. Para tal ser possível, foi necessário criar uma ligação ao servidor X, saber o id da janela criada com o Libvlc, seleccionar o evento desejado e, por fim, ler o evento. Para realizar essa tarefa foi usado o seguinte código:

```
Display *d;

Window w;

XEvent e;//estrutura do evento

D=XopenDisplay(NULL);//Abre ligação usando o valor da variável de
ambiente DISPLAY

XGetInputFocus (d, &w, &rev_to_ret);// Saber o id da janela criada com
o libvlc

XSelectInput (d, w, KeyPressMask); //Seleccionar os eventos a receber

XSync (d, 1); //limpar a fila de eventos antes de a ler

XNextEvent (d, &e); //ler o próximo evento da fila

Switch (e.type){ // saber se é o tipo de evento desejado
```

```
          Case KeyPress:

            Keysym key_symbol=XkeycodeToKeysym(d,e.xkey.keycode,0); //
passa de keycode para keysymbol
```

Uma vez que o vídeo a reproduzir é um vídeo que está a ser enviado pelo servidor, no código da secção 4.6.3 teve que se mudar o nome do ficheiro a ser reproduzido. Assim, onde aparece: `char *filename = "/home/videos/2004-10-30_pl_arsenal-southampton_2-2_van_persie.wmv"` deve ser colocado: `char *filename="udp://@"` para receber as *streams unicast* RTP enviadas pelo servidor ou `char *filename="udp://@233.1.1.3"` para receber *streams multicast*.

Para garantir a recepção de todas as interacções do utilizador com esta janela, o código relativo ao processo de leitura dos eventos teve de ser colocado num ciclo, apenas interrompido para tratar as teclas recebidas.

Mediante a tecla que é recebida, o tratamento desta pode variar. Caso o utilizador tenha pressionado um número, é usado o módulo "Janela com a programação de um canal" (que posteriormente será explicado). Se o utilizador seleccionar ver programação, ver vídeo guardado ou gravar um vídeo, é lançada a aplicação relativa a estes pedidos. O passo seguinte foi proporcionar uma resposta a um pedido de alteração do volume. Para tal, cada vez que é lido um pedido de mudança do volume, é necessário, antes de mais, ler o valor actual do volume usando a função `libvlc_audio_get_volume(inst, &excp)`. O valor retornado por esta função é um inteiro entre 0 e 100, que indica a percentagem do volume. Para definir o novo volume, é usada a função `libvlc_audio_set_volume (inst, volumedesejado, &excp)`, onde a variável `volumedesejado` representa a nova percentagem de volume desejada. Assim, se o utilizador deseja aumentar o volume, a variável `volumedesejado` é incrementada enquanto que se deseja diminuir o volume, a variável `volumedesejado` é decrementada. Cada vez que é alterado o volume é lançada uma janela que indica o volume.

Seguidamente, foi utilizada a função `libvlc_audio_toogle_mute (inst, &excp)` para cortar ou colocar o som, caso a tecla que o utilizador premiu assim o pedisse. Quando esta tecla é premida e o som é cortado, é lançada uma janela que indica que o som está cortado, ficando esta janela no ecrã até o utilizador pressiona a tecla para colocar o som. Uma vez lançada a janela que indica o corte do som, tem que lhe ser

retirado o *focus* para que esta janela principal continue a receber as teclas pressionadas pelo utilizador.

Para permitir ao utilizador a paragem em tempo real, usou-se a função `libvlc_playlist_pause (inst, &excp)`. Esta função é usada cada vez que o utilizador prime a tecla *pause* do seu comando. Ao ser premida esta tecla é mudado o estado *pause,* ou seja, é usado o seguinte princípio: se está em *pause* muda para *play*; se está em *play* muda para *pause*.

A última funcionalidade que se implementou foi a mudança de aspecto visual. Ao ser carregada a tecla relativa a esta operação, o aspecto mudará automaticamente para os aspectos 16:9, 4:3, entre outros. Para tal ser possível, foi usada a função `libvlc_video_set_aspect_ratio()` com o aspecto desejado. No entanto, para esta mudança funcionar correctamente, antes de ser operada a mudança, é necessário saber o aspecto actual. Tal é possível usando:

`libvlc_video_get_aspect_ratio(inst,&excp).`

7.2.2. OBTER FICHEIRO XMLTV

De forma a se obter o ficheiro XMLTV é usado o programa xmltv-druid, que vai buscar a programação ao site da Tvcabo. O modo de utilizar e configurar este programa foi referido nos testes preliminares. No entanto convém relembrar que este programa necessita de uma configuração na sua primeira utilização sendo que nesta se escolhem os canais para os quais se quer obter a programação e a hora à qual se quer realizar a obtenção dos canais. Se nada for alterado, o ficheiro XMLTV obtido é guardado na pasta `/home/user/.xmltv/tv_grab_pt.xml`.

7.2.3. PARCELAR FICHEIRO XMLTV

A missão deste módulo é parcelar o ficheiro XMLTV de modo a obter a programação dos vários canais. Tal pode ser feito para todos os canais e todas as horas, ou para um único canal a uma hora específica. Para um único canal, é obtida a hora a que se acede à programação e faz-se um parcelamento, de modo a obter o programa que, àquela hora, está a dar naquele canal.

Uma vez que já era conhecida a organização do ficheiro, passou-se a usar o pacote de bibliotecas Libxml, de forma a ser possível parcelar o ficheiro, assim como obter a informação desejada. O estudo feito sobre esta biblioteca está documentado no Anexo C.

Passou-se assim a criar uma aplicação que receba um ficheiro XMLTV, a hora e data actuais, o número de um canal e, usando estes dados, parcele o documento de forma a se conhecer o programa que está a ser transmitido para um determinado dia, hora e canal.

Nesta aplicação, escrita em C, são usadas as bibliotecas `libxml/xmlmemory.h` e `libxml/parser.h`. Para começar, seleccionou-se o ficheiro XMLTV a parcelar e achou-se o nó raíz do ficheiro XMLTV. Para tal usou-se:

```
xmlChar *txt,*txt2,*txt3;

xmlDocPtr doc;

xmlNodePtr cur, sub, sub2;

doc=xmlParseFile (argv[1]); //Sendo argv[1] documento a parcelar

cur=xmlDocGetRootElement(doc); //Obter elemento raíz do documento

xmlStrcmp (cur->name, (const xmlChar *) "tv");//Ver se o elemento raíz
obtido é o correcto
```

O passo seguinte foi a obtenção de um valor de data e hora passível de ser comparado com os valores presentes no ficheiro XMLTV. No ficheiro XMLTV a data e a hora de início e fim de um programa têm o formato: aaaammddhhmm (exemplo: 200806250200). Para se conhecer a data e a hora actual, é usada a linha de comandos. Assim, na linha de comandos, deve ser usado `date +%Y%m%d` para saber a data no formato aaaammdd e `date +%H%M` para obter a hora no formato hhmm. A data e a hora obtida com esta chamada de sistema serão dois dos parâmetros recebidos por esta aplicação.

De forma a ser possível estabelecer uma ligação entre o número de um canal e um dos canais do ficheiro XMLTV, foi criada uma aplicação que recebe um número de entrada e, para esse número, devolve o nome do canal correspondente. Para tal, foi simplesmente usada uma tabela de correspondências número/canal. De salientar que o facto desta tabela ter este formato, permite que, no futuro, se desejado, se possa desenvolver uma aplicação que permita ao utilizador mudar a lista de canais. O nome de canal obtido é também um dos parâmetros enviados para a aplicação que parcela o ficheiro XMLTV.

Seguidamente, desceu-se um nível no ficheiro XMLTV, para o 2º nível, e procurou-se pelo sub-nó `programme`. Usou-se por isso `sub=cur->children` para efectuar a referida descida de nível e a comparação `if( (!xmlStrcmp(sub->name, (const xmlChar *)"programme" ))` para apenas se trabalhar no sub-nó desejado.

Uma vez a trabalhar no sub-nó desejado, procurou-se obter o valor das propriedades do sub-nó `programme`, para estas poderem ser comparadas com as variáveis de entrada desta aplicação. Assim, usou-se:

```
txt = xmlGetProp( sub, (const xmlChar *)"channel" );
txt2 = xmlGetProp( sub, (const xmlChar *)"start" );
txt3= xmlGetProp( sub, (const xmlChar *)"stop" );
```

para se poder obter o valor das propriedades .

Caso o nome do canal de entrada seja igual ao valor da propriedade `channel`, passa-se a comparar a data. Caso ambas as comparações sejam válidas, compara-se a hora actual com a hora de ínício e de fim do programa. Se a hora actual for maior que a hora de ínicio e menor que a hora de fim, desce-se mais um nível para se poder obter a programação. Para obter a programação, visto esta não se tratar do valor de uma propriedade, é necessário ler o valor contido neste nó, usando `txt=xmlNodeListGetString (doc, sub2->xmlChildrenNode,1)`

O fluxograma seguinte representa o procedimento necessário para fazer o parcelamento do ficheiro XMLTV.

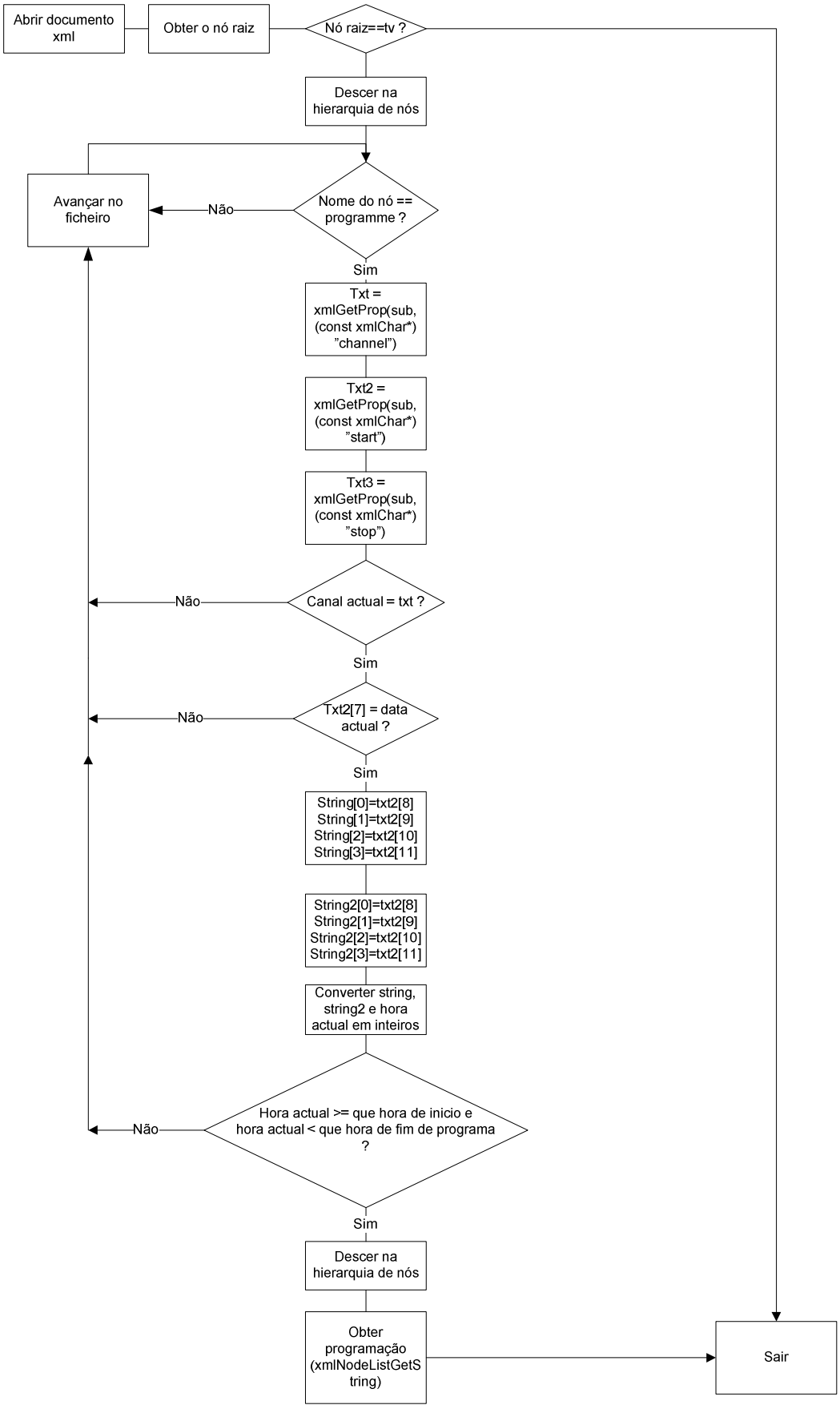


Figura 22 Fluxograma representativo do parcelamento

7.2.4. JANELA COM A PROGRAMAÇÃO DE UM CANAL

Esta janela é chamada quando o utilizador está na janela principal e este pressiona um número para mudar de canal. Deve identificar o programa que está a ser transmitido no canal actual (obtido através do parcelamento) e o número de canal que está a ser seleccionado. Caso o utilizador deseje mudar para um canal cujo número tenha dois dígitos, onde o primeiro dígito é o número premido na janela principal, a janela espera um segundo por esse segundo dígito. Quando o utilizador já escolheu canal que deseja ver, o seu número é enviado para o cliente TCP/IP. Esta janela recebe o número pressionado e a programação do canal actual.

Primeiramente, definiram-se as variáveis e *widgets* necessários, iniciou-se o GTK usando a função `gtk_init ()` e criou-se uma janela. Para criar a janela com a forma desejada usou-se:

- `window = gtk_window_new (GTK_WINDOW_TOPLEVEL);` que permite a criação de uma janela
- `gtk_window_set_decorated (GTK_WINDOW (window),0);` para remover a *task bar* (bordo da janela que inclui os botões de redimensionamento e o nome da janela)
- `gtk_widget_set_size_request (window, win_width-400,200);` para definir o tamanho da janela. De salientar que a obtenção da variável `win_width`, foi obtida usando a biblioteca `Xlib.h` e foi demonstrado na janela em Xlib (na secção4.9).
- `gtk_widget_set_uposition (window,200,win_height-200);` para definir a posição da janela no ecrã (coloca a janela no fundo do ecrã).

Criada a janela, dividiu-se esta em duas, de forma a permitir colocar nesta ambas as *labels* na posição desejada. Assim, usou-se :

- `vbox1=gtk_vbox_new (FALSE,0);` para criar duas caixas na vertical
- `gtk_widget_show(vbox1);` para as caixas serem vísiveis
- `gtk_container_add(GTK_CONTAINER(window),vbox1);` para adicionar as caixas à janela (o que divide a janela em dois)
- `label=gtk_label_new ("label");` para criar uma `label` com o texto "label"
- `gtk_widget_show(label);` para mostrar a `label`

- `gtk_box_pack_start (GTK_BOX (vbox1),label,FALSE,FALSE,0);` para adicionar a `label` à primeira das duas caixas criadas (a caixa mais acima)
- `label2=gt_label_new ("label2");` para criar uma *label* com o texto "label2"
- `gtk_widget_show (label2);` para mostrar a `label2`
- `gtk_box_pack_start` (GTK_BOX (vbox1),label2,FALSE,FALSE,0); para adicionar a `label2` à segunda das duas caixas criadas (a caixa mais abaixo).

Uma vez criada a estrutura gráfica base, foi necessário alinhar as *labels* para a posição desejada e definir o tamanho destas. Tal foi possível usando o código:

```
gtk_widget_set_size_request (label,1500, 100); //tamanho da
label:x=1500,y=100

gtk_widget_set_size_request (label2, 1500, 100); //tamanho da
label2:x=1500,y=100

gtk_misc_set_alignment (GTK_MISC (label), 0.05, 0.2);//alinhamento da
label. Mesmo alinhamento horizontal (0.05), mas diferente alinhamento
vertical (0.2 mais acima, 0.1 mais abaixo)

gtk_misc_set_alignment (GTK_MISC (label2), 0.05, 0.1);
```

Para a janela aparecer no ecrã, é necessário mostrar a janela (`gtk_widget_show(window)`,equivalente ao mapeamento usando Xlib), mostrar todo o seu conteúdo (`gtk_widget_show_all(window)`) e processar todos os eventos pendentes (`gtk_main()`). Nesta altura foi obtida uma janela com dimensões e localização desejadas, com uma letra de tamanho reduzido e incapaz de receber a interacção do utilizador. Para mudar a fonte da letra usou-se:

```
GtkSettings *settings; //definições globais do programa e não apenas da
janela

settings = gtk_settings_get_default(); g_object_set(G_OBJECT(settings),
"gtk-font-name", "Sans 24", NULL); //tamanho e letra desejada
```

Uma vez que esta aplicação recebe como primeiro parâmetro o número pressionado na janela principal e como segundo parâmetro a programação para o canal actual, colocou-se o valor destes parâmetros nas devidas *labels*.

```
strcpy(canal,argv[1]);

gtk_label_set_text(label,canal);

strcpy(texto,argv[2]);

gtk_label_set_text(label2,texto);
```

Para ser possível receber as teclas pressionadas pelo utilizador, simplesmente se selecciona o evento que se pretende receber (tal como no Xlib, onde se usava `XSelectInput (display, win, KeyPressMask)`) e a chamada que deve ser feita quando esse evento é recebido.

```
g_signal_connect ((gpointer) window, "key_press_event",G_CALLBACK
(on_label_key_press_event),(gpointer)label);
```

O código mostrado acima selecciona o evento `key_press`, chama a função `on_label_key_press_event()` quando um evento é recebido e passa para essa função o *widget* `label`.

Por essa razão, o passo seguinte foi a criação da função `on_label_key_press_event ()` com o intuito de ler o valor da tecla que gerou o evento (`event-> keyval`) e, caso se trate de um número, colocá-lo na janela. Uma vez que o objectivo é ler números enquanto o utilizador pressionar teclas e o número máximo de dígitos de um canal é dois, é necessário averiguar o tamanho do canal pressionado. Deste modo, se o tamanho actual do número de canal é igual a um (1), o valor lido no evento é concatenado a número actual; se for dois (2) o número da tecla pressionada substitui o número actual.

```
if (strlen(canal)==2) //canal representa o canal pressionado
             strcpy(canal,key);
           else //igual a 1
             strcat(canal,key);
```

Neste ponto, a aplicação produz uma janela que fica continuamente a ler canais até ser destruída.

A função `gtk_main()` processa os eventos pendentes, contudo deixa o programa "adormecido" à espera de eventos até esta janela ser destruída. Para contornar esta limitação, sustitui-se esta função `gtk_main()` por:

```
while(gtk_events_pending())
          gtk_main_iteration();
```

O código mostrado acima faz uma iteração à *main* de cada vez, ou seja, processa um evento enquanto existirem eventos pendentes. De salientar que a criação de uma janela, por exemplo, corresponde a um evento. Este código permite na mesma mostrar a janela, mas,

em vez de “encravar” a aplicação à espera de eventos até a janela ser destruída, avança no código. Ou seja, a seguir a este código, é possível definir de quanto em quanto tempo se quer ler os eventos recebidos. Portanto, usou-se o seguinte código para dar um (1) segundo ao utilizador para premir a segunda tecla e mais um (1) segundo, caso o utilizador se tiver enganado no número, voltar a poder mudar o número do canal desejado.

```
sleep(1); //espera 1 s por tecla

if(gtk_events_pending())//para receber apenas da primeira vez

          {

          gtk_main_iteration();//processa tecla

          sleep(1);//espera mais um segundo

          }

while(i==1)//mantém-se neste ciclo caso receba teclas todos os
segundos, valor colocado a 1 na on_label_key_press_event()

{

if(gtk_events_pending()) //verifica se recebeu teclas durante aquele
segundo

          {

          gtk_main_iteration();//processa evento (trata tecla recebida)

          sleep(1);

          }

else //se não recebeu tecla num segundo sai do ciclo

          i=0;

}
```

Quando o utilizador já definiu o canal que quer ver, esta aplicação envia o número de canal para o cliente TCP/IP usando o FIFO. O modo como é enviado usando o FIFO será explicado mais à frente.

Uma vez que não é totalmente perceptível o procedimento por detrás desta janela, seguidamente, é mostrado um fluxograma o representa.

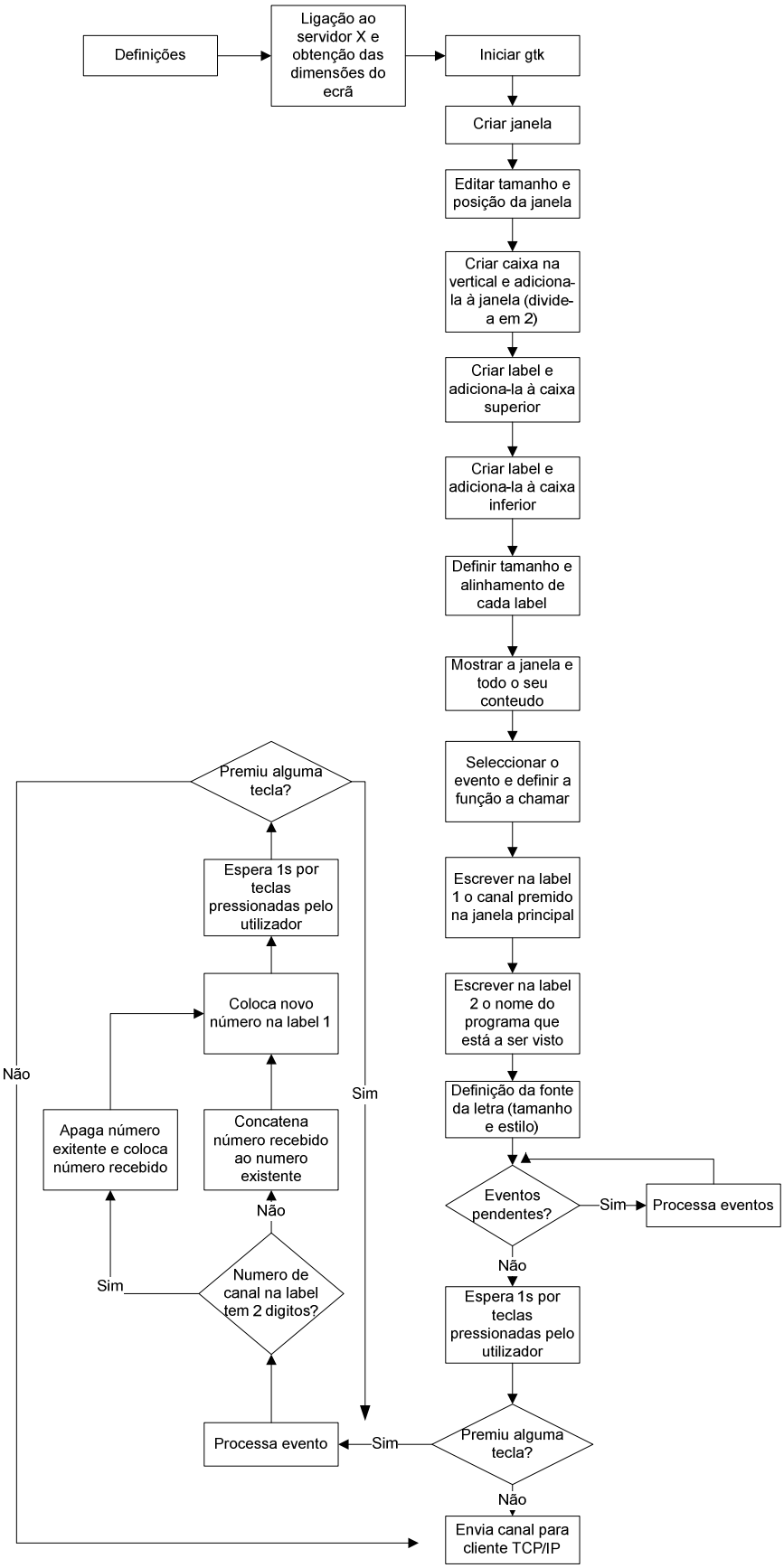


Figura 23 Fluxograma representativo do procedimento por detrás da janela GTK

7.2.5. JANELA COM INDICAÇÕES DO SOM

Esta janela foi criada para permitir indicar ao utilizador as modificações por ele efectuadas no volume.

Para criar esta janela, usou-se um procedimento semelhante ao usado para criar a "janela com a programação de um canal". As diferenças operadas são: tamanho da janela, posição da janela, parâmetros recebidos. Assim, optou-se então por uma janela um pouco acima do centro do ecrã, encostada ao lado esquerdo e com uma largura reduzida. Esta mudança na posição permite ter no ecrã esta janela a indicar "Sem Som" e ao mesmo tempo a janela com a programação de um canal.

Esta aplicação apenas receberá o texto a escrever na janela.

Tal como no procedimento usado para criar a janela com a programação de um canal, começou-se por definir os *widgets* necessários, ligar ao servidor X, iniciar o GTK e criar uma janela.

```
d=XOpenDisplay(display_name);//ligar ao servidor X

gtk_init (&argc, &argv);//iniciar gtk

window = gtk_window_new (GTK_WINDOW_TOPLEVEL);//criar janela

gtk_window_set_decorated (GTK_WINDOW (window),0);//retirar task bar da
janela
```

O passo seguinte foi o dimensionamento e a colocação da janela na posição desejada. Para saber as dimensões do ecrã, foram usadas, mais uma vez, funções da biblioteca Xlib.

```
win_width = DisplayWidth(d,s);

win_height = DisplayHeight(d,s);
```

Usando os dados aqui obtidos, dimensionou-se a janela:

```
gtk_widget_set_size_request (window, win_width-1050, 75);
```

Definiu-se a sua posição:

```
gtk_widget_set_uposition (window,0,win_height-500);
```

Para a janela ter a aparência desejada, falta apenas colocar na janela o texto que esta recebe. Necessitou-se assim de criar uma caixa vertical na janela e colocar nesta caixa uma *label* com o texto recebido.

```
label = gtk_label_new ("label");//criar label

strcpy(canal,argv[1]);//Copiar texto recebido no Texto a colocar na
label

gtk_label_set_text(label,canal);//Colocar na label o texto desejado
```

Uma vez que esta janela só deve aparecer durante um (1) segundo (excepto no caso de estar "sem som"), como foi explicado na secção7.2.4, após se mostrar a janela usando (`gtk_widget_show (window))`, não pode ser usada a função `gtk_main()`. Assim, necessitou-se de usar mais uma vez, o ciclo de leitura de eventos pendentes para fazer a janela aparecer.

```
while(gtk_events_pending())

      gtk_main_iteration();
```

Com a janela presente, aguarda-se um segundo para o leitor poder ver o texto presente na janela, para a seguir se sair da janela.

```
sleep(1);

gtk_main_quit();
```

7.2.6. JANELA COM TODA A PROGRAMAÇÃO

O objectivo desta janela é criar um guia para o utilizador ver toda a programação, canal a canal. Sendo assim, a janela deve ter rolamentos horizontais e verticais e demonstrar a programação dos vários canais em colunas verticais.

Mais uma vez, definiram-se as variáveis necessárias e iniciou-se o GTK (`gtk_init()`).

O passo seguinte é a criação de uma janela com as características desejadas, portanto criou-se uma janela de diálogo, para colocar uma janela com rolamentos lá dentro, cujo tamanho foi posteriormente definido.

```
gtk_widget_set_size_request (window,display_width , display_height);
//onde display_width e display_height são obtidos usando Xlib, e por
isso já foi demonstrado anteriormente como obtê-los
```

Para criar a janela com rolamentos usou-se a função `gtk_scrolled_window_new()`.

Uma vez que se deseja que os rolamentos, horizontal e vertical, estejam sempre presentes na janela, foi usada a opção `GTK_POLICY_ALWAYS`, quer para o rolamento horizontal

quer para o vertical. Caso se pretendesse que um rolamento aparecesse apenas quando necessário, teria de ser usada a opção `GTK_POLICY_AUTOMATIC`. Tal foi definido usando:

```
gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW
(scrolled_window),GTK_POLICY_ALWAYS, GTK_POLICY_ALWAYS);
```

Mais uma vez foi necessário usar uma caixa vertical e mostrar a janela com rolamentos

```
gtk_box_pack_start (GTK_BOX (GTK_DIALOG(window)-> vbox),
scrolled_window, TRUE, TRUE, 0);

gtk_widget_show (scrolled_window);
```

Para criar uma grelha com a programação, foi necessário criar uma tabela com um número de células horizontais igual ao número total de canais e com um número de células verticais igual a 24*2+1 (uma célula para cada meia hora mais uma para o nome do canal).

```
table = gtk_table_new (24*2+1, nrcanais, FALSE);

gtk_table_set_row_spacings (GTK_TABLE (table), 10);//para definir um
espaço de 10 para cada linha

gtk_table_set_col_spacings (GTK_TABLE (table), 10);//para definir um
espaço de 10 para cada coluna
```

Seguidamente, colocou-se a tabela na janela com rolamentos

```
gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW
(scrolled_window), table);

gtk_widget_show (table);
```

Posteriormente, preencheu-se a tabela com os campos desejados. Uma vez que a primeira linha contém os nomes dos canais, criou-se um ciclo que percorre a lista de canais e obtém os seus nomes. Para cada nome, é criada uma *label* com o nome do canal e esta *label* é colocada na devida posição.

```
j=0;

for (i = 1; i<101;i++)

            {

            …

            label=gtk_label_new (nomecanal)

            gtk_table_attach_defaults (GTK_TABLE (table),
label,i,i+1,j,j+1);

            }
```

Uma vez preenchida a linha referente ao nome dos canais, tratou-se de preencher as colunas relativas a cada canal. Mais uma vez, foi necessário usar um ciclo que corra todas as colunas e que, para cada uma delas, vá buscar toda a sua programação.

Já que, a aplicação desenvolvida anteriormente para o parcelamento do ficheiro XMLTV apenas retorna a programação para a hora actual, foi necessário desenvolver uma aplicação semelhante, mas que retorne para um ficheiro de texto a programação toda (incluído hora de início e de fim de programa) para um determinado canal e dia. Assim, em relação à aplicação de parcelamento já explicada na secção 7.2.3, foi inevitável remover a comparação feita entre a hora actual, a hora de início e a hora de fim, passando assim a haver acesso a toda a programação de um determinado canal. Cada vez que se encontra um programa relativo ao canal desejado, é guardado num ficheiro de texto o nome do programa, a hora de início e a hora de fim. O facto do ficheiro XMLTV ordenar por si só os programas por ordem de hora de início para hora de fim, faz com que o ficheiro de texto no qual é guardada a programação de um canal, também contenha a programação desse canal já ordenada.

```
j=1; //para começar na 2a linha porque a primeira tem a programação

for (i = 1; i<(nrcanais+1);i++)

    {

          system(parcelador); //sendo parcelador uma string que abre o
novo parcelador com o parâmetros de entrada o numero do canal e
redireccionando o seu output. Exemplo: ”./novoparcelador 15 >
/home/EPG2”

          f=fopen("/home/EPG2","r");//ficheiro que contem a programação
de um canal

          while(fgets(nomecanal,81,f))//enquanto houver algo para ler
lê

            {

            linha[datasize-1]=0;

            j++;//Cada linha que lê do ficheiro corresponde a uma linha
da tabela

            label=gtk_label_new (nomecanal)//cria uma label com o que
leu no ficheiro

             gtk_table_attach_defaults (GTK_TABLE (table),
label,i,i+1,j,j+1);   //colocar a label na posição desejada

            gtk_widget_show (label);//mostrar a label

            fclose(f);
```

```
        }
```

O ciclo anteriormente demonstrado, para cada coluna (cada canal), corre o novo parcelador e envia o seu *output* para um ficheiro de texto, entrando depois num ciclo de leitura do ficheiro de texto que cria, para cada linha desse ficheiro (cada programa), uma *label* com o texto contido nessa linha. O facto de, no novo parcelador, cada programa de determinado canal ter sido escrito no ficheiro de texto usando "`\n`" no fim, permite ler de cada vez apenas um programa (uma linha) da programação presente no ficheiro de texto. Para tal ser possível, usou-se a função `fgets()`.

Neste caso pode usar-se a função `gtk_main()`, pois só se pretende sair desta aplicação a mando do utilizador e não após um instante de tempo.

Como os rolamentos das janelas não respondem às teclas direccionais, foi necessário colocá-los a funcionar em reacção a essas teclas. Logo, seleccionou-se o evento de tecla pressionada e, mediante a tecla recebida, ajustou-se a posição dos rolamentos.

```
if(event->keyval == GDK_Right)

          {

            adj = gtk_scrolled_window_get_hadjustment(scrolled_window);
//para saber posição actual

            adj->value += 200;//aumento de 200

            if(adj->value >= adj->upper) //Se valor actual depois de ser
incrementado for maior ou igual ao valor máximo, valor actual passa a
ser igual ao valor máximo

                  adj->value = adj->upper;

            gtk_adjustment_value_changed(adj);

          }
```

No código mostrado acima, ao ser pressionado o cursor direito, define-se uma variável de ajustamento que, posteriormente, é usada para receber as características do rolamento horizontal. Nessas características estão presentes o valor da posição superior do rolamento, o valor da posição inferior e o valor da posição actual. Uma vez que o valor mínimo (a origem) se encontra na posição mais à esquerda possível, ao ser pressionado o cursor direito, o valor da posição actual tem que ser aumentado. No entanto, se com esse aumento se atingir ou ultrapassar o limite máximo, o valor actual passa a ser igual ao valor máximo. Por fim, aplicou-se esse valor. O mesmo raciocínio foi aplicado para o caso do utilizador pressionar o cursor esquerdo. No entanto, ao ser pressionada esta tecla, o valor actual do

rolamento é decrementado e, em vez de ser comparado com o valor superior, é comparado com o valor inferior.

Caso seja pressionado ou o cursor cima, ou o cursor baixo, o raciocínio é idêntico. Uma vez que a origem, neste caso, é a parte superior do ecrã, ao ser pressionado o cursor cima, o valor actual tem que ser diminuído enquanto que se for pressionado o cursor baixo, o valor actual tem que ser aumentado. Ou seja, o código relativo ao cursor direito é igual ao relativo ao cursor baixo, enquanto que código relativo ao cursor cima é igual ao relativo ao cursor esquerdo. A única diferença é que para saber as caraterísticas do rolamento para os cursores cima e baixo é usada a função `gtk_scrolled_window_get_vadjustment(scrolled_window)`, enquanto que para os cursores esquerda e direita é usada a função `gtk_scrolled_window_get_hadjustment(scrolled_window.`

Para finalizar, definiu-se que ao ser pressionada a tecla de saída se sai desta janela. Para tal usou-se a função `gtk_main_quit()`.

O fluxograma seguinte representa o procedimento para criar e operar o guia de programação.

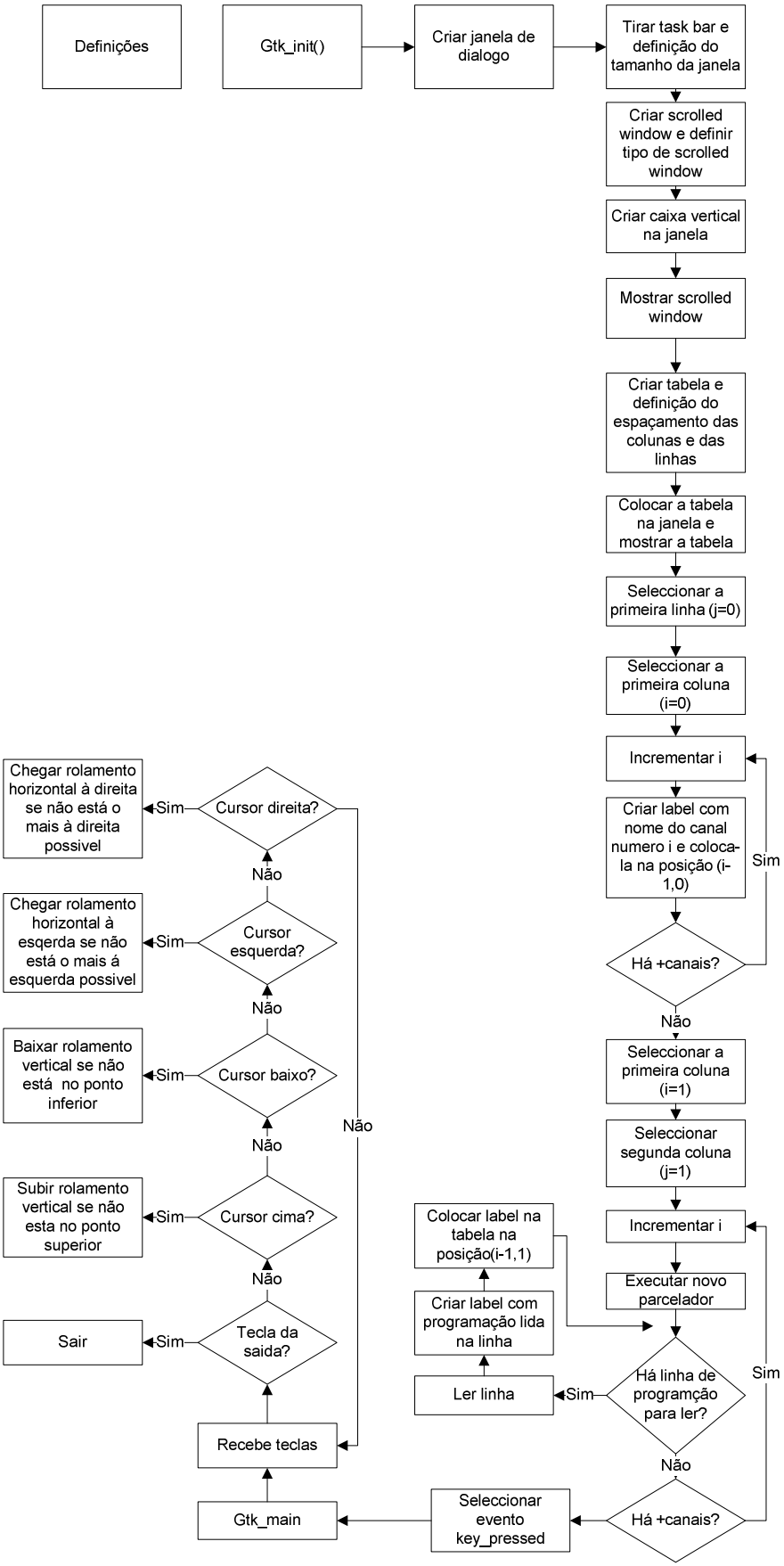


Figura 24 Fluxograma ilustrativo do mecanismo por detrás desta aplicação

7.2.7. JANELA DE GRAVAÇÃO

Para permitir ao utilizador a gravação de programas ou de períodos de transmissão de um determinado canal, esta janela deve permitir ao utilizador a introdução dos seguintes dados:

- Canal a gravar
- Dia de gravação
- Hora de inicio (hh:mm) de gravação
- Hora de fim (hh:mm) de gravação

Mais uma vez, foi criada uma janela e foi-lhe retirada a sua *task bar*. Criou-se então uma caixa vertical principal que se colocou na janela. Dentro desta caixa principal colocou-se uma *frame*, na qual se vão colocar os dados relativos ao dia a gravar.

```
frame = gtk_frame_new ("Dia");

gtk_box_pack_start (GTK_BOX (main_vbox), frame, TRUE, TRUE, 0);
```

O passo seguinte foi a criação de uma *label* e um botão de selecção para o dia, o mês e o ano. Ilustra-se agora como foi criada a *label* e a caixa de selecção apenas para o dia, uma vez que o processo se repete para o mês e o ano, mudando apenas o nome da *label* e as possibilidades de selecção no botão de selecção.

```
label = gtk_label_new ("Dia :");//criar label

gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);//escolhe posição da
label

gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);//colocar
label na caixa vertical

adj = (GtkAdjustment *) gtk_adjustment_new (01.0, 01.0, 31.0,
01.0,05.0, 0.0);//ajustar propriedades desejadas para o botão de
selecção. Neste caso números de 1 até 31, para o mês de 1 a 12, e para
o ano, do ano actual até 2012

elementos->spinner5 = gtk_spin_button_new (adj, 0, 0);//criar botão de
selecção com as propriedades acima definidas.

gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (elementos->spinner5),
TRUE);//para permitir saltar do valor superior para o valor inferior

gtk_box_pack_start (GTK_BOX (vbox2), elementos->spinner5, FALSE, TRUE,
0);//adicionar à caixa criada.

  g_signal_connect ((gpointer) elementos->spinner5,
"key_press_event",G_CALLBACK (on_spinner5_key_press_event),(gpointer)
elementos);//seleccionar evento tecla pressionada relativo a este botão
de selecção
```

Uma vez criado um bloco horizontal superior, relativo ao dia de gravação, procedeu-se à criação de um bloco horizontal central, relativo à hora de gravação. Para tal foi necessário repetir o processo usado para a criação do bloco horizontal superior, alterando apenas o seu conteúdo. Colocou-se, portanto, o conteúdo desejado dentro deste bloco horizontal central.

```
label = gtk_label_new ("Hora :");

adj = (GtkAdjustment *) gtk_adjustment_new (0.0, 0.0, 24.0,1.0, 1.0,
0.0);

elementos->spinner1 = gtk_spin_button_new (adj, 0.0, 0);

gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (elementos->spinner1), TRUE);

gtk_widget_set_size_request (elementos->spinner1, 100, -1);

gtk_box_pack_start (GTK_BOX (vbox2), elementos->spinner1, FALSE,
TRUE,0);

g_signal_connect ((gpointer) elementos->spinner1,
"key_press_event",G_CALLBACK (on_spinner1_key_press_event),(gpointer)
elementos);
```

Como se pode ver, as únicas alterações são o nome da *label* criada e os valores de configuração do botão de selecção. Também faz parte deste bloco central um botão de selecção relativo aos minutos de início de gravação. Por essa razão, o código anterior foi mais uma vez repetido, mudando apenas o nome da *label* (Minutos) e as propriedades relativas ao botão de selecção (de 0 a 59).

Seguidamente, gerou-se do bloco horizontal inferior, cujo processo de criação é igual ao do bloco central, excepto no nome da *frame* (Hora Fim de Gravação).

Para finalizar o processo de criação do interface, foi adicionado um botão ao canto inferior direito para confirmar o pedido e um botão de selecção do número do canal a gravar. A criação do botão de selecção respectivo ao canal foi igual à criação dos outros botões de selecção (criar caixa vertical e adicioná-la à respectiva caixa horizontal, editar botão e criar botão com essas propriedades).

```
elementos->button = gtk_button_new_with_label ("Gravar");

g_signal_connect ((gpointer) elementos->button, "clicked",G_CALLBACK
(trata_data),(gpointer) elementos);

gtk_box_pack_start (GTK_BOX (hbox), elementos->button, TRUE, TRUE, 5);

g_signal_connect ((gpointer) elementos->button,
"key_press_event",G_CALLBACK (on_button_key_press_event),(gpointer)
elementos);
```

Para tornar a janela visível e esperar pela interacção do utilizador, entrou-se na *main* do GTK (`gtk_main()`).

Como se verficou, foi seleccionado o evento de tecla pressionada em todos os botões de selecção e no botão de gravação. Tal deveu-se ao facto dos botões de selecção não reagirem ao pressionar de teclas por parte do utilizador. Assim, nos eventos relativos a cada botão de selecção, foi necessário verificar se o utilizador premiu o cursor cima, o cursor baixo, o cursor direito ou o cursor esquerdo. Se um botão de selecção tem o *focus* e o utilizador pressionar o cursor direito, o *focus* deve passar para o botão de selecção (ou o botão de gravação) seguinte; se pressionou o cursor esquerdo, o *focus* deve passar para o botão de selecção (ou o botão de gravação) anterior; se pressionou o cursor cima, deve incrementar o valor actual do botão que tem o *focus* e se pressionou o cursor baixo, deve decrementar o valor actual do botão que tem o *focus*.

```
val=gtk_spin_button_get_value(elem->spinner3);//Saber valor actual do
botão de selecção.

if(event->keyval == GDK_Right)

          {

          gtk_widget_grab_focus((gpointer)elem->spinner4);

          }

…

if(event->keyval == GDK_Down)

          {

          val--;

          gtk_spin_button_set_value(elem->spinner3,val);

          }

}
```

Caso o *focus* esteja no botão de gravação e o utilizador pressione o cursor esquerdo, o *focus* deve ser passado para o botão selecção relativo aos minutos da hora de fim de gravação. Se pressionar o cursor direito, o *focus* deve ser passado para o botão de selecção relativo ao dia de gravação. Ao ser pressionada a tecla de confirmação de gravação, os dados relativos à gravação são enviados para o cliente TCP/IP usando o FIFO (será explicado mais à frente).

Convém explicar algo que ainda não foi explicado. Para ser possível fazer a mudança de *focus*, as funções que tratam os eventos necessitam de receber, não só o *widget* relativo àquela função, mas todos os *widgets,* usando para esse efeito uma estrutura. Portanto foi necessário criar uma estrutura que contivesse todos os *widgets* relativos aos botões de selecção e ao botão de gravação.

O fluxograma seguinte demonstra como é criado e como opera a aplicação anteriormente descrita.

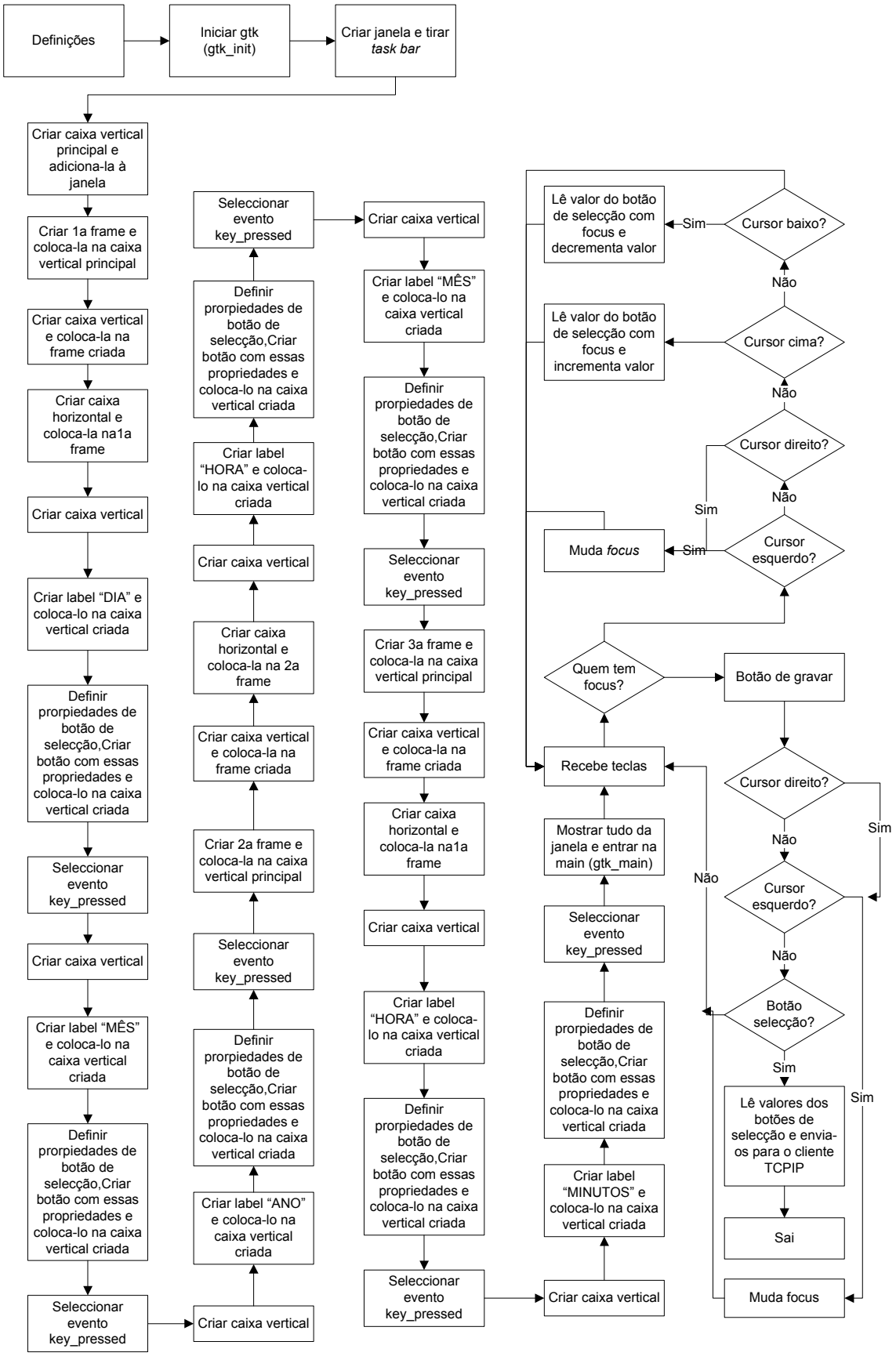


Figura 25 Fluxograma com o raciocínio usado no desenvolvimento desta aplicação

7.2.8. ACEDER A FICHEIROS GRAVADOS

De forma a permitir ao utilizador o acesso aos ficheiros que gravou, desenvolveu-se um interface gráfico em GTK que permite ao utilizador, seleccionar o vídeo que deseja ver entre os vários vídeos gravados.

Para desenvolver este interface, iniciou-se o GTK, foi criada uma janela, a sua posição foi definida e retirou-se a sua *task bar*.

Com o objectivo de ser possível a selecção do vídeo desejado por parte do utilizador, optou-se pelo uso de botões com o nome igual ao nome do vídeo. Para permitir tal opção, concebeu-se uma *frame* com o nome "vídeos disponíveis" e uma caixa vertical inserida nesta, para permitir introduzir os botões desejados na *frame*.

```
frame = gtk_frame_new ("Videos Disponiveis:");//criar frame

gtk_box_pack_start (GTK_BOX (main_vbox), frame, TRUE, TRUE,
0);//adicionar a frame à caixa vertical

vbox = gtk_vbox_new (FALSE, 0);//criar caixa vertical

gtk_container_add (GTK_CONTAINER (frame), vbox);//adicionar caixa
vertical à frame
```

Uma vez criada a *frame* que conterá os botões relativos aos vídeos, passou a ser necessário criar os botões com o nome do vídeo. Para obter o nome dos vídeos disponíveis, optou-se por usar um ficheiro de texto que contém o nome de todos os vídeos e que é obtido pelo Cliente TCP/IP através de um pedido ao servidor TCP/IP.

```
f=fopen("videos.txt","r");

while(fgets(linha,81,f))
      {
      linha[datasize-1]=0;

      printf("%s",linha);


      …


      }
```

Uma vez que o ficheiro tem um nome de vídeo por linha, usou-se o ciclo mostrado anteriormente para ler o nome de um vídeo de cada vez. Seguidamente, para cada nome lido, foi criada uma caixa vertical, um botão com o nome do vídeo e foi seleccionado o evento de botão pressionado.

```
g_signal_connect_swapped (G_OBJECT (button), "clicked", G_CALLBACK
(on_key_pressed), (gpointer)button);//seleccionar evento de botão
pressionado

button = gtk_toggle_button_new_with_label (linha);//criar label com o
nome lido no ficheiro

gtk_widget_set_usize(button,win_width/5-5,25);//definir tamanho do
botão

gtk_box_pack_start (GTK_BOX (vbox2), button, FALSE, TRUE,
0);//adicionar botão à caixa criada (no fundo à frame)
```

O passo seguinte foi a definição do sucedido quando um botão é pressionado, ou seja o que acontece quando há um evento. Ao ser lançado o evento, é lido o nome do botão pressionado e, posteriormente, enviado para o cliente TCP/IP usando o FIFO.

```
str=gtk_button_get_label(button); //para ler nome do botão
```

De seguida, foi criado um botão de saída da aplicação. Para tal, bastou criar uma caixa horizontal, uma caixa vertical dentro desta, um botão e, depois, seleccionou-se o evento de botão pressionado.

```
button = gtk_button_new_with_label ("Sair");

g_signal_connect_swapped (G_OBJECT (button), "clicked", G_CALLBACK
(on_key_pressed), (gpointer)button);
```

Para finalizar, foi adicionada uma comparação ao código do evento, que, em caso de ter sido pressionado o botão de saída, sai da aplicação.

```
str=gtk_button_get_label(button);

if(strcmp(str,"Sair")==0)

          {

                  gtk_main_quit ();

          }
```

Para se poder compreender os passos seguidos na criação desta aplicação pode ser consultado o fluxograma seguinte.

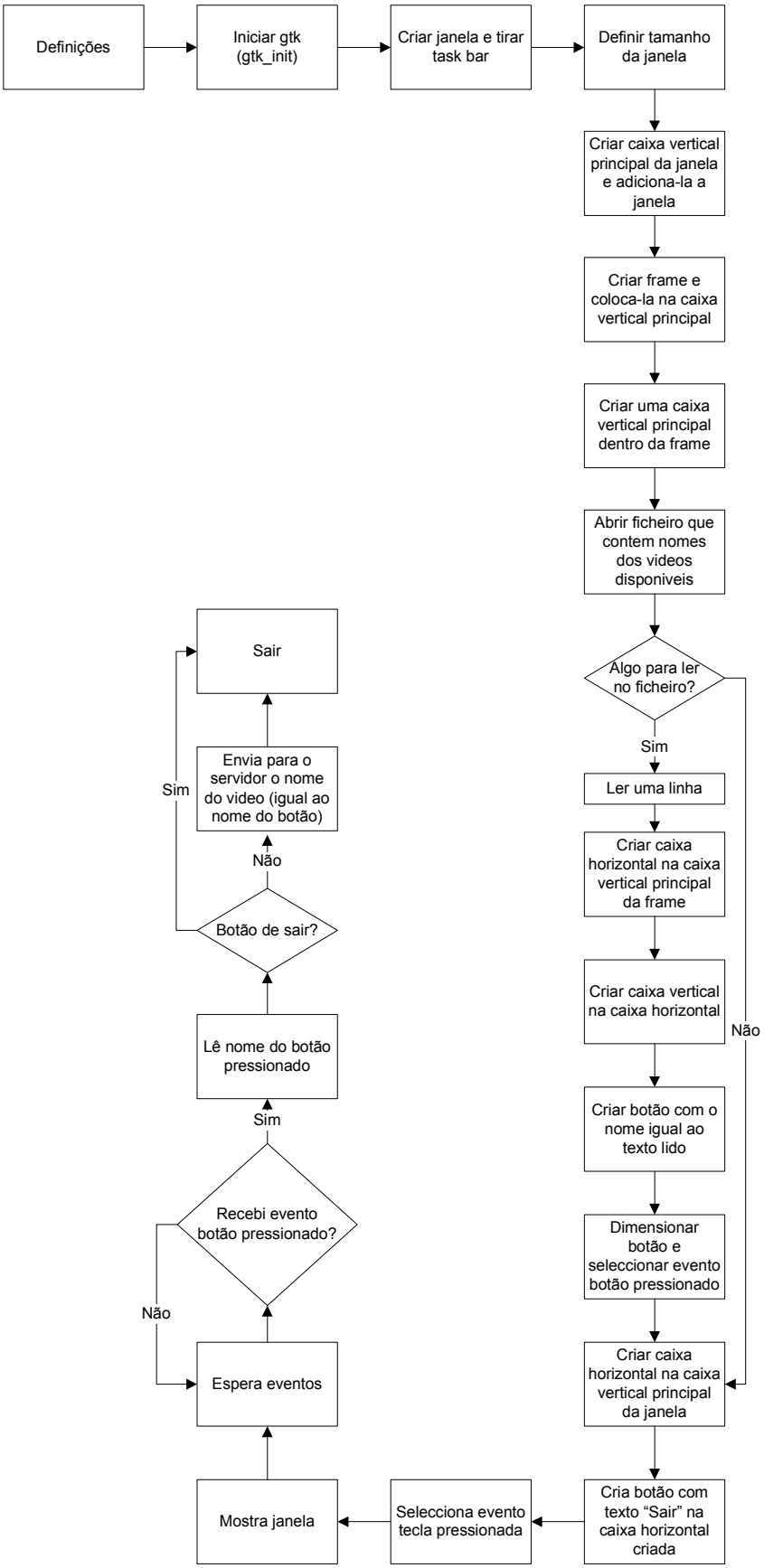


Figura 26 Fluxograma representativo da aplicação selecção de vídeos

7.2.9. VOD

Um serviço de VoD baseia-se num servidor HTTP que aloja o conteúdo a ser acedido. Assim, usou-se a instalação do servidor HTTP Apache descrita na secção4.5.1, mas em vez de se aceder ao este servidor com um *browser* acedeu-se usando o VLC.

Assim, como o servidor tem como endereço 172.16.12.1, para aceder ao vídeo "apanhados" alojado neste servidor, é usado o seguinte comando:

```
%vlc http://172.16.12.1:8080/apanhados.mpg
```

Para integrar o acesso a um serviço destes na solução, utilizaram-se os mesmos princípios do módulo de acesso a vídeos gravados descritos na secção7.2.8, ou seja, o cliente recebe um ficheiro de texto com o nome dos vídeos disponíveis no servidor VoD e cria um interface que permite seleccionar o vídeo desejado através do seu nome.

7.2.10. FIFO

Para haver uma comunicação entre o cliente TCP/IP e os outros módulos do cliente é usado um FIFO. Esta comunicação é fundamental pois permite o envio dos resultados da interacção do utilizador com a aplicação para o cliente TCP/IP.

Os FIFOs são *pipes* conhecidos (*named pipes*) e permitem colmatar os problemas dos *pipes* anónimos. Os FIFOs possuem um nome com existência no sistema, qualquer processo pode aceder-lhes e existem enquanto não forem completamente destruídos. No entanto, são na mesma *pipes*: só permitem um sentido de comunicação; tem de ser aberto no modo de escrita ou de leitura; a escrita, assim como a leitura, são bloqueadas se a *pipe* estiver vazia.

O mecanismo inerente ao uso do FIFO consiste em colocar o cliente TCP/IP a ler constantemente o FIFO (só lê quando o FIFO tem algo escrito) e as aplicações que querem enviar algo para o cliente TCP/IP a abrir o FIFO e a escrever neste, sempre que desejarem.

Assim, no cliente TCP/IP é usado o seguinte código dentro de um ciclo infinito (`while (!ciclo))`:

```
unlink("temp_fifo");//destrói FIFO temp_fifo se existir
mknod("temp_fifo",S_IFIFO|0660,0);//cria fifo
fd=open("temp_fifo",O_RDONLY);//abre o fifo em modo apenas de leitura
```

```
read(fd,&msg,sizeof(msg));//lê quando estiver alguma coisa escrita no
FIFO
```

O facto de se destruir o FIFO no início do ciclo, faz com que, quando é usado o comando `read ()`, este espere até que seja escrita alguma coisa no FIFO, pois este comando bloqueia enquanto o FIFO não contém nada escrito.

Para escrever no FIFO, as aplicações usam o seguinte código:

```
fd=open("temp_fifo",O_WRONLY);//abrir fifo em modo de escrita

write(fd,&msg,sizeof(msg));//escrever no fifo
```

Como se pode ver, primeiramente abre-se o FIFO em modo de escrita para depois se escrever neste.

Este FIFO é usado para passar uma estrutura, que contém os dados que se quer enviar para o servidor TCP/IP (o número de um canal, por exemplo) e um identificador do que se está a enviar.

```
struct my_msg{

int elem;// identificador do tipo de dados (numero de canal, nome de
vídeo contido no servidor, etc)

char frase[SIZE];//dados a enviar

};
```

7.2.11. CLIENTE TCP/IP

Este módulo corre em *background* no cliente, de forma a permitir ao cliente comunicar com o servidor. Como já foi referido, para comunicar com a aplicação é usado um FIFO.O cliente TCP/IP tem como função enviar para o servidor TCP/IP os pedidos efectuados pelo utilizador. Assim, este módulo deve enviar o número do canal pedido e, caso seja pedido um vídeo alojado no servidor, o nome desse vídeo. Para efectuar um pedido de gravação deverá indicar o número do canal a gravar e a hora de início e de fim de gravação.

No desenvolvimento deste módulo foram usadas as bases do cliente TCP/IP desenvolvidas nos testes preliminares (secção4.5.3). Uma vez que todo o código implicado neste módulo já foi explicado anteriormente, não será aqui explicado o código envolvido, apenas o raciocínio por detrás deste.

Assim, começa-se por criar o *socket*, editar a sua estrutura e conectar-se. De forma a passar para o servidor TCP/IP os dados desejados, foi criado um ciclo infinito que permite estar constantemente a receber dados via FIFO e enviá-los para o servidor TCP/IP.

No entanto, depois de receber os dados via FIFO, mas antes de os enviar para o servidor TCP/IP, é lido o identificador do tipo de dados da estrutura recebida no FIFO para saber se deve sair e parar de operar.

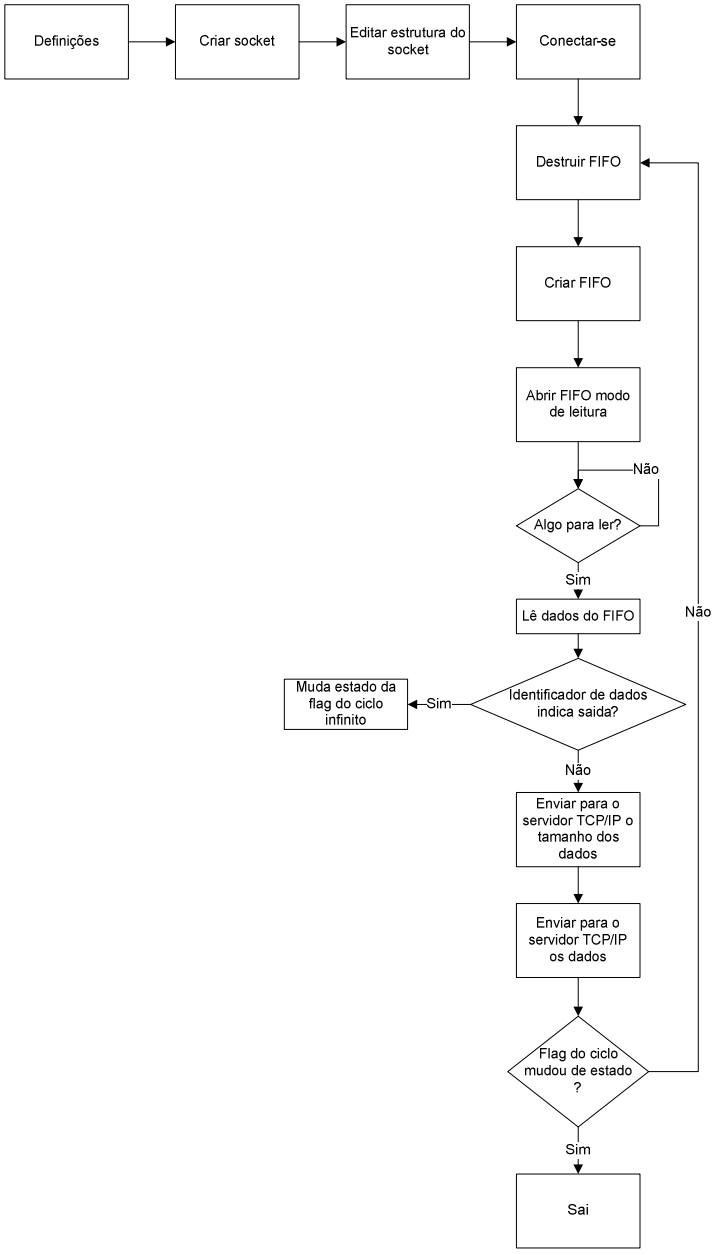


Figura 27 Fluxograma representativo do raciocínio por detrás do cliente TCP/IP

8. APRESENTAÇÃO DA SOLUÇÃO FINAL

Uma vez conhecidos os módulos de *software* existentes nesta solução bem como o modo como foram desenvolvidos estes módulos, de seguida apresenta-se o interface gráfico desenvolvido.

Ao ser lançada a aplicação é mostrada a janela principal que se encontra a reproduzir o vídeo que está a ser enviado pelo servidor.

Figura 28 Janela principal

Uma vez nesta janela, o utilizador tem todas as opções disponíveis bastando-lhe premir a tecla desejada para aceder às opções.

Se premiu a tecla que permite o corte do som, este é desligado e a seguinte janela é acrescentada no ecrã:

Figura 29 Janela com indicação de corte de som

Uma vez com o som desligado, se o utilizador voltar a premir a mesma tecla, o som é reposto e isso é indicado no ecrã durante um (1) segundo.

Figura 30 Janela com indicação de reactivação do som

Se o utilizador premiu uma tecla de mudança do nível de volume, o novo nível de volume é indicado no ecrã.

Figura 31 Janela com indicação do novo nível de volume

Outra das funcionalidades disponíveis é a mudança do tipo de aspecto do ecrã. Entre as opções disponíveis encontra-se 16:9 e 4:3. Assim, ao ser premido o botão que opera essa mudança, é feita uma alteração do estado actual para outro estado. Assim, obtêm-se, entre outros, os seguintes aspectos:

Figura 32 Aspecto 4:3

Figura 33 Aspecto 16:9

Caso o utilizador efectue uma mudança de canal, quer premindo os botões de aumento ou diminuição do número de canal, quer seleccionando o canal pelo seu número, é mostrada a seguinte janela:

Figura 34 Janela de selecção do canal 4 quando se vê canal 7

De salientar que no exemplo acima, o utilizador está a ver o programa Bloomberg Live no canal número 7 e deseja mudar para o canal número 4 (o conteúdo que está a ser reproduzido não corresponde ao canal Bloomberg, tratando-se apenas de uma simulação). Convém referir que, uma vez nesta janela que contém o canal a ser seleccionado, é possível seleccionar um canal na casa dos quarenta, se tal for feito no intervalo de um (1) segundo. Uma vez seleccionado um canal de 40 a 49, o utilizador tem mais um (1) segundo para escolher outro canal caso se tiver enganado no número. Este raciocínio já foi explicado anteriormente

Outra das opções disponíveis quando o utilizador se encontra na janela principal é a visualização de um guia de programação. Assim, se o utilizador premir a tecla relativa à programação, visualiza a seguinte janela:

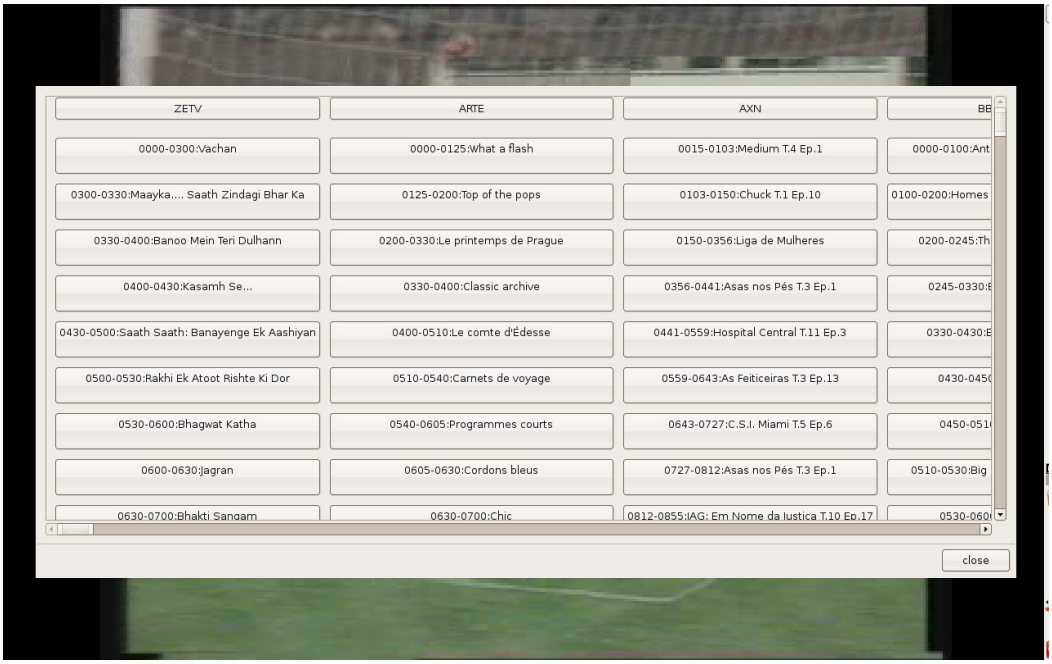


Figura 35 Guia de programação

Uma vez a visualizar o guia de programação, o utilizador pode viajar ao longo do guia usando as teclas direccionais e sair do guia ao premir a tecla de saída.

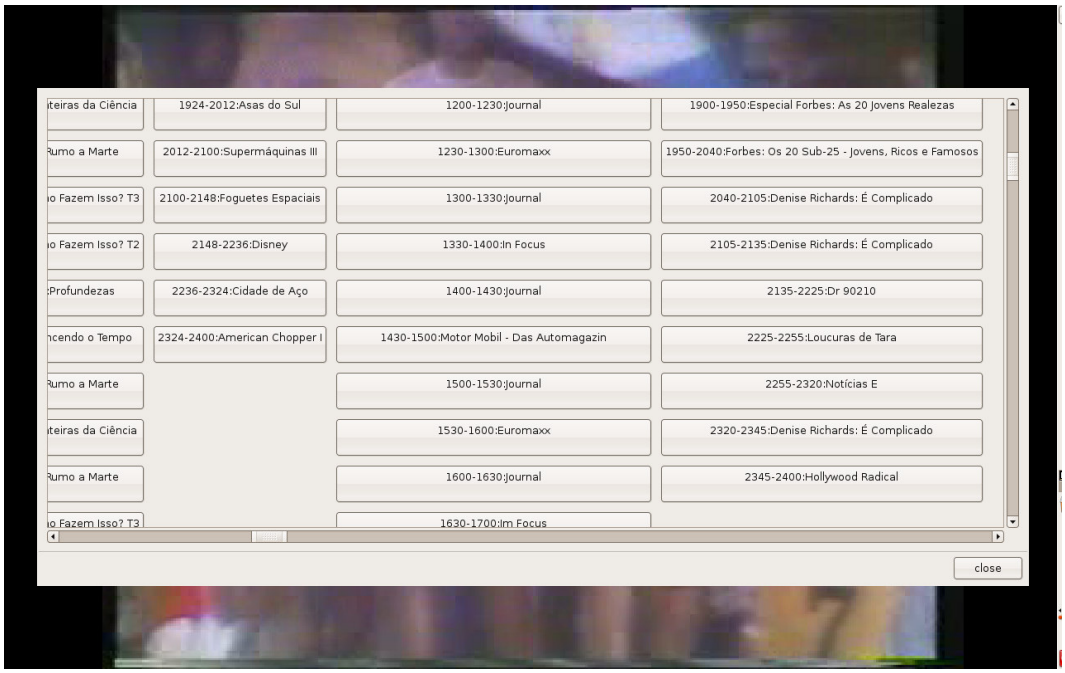


Figura 36 Guia de programação após avanços horizontais e verticais

Caso o utilizador deseje gravar um canal em determinado dia e entre certo intervalo, basta premir a tecla de gravação e obtém o seguinte interface.

Figura 37 Janela de gravação

Na janela apresentada anteriormente, o utilizador deve usar os cursores esquerdo e direito para mudar de campo de selecção, os cursores cima e baixo para mudar o valor do campo que tem *focus*, a tecla de selecção para colocar a gravar quando o focus está no botão "Gravar" e a tecla de saída para sair desta janela sem gravar.

Caso o utilizador pretenda ver um vídeo gravado, alojado no servidor, deve premir a tecla de visualização de vídeos gravados.

Figura 38 Menu de selecção de vídeo gravado

Neste menu o utilizador pode usar os cursores para colocar o *focus* no botão correspondente ao vídeo que deseja visualizar e premir a tecla de selecção para escolher o vídeo correspondente ao botão que tem o *focus*. Para sair deste menu o utilizador pode premir a tecla de saída ou premir a tecla de selecção quando o botão "Sair" tem o *focus*.

De salientar que existe ainda a possibilidade de pausa em tempo real.

9. CONCLUSÕES

De modo a possibilitar o desenvolvimento desta Tese, foi realizado um estudo sobre os serviços e a arquitectura das soluções de IPTV. Este estudo permitiu conhecer a estrutura de uma rede IPTV, bem como os seus protocolos, as soluções existentes e as suas limitações. Assim, propôs-se que, para contornar as limitações das soluções existentes e, para se poderem introduzir novos serviços, seria necessário incluir na casa dos utilizadores um servidor que distribuisse o serviço para várias televisões. Com o intuito de demonstrar a viabilidade desta proposta, desenvolveu-se um protótipo funcional.

Devido às suas características, o protótipo desenvolvido consegue ultrapassar a maior limitação imposta pelas soluções existentes, o número de *boxes*. Tal foi possível devido ao uso de um modelo servidor-cliente, para colocar o servidor local a comunicar com as *boxes*. Assim, o servidor aceita pedidos de ligação das *boxes* e trata-os individualmente. Caso já tenha sido atingido o limite de canais que estão a ser vistos em simultâneo nas várias *boxes* de uma casa e uma nova *box* se ligue ao servidor, o utilizador desta *box* apenas poderá visualizar os mesmos canais que os utilizadores das outras *boxes* ou aceder a outros serviços disponibilizados, tais como programas gravados ou vídeos alojados no servidor.

Para se aproximar o protótipo a uma solução real, foram desenvolvidas aplicações que permitem visualizar a programação actual de um canal, consultar o guia de programação, VoD, gravação de canais, visionamento de conteúdos gravados e pausa em tempo real. Foi ainda testada, satisfatoriamente, a reprodução de vídeos em HD. Estas aplicações desenvolvidas foram realizadas recorrendo apenas a *software* livre, o que permitiu excluir custos de *software* do protótipo desenvolvido.

No que diz respeito aos custos de *hardware*, embora à primeira vista pareça que a inclusão de um servidor na casa de cada cliente encarece a solução proposta, o facto das *boxes* não necessitarem de disco e do servidor não precisar de ter nem placa gráfica nem uma grande capacidade de processamento, compensa a adição do servidor local. Ou seja, para um cliente de um serviço destes que deseje ter três ou quatro *boxes*, o valor que é poupado em cada *box* compensa o custo do servidor local.

Pode-se então concluir que, no geral, esta solução preenche os requisitos necessários, bem como os objectivos propostos, tendo no entanto algumas limitações. Embora este protótipo tenha sido testado usando conteúdos alojados localmente, seria interessante testar e validar a solução tendo acesso a um serviço real de IPTV. Nesse caso, haveria a hipótese de testar, entre outros, as velocidades de acesso deste protótipo a cada canal. Mesmo assim, com os testes efectuados, foi possível verificar alguma consistência e uma boa resposta por parte do protótipo desenvolvido.

9.1. DESENVOLVIMENTOS FUTUROS

Uma vez que o sistema implementado se trata de um protótipo, existe sempre a possibilidade de melhorar esta aplicação de modo a se ter uma solução real de TV digital. Assim, para melhorar o protótipo apresentado, este deve ser implementado num *hardware*, de baixo custo, adequado para a função que vai desempenhar.

Com o intuito de melhorar a solução apresentada e para se poderem realizar outro tipo de testes, o servidor local deve conseguir receber canais de IPTV de um fornecedor

Devido à inclusão do servidor local, é possível acrescentar ao sistema proposto algumas funcionalidades e serviços que não estão disponíveis nas soluções comerciais.

Para além dos canais disponibilizados pelo fornecedor do serviço de IPTV, o servidor local pode oferecer aos utilizadores canais de *Internet* TV, bastando para tal, o servidor juntar-se a um desses canais e colocar o VLC a distribuí-lo para os utilizadores que o desejam ver.

O facto das *boxes* terem a possibilidade de aceder à *Internet*, pode ser facilmente acrescentado um *browser* que permita ao utilizador navegar na *Internet* e consultar o seu *e-mail*.

Outra funcionalidade passível de ser acrescentada a este protótipo, é permitir que o utilizador visualize o que um sistema de vigilância está a gravar. Tal é possível se o sistema de vigilância se encontrar ligado ao servidor local ou se este sistema estiver disponível a partir de um endereço IP.

Pode também ser implementado um serviço de telefone que permita o envio e a recepção de SMS, bastando acrescentar ao *hardware* do servidor um módulo GSM.

Para tirar ainda maior partido da inclusão do servidor local na solução, este pode ser aperfeiçoado, de forma a oferecer ao utilizador mais opções e poupar recursos. Um dos melhoramentos que pode ser realizado, é colocar o servidor a calcular qual o canal mais visto pelos utilizadores e, devido à sua grande capacidade de armazenamento, guardar em disco todos os programas que foram emitidos por esse canal nas últimas 24 horas. Assim, um utilizador ao ver um programa gravado anteriormente, não está a ver um canal em tempo real, poupando-se essa largura de banda para outro utilizador. No fundo, muitos progressos podem ser feitos no servidor local, pois quanto mais "inteligente" este for, mais opções têm os utilizadores.

Numa fase mais avançada, pode-se partir para a criação de um protótipo de um fornecedor de um serviço de IPTV. Este deve oferecer os serviços normalmente disponibilizados pelas operadoras: canais de TV, VoD, *pay per view*. No entanto, para além destes serviços já existentes no mercado português, existem outros que podem ser implementados.

O facto de uma rede de IPTV permitir uma comunicação bidireccional, outra funcionalidade passível de ser implementada, é a TV interactiva, que permite a introdução de *chats* em cada programa, votações e selecção do ângulo da câmara desejado numa transmissão.

Nos Estados Unidos, as soluções necessitam de ter um módulo que permita avisar o público em situações de emergência, o *emergency alert system*. Este sistema é usado, por exemplo, quando o presidente tem uma comunicação importante para fazer ao país. Uma funcionalidade destas pode ser implementada noutro tipo de emergências, tais como alertas de mau tempo.

Referências Documentais

[1] WWW.NETCRAFT.COM

[2] STALLINGS,William—*Data and Computer Communications*.

[3] TANEMBAUM,Andrew S. —*Computer Networks*. Prentice Hall, Inc.

[4] HELD, Gilbert—*Understanding IPTV*.

[5] Juniper Networks —Introduction to IGMP for IPTV Networks, Understanding IGMP Processing in the Broadband Access Network

[6] http://www.marketingcharts.com/television/iptv-subscribers-to-number-60-million-by-2011-484/

[7] http://www.abiresearch.com/eblasts/archives/analystinsider_template.jsp?id=106

[8] http://www.telecom.pt/InternetResource/PTSite/PT/Canais/Media/DestaquesHP/Meo 100mil.htm

[9] http://acesso.clix.pt/televisao/index.html

[10] http://imgs.sapo.pt/files/meo_v2/pdf/AF_Guia_Utilizador.pdf

[11] http://loja.ptcom.pt/NR/rdonlyres/7BC5BBB2-9EA4-42D8-AE8B-0093E04AB922/0/GuiaMEO_110707.pdf

[12] http://tldp.org/HOWTO/VideoLAN-HOWTO/

[13] http://www.videolan.org/doc/streaming-howto/en/streaming-howto-en.html

[14] http://en.wikipedia.org/wiki/X_Window_System

[15] http://www.x.org/wiki

[16] http://lirc.sourceforge.net/remotes/atiusb/lircd.conf.atiusb

[17] http://www.lirc.org/

[18] http://www.xmltv.org/wiki/

[19] http://www.adobe.com/devnet/flashplayer/articles/hd_video_flash_player_print.html

[20] http://www.zon.pt/zonbox/

[21] http://www.ebu.ch/en/technical/trev/trev_300-wood.pdf

[22] http://www.ebu.ch/en/technical/trev/trev_299-ive.pdf

[23] http://www.mediacollege.com/downloads/video/hd/

[24] http://www.freebsd.org/cgi/man.cgi?query=crontab&sektion=5

[25] TRONCHE, Christphe— Xlib --- C *Language X Interface.*

[26] O'REILLY & ASSOCIATES, INC—Xlib Programming Manual.

[27] http://users.actcom.co.il/~choo/lupg/tutorials/xlib-programming/xlib-programming.html

[28] http://zetcode.com/tutorials/gtktutorial/

[29] http://www.gtk.org/documentation.html

[30] http://library.gnome.org/devel/gtk-tutorial/stable/

[31] http://www.yolinux.com/TUTORIALS/GTK+ProgrammingTips.html

[32] FLECK, John—Libxml Tutorial.

[33] LATTRE, Alexis; DAOUD,Anil; PRACHT,Benjamin; STENAC,Clémant; SAMAN.Jean-Paul — VLC Play Howto

[34] http://www.anacise.com/pdf/IPTV%20Network.pdf

[35] http://www.apricot.net/apricot2006/slides/conf/wednesday/Sachin_Natu-Fast%20Reroute%20for%20Triple%20Play.pdf

[36] http://www.broadbandservicesforum.org/images/Pages/IPTV%20Explained.pdf

[37] http://www.apricot.net/apricot2006/slides/conf/wednesday/Sachin_Natu-Fast%20Reroute%20for%20Triple%20Play.pdf

[38] http://www.iptvmagazine.com/2005_10/iptvmagazine_2005_10_new_iptv_requirements.htm

[39] Shenick Network Systems—Test Performance of IGMP based Multicast Services with emulated IPTV STBs

[40] ROSE, Marshall T. —*The Simple Book: An Introduction to Networking Management, Readings in Simple Network Management Protocol*. Prentice Hall, Inc, 1996. ISBN 0-13-451659-1.

[41] SCHÖNWÄLDER, Jürgen—*Internet Management Technologies*. International University Bremen, Germany, http://www.faculty.iu-bremen.de/schoenw

[42] SIMPSON,Wes;GREENFIELD Howard — *IPTV and Internet Vídeo*.Focal Press

[43] http://www.dolby.com/index.

Anexo A. VideoLan

O *VideoLan* é um projecto desenvolvido por estudantes da *Ecole Centrale Paris* e por programadores de todo mundo, sob a GNU *General Public License* (GPL). Este projecto permite fazer *streaming* de vídeos MPEG em redes com grande largura de banda. O VideoLan inclui dois serviços: o *VideoLanServer* (VLS) e o *VideoLanClient* (VLC). Apresenta-se de seguida a solução usada no site deste projecto para ilustrar as suas capacidades de *streaming*.

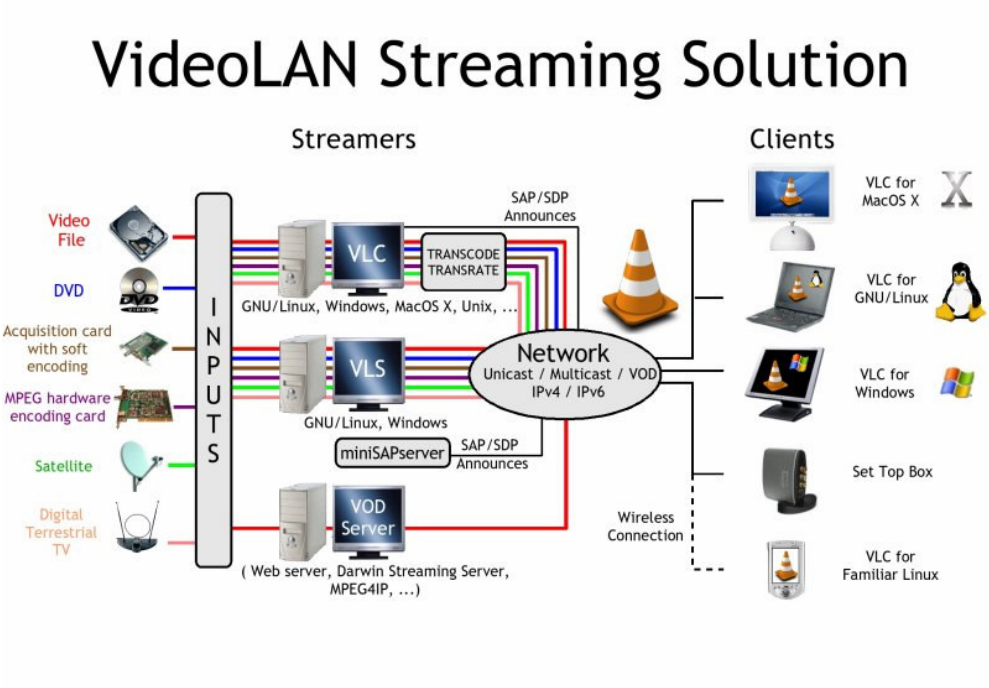


Figura 39 Solução para VideoLan *streaming*[13]

As capacidades de cada um dos componentes deste projecto são agora ilustradas

O VLC tem capacidade de:

- Ler ficheiros MPEG-1, MPEG-2 e MPEG-4 / DivX provenientes de um disco, um CD-ROM, DVDs e VCDs;
- Ler a partir de um cartão de satélite (DVB-S);

- Ler *streams* MPEG-1, MPEG-2 e MPEG-4 provenientes de uma rede, que podem ser enviadas usando o VLS ou o VLC;
- Ser usado como servidor para fazer *stream* de ficheiros MPEG-1, MPEG-2 e MPEG-4 / DivX, de DVDs, ou a partir de um *encoding card* MPEG. Este *streaming* pode ser feito usando *unicast* ou *multicast.*

O VLS consegue:

- Fazer *streaming* a partir de ficheiros MPEG-1, MPEG-2, MPEG-4, localizados num disco, CD, ou DVD;
- Fazer *streaming* a partir de um cartão de satélite (DVB-S) ou de um cartão de televisão digital terrestre (DVB-T);
- Fazer *streaming* a partir de um *encoding card* MPEG (tal como o VLC, este *streaming* pode ser feito usando *unicast* ou usando *multicast*);

- **Operar o VLC usando a linha de comandos**

Um dos maiores atributos do VLC é a capacidade de poder ser operado, quer usando linha de comandos, quer através do seu interface gráfico. Para a solução apresentada nesta Tese, o VLC é apenas operado através do uso da linha de comandos. Por essa razão, passa-se a apresenta-se seguidamente os comandos necessários para conseguir realizar as principais funções deste programa.

Receber uma *stream unicast*

```
% vlc –vvv udp:
```

Receber uma *stream multicast*

```
% vlc –vvv udp:@239.255.12.42
```

onde 239.255.12.42 é o endereço *multicast* ao qual nos queremos juntar.

Receber uma *stream* HTTP/FTP/MMS

Para uma *stream* HTTP:

```
% vlc –vvv http://example/stream.xtz

ou

% vlc –vvv http://172.16.12.1
```

Para uma *stream* FTP:

```
% vlc –vvv ftp://example/stream.xyz
```

Para uma *stream* MMS:

```
%vlc –vvv mms://viptvr.yacast.fr/encoderfranceinfo
```

Gravar uma *stream* num disco

Uma das propriedades importantes deste programa é a capacidade de gravação de uma *stream* num disco. Esta funcionalidade é usada na solução apresentada, quando se deseja gravar um determinado programa. Para tal, é usada a *stream output* do VLC, sendo também necessário especificar o formato do *muxer* a usar (AVI, OGG, MPEG-2-PS,MPEG-2-TS) e o nome do ficheiro para o qual se pretende gravar, com a extensão adequada. Assim, aos comandos apresentados anteriormente, acrescenta-se:

```
–-sout file/muxer:nomeficheiro.xyz
```

Fazer *streaming* de um ficheiro

Quando um utilizador deseja ver um vídeo anteriormente gravado, é necessário enviá-lo. Para tal usa-se o nome do vídeo (neste caso RTP_21_12_08.mpeg), o IP da máquina para a qual se pretende enviar (neste caso 172.16.12.1) e o TTL dos pacotes IP:

```
% vlc –vvv RTP_21_12_08.mpeg –-sout udp:172.16.12.1 –ttl 12
```

Fazer *streaming* de um DVD

```
% vlc –vvv devdold:/dev/dvd –sout udp:172.16.12.1 –ttl 12
```

onde `/dev/dvd` é o nome da *drive* do DVD e 172.16.12.1 é o endereço IP para o qual se quer fazer *unicast* .

Fazer *streaming* de *driver* de vídeo/*Transcoding*

Essencialmente no início dos testes, para a obtenção da solução aqui apresentada, foi utilizada uma *webcam* para simular um canal de TV reproduzido em tempo real. Foi portanto necessário inserir no *kernel* um módulo para esta *webcam* ser reconhecida (gspca.ko). Para além de ser possível reproduzir estes dispositivos com o VLC, é também permissível fazer *streaming* e gravar o que está a ser reproduzido em disco. Para fazer *streaming* é necessário transcodificar o conteúdo:

```
% vlc  -vvv
v4l:/dev/video0:norm=secam:frequency=543250:size=640x480:channel=0:adev
=/dev/dsp:audio=0–
sout'#transcode{vcodec=mp4vacodec=mpga,vb=3000,ab=256,vt800000,keyint=8
0,deinterlance}:std{Access=udp,mux=ts,url=239.255.12.13}' ttl 12
```

Sendo:

- /dev/video0 a webcam
- *norm=secam* o nome do sinal analógico *strandard*
- *frequency*=543250 a frequência do canal em kHz
- *size*=640x480 o tamanho/resolução do vídeo
- *channel*=0 o número do canal (*tuner*, *composite* ou *svideo*)
- *adev=/dev/dsp* o dispositivo áudio
- *audio=1* o numero do canal áudio (*mono* ou *stereo*)
- *vcodec=mp4v* o formato de vídeo desejado (MPEG-4,MPEG-1, h263, *DIV1*, *DIV2*, *DIV3*, I420, I422, I444, RV24, YUY2)
- *acodec=mpga* o formato áudio (MPEG,A52)
- *vb=3000* a *bitrate* de vídeo em Kbit/s
- *ab=256* é a *bitrate* de áudio em Kbit/s
- *vt=800000* a tolerância da *bitrate* vídeo em bit/s
- 192.168.0.42 o endereço IP da máquina para a qual se quer fazer *unicast*
- 12 o valor do TTL

[12][13][33]

- **LIBVLC**

Com o intuito de embeber o VLC nesta solução, é usada uma API externa do VLC, que consiste num ficheiro escrito em linguagem C, com o intuito de permitir o uso de algumas funções e propriedades do VLC num programa C. Deste modo, através da inclusão do ficheiro libvlc.h é possível aceder às funções do VLC, com as quais se consegue criar uma janela que reproduz um conteúdo seleccionado. Seguidamente destacam-se algumas das principais funções da API.

Inicialização da estrutura de uma excepção

```
Void libvlc exception init (libvlc exception t * p_exception);
```

Onde *p_exception* é a excepção a iniciar

Exemplo:

```
Libvlc_exception_t excp;

Libvlc_exception_init (&excp) ;
```

Criar e iniciar uma instância Libvlc

```
Libvlc_instance_t* libvlc_new (int, const char * const *,
libvlc_exception_t*);
```

onde é usado o apontador da excepção iniciada.

Exemplo:

```
libvlc_instance_t *inst;

inst = libvlc_new (argc,argv, & excp) ;
```

Adicionar item a uma *playlist*

Para a reprodução de um conteúdo, usando esta API, é necessário adicionar à *playlist,* da instância criada, o item a reproduzir.

```
int libvlc_playlist_add(libvlc_instance_t *,const char *,const char
*,libvlc_exception_t *) ;
```

É utilizada a instância e a excepção iniciada, o nome do ficheiro a reproduzir (localização no disco incluída) e o nome desejado na *playlist*. Esta função retorna o identificador do item adicionado (item). De salientar que esta função adiciona o item ao fim da *playlist*.

Exemplo:

```
char * filename = “udp://”;
```

Ou

```
char * filename = /home/lugar/do/video/video.mpg”;

int item;

item=libvlc_playlist_add (inst,filename,NULL, &excp);
```

Reproduzir uma *playlist*

Para reproduzir a *playlist* usa-se a função:

```
void libvlc_playlist_play (libvlc_instance_t*,int,int,char **,
libvlc_exception_t *);
```

Exemplo:

```
libvlc_playlist_play (inst, item, 0, NULL, & excp);
```

É usada a instância e a excepção criada, o identificador do item a reproduzir (item), o número de opções introduzidas (neste caso 0) e a opção escolhida (nenhuma, neste caso).

Obter a entrada que está a ser reproduzida

Para se aceder a algumas funções do VLC disponíveis nesta API, principalmente as relacionadas com as propriedades da janela criada, é necessário conhecer a entrada que está a ser reproduzida na *playlist*, sendo portanto usadas a instancia e a excepção criadas.

```
libvlc_input_t *libvlc_playlist_get_input (libvlc_intance_t *,
libvlc_exception_t *);
```

Exemplo :

```
libvlc_input_t * inp;

inp = libvlc_playlist_get_input (inst, &excp);
```

Colocar em *FullScreen*

Para colocar a janela criada em *fullscreen* é usada a entrada obtida, assim como a excepção criada, sendo também usada uma *flag* booleana para activar e desactivar o *fullscreen* (1 para activar *fullscreen* e 0 para desactivar).

```
void libvlc_set_fullscreen (libvlc_input_t *, int,
libvlc_exception_t*);
```

Exemplo:

```
livlc_set_fullscreen (inp,1,&excp);
```

Paragem durante reprodução

Para possibilitar paragem em tempo real da reprodução de um conteúdo, são usadas a excepção e a instância criadas. De salientar, que esta função opera uma mudança de estado, ou seja, se um conteúdo está a ser reproduzido e esta função é chamada, a reprodução é parada. Por outro lado, se a reprodução está parada e esta função é chamada, o conteúdo começa a ser reproduzido.

```
void libvlc_playlist_pause (libvlc_instance_t *, libvlc_exception_t *);
```

Exemplo:

```
libvlc_playlist_pause (inst,& excp);
```

Alterar o volume

Para alterar o volume de um conteúdo que está a ser reproduzido, é necessário conhecer o volume actual e modifica-lo. Nesta API, o volume é representado por um valor inteiro que indica a percentagem actual. Assim, para conhecer o volume actual:

```
int libvlc_audio_get_volume (libvlc_instance_t *, libvlc_exception_t
*);
```

Exemplo:

```
int volume;

vlc_bool_t mute;

volume = libvlc_audio_get_volume (inst, &excp);
```

Uma vez identificado o valor actual do volume, este pode ser incrementado ou decrementado:

```
void libvlc_audio_set_volume (libvlc_instance_t *, int,
libvlc_exception_t *);
```

Exemplo:

```
volume=volume+5; //permite incrementar 5% o volume actual

libvlc_audio_set_volume (inst,volume,&excp);
```

Cortar/colocar o Som

Para se poder retirar ou colocar o som de um conteúdo que está a ser reproduzido, é necessário usar uma função que permita mudar o estado actual para o estado oposto. Para tal:

```
void libvlc_audio_toogle_mute (libvlc_instance_t *, libvlc_exception_t
*);
```

Exemplo :

```
libvlc_audio_toogle_mute (inst, &excp);
```

Saber se a *playlist* ainda reproduz

Antes de se fechar uma janela criada, pode ser necessário identificar se a *playlist* ainda reproduz. Em caso positivo, retorna um (1); caso contrário retorna zero (0). Para tal, deve ser usada a seguinte função que faz uso da instância e da excepção criadas:

```
int libvlc_playlist_isplaying (libvlc_instance_t *, libvlcexception_t
*);
```

Exemplo:

```
int playing;

playing=libvlc_playlist_isplaying (inst, &excp);
```

Destruir a Janela

Para destruir a janela de reprodução do VLC, é inevitável destruir a instância criada. Assim sendo, apenas se usa a seguinte função:

```
void libvlc_destroy (libvlc_instance_t *);
```

Exemplo :

```
libvlc_destroy (inst) ;
```

Anexo B. Xlib

O sistema de janelas X (X *window system*), desenvolvido pelo MIT em 1984, opera com janelas e implementa o protocolo de *display* X.

O X segue o modelo servidor-cliente. Enquanto o servidor X, instalado numa máquina, permite a recepção de pedidos através de um porto de conexão, o cliente X tem de se conectar ao servidor através dessa porta e enviar pedidos através do uso do protocolo X que, por sua vez, faz uso da biblioteca X (Xlib).

O X providencia as primitivas necessárias para a construção de um GUI: mover janelas no ecrã e interagir com estas através de periféricos. Este protocolo é construído como uma camada de aplicação adicional do *kernel* de um sistema operativo.

De salientar que os clientes não necessitam de se encontrar na mesma máquina do que o servidor, isto é, o X possui transparência à rede: a máquina onde a aplicação "corre" (cliente) é diferente da máquina local do utilizador (servidor). No entanto, para esta comunicação ser segura, deve ser feito *tunneling* à conexão numa sessão encriptada (fazer passar por uma sessão encriptada). Para ser possível usar uma sessão remota, é normalmente usado o SSH ou telnet.

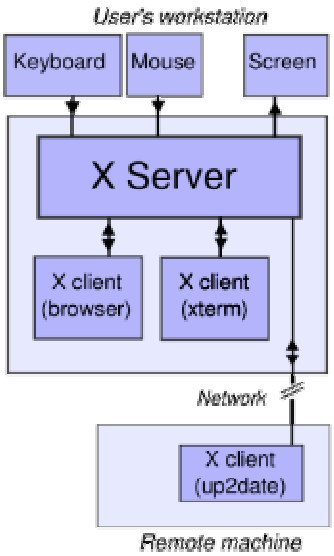


Figura 40 Exemplo de ligação servidor-cliente [14]

Uma variável importante na operação com o X é o *display*. Num ambiente Unix, o valor desta variável pode ser consultado usando o seguinte comando: `echo $DISPLAY`. No caso de se abrir uma sessão remota, esta variável necessita de ser exportada.

Assim, a Xlib é uma biblioteca escrita em linguagem C, que serve de interface aos clientes para estes se ligarem ao servidor.[15]

- **X *Window System***

Para a mesma máquina, podem existir vários ecrãs, que não são mais do que um monitor e um *hardware* colorido em escala de cinza ou monocromático. Um *display* é um conjunto de ecrãs disponíveis para um único utilizador, um rato e um teclado. Um servidor X pode providenciar vários ecrãs.

As janelas num servidor X estão organizadas por hierarquias, sendo que a janela do topo dessa hierarquia é denominada janela raiz. Tal como acontece com os processos nos sistemas Unix, todas as janelas têm um pai excepto a janela raiz; cada janela pode conter uma ou mais janelas filho. Cada janela tem um sistema de coordenadas, para ser possível situá-la no ecrã, é usada a letra X para representar o eixo horizontal e a letra Y para representar o eixo vertical (a origem no canto superior esquerdo do ecrã).

A biblioteca Xlib é a responsável por enviar os pedidos do cliente ao servidor X. No entanto as suas funções não enviam imediatamente os pedidos ao servidor, em vez disso, armazenam esses pedidos num *buffer* de saída. Para garantir que esse *buffer* é totalmente actualizado, é necessário usar as funções `XSync` ou `XFlush` que serão explicadas posteriormente.

De salientar ainda que o X *Window System* usa eventos para permitir uma interacção cliente-servidor, ou seja, existe uma série de eventos que podem indicar algo que o programa necessita saber, algo que o programa está a fazer e que outros clientes necessitam de saber ou algo que o gestor de janelas necessita de saber. Alguns desses eventos são: tecla pressionada, janela exposta, janela escondida, etc.. Os eventos recebidos são guardados numa fila que pode ser acedida pelo cliente, assim, enquanto o servidor X envia eventos, os clientes necessitam de aceder à fila para ler esses eventos.

- ***Display***

Para se poder usar um *display* é necessário estabelecer uma ligação ao servidor X. Só a partir desta ligação é possível começar a usar as funções existentes nesta biblioteca. Assim, para abrir um *display* é usado o seguinte código:

```
char *display_name;
Display *XOpenDisplay(display_name);
```

De salientar que a variável `display_name` deve conter o nome do *display* do *hardware* com o qual se deseja comunicar ou o valor da variável de ambiente DISPLAY. Se a variável `display_name` tiver o valor `NULL`, é equivalente a seleccionar o valor da variável de ambiente `DISPLAY` da máquina local.

A variável de ambiente `DISPLAY` é uma *string* com o seguinte formato:

```
hostname:number.screen_number
```

na qual `hostname` representa o nome da máquina local; `number` representa o número do servidor nessa máquina local e `screen_number` especifica o ecrã a ser usado nesse servidor.

A um *display* está associado um conjunto de propriedades. Existem funções para se identificar cada uma dessas propriedades (tal como o *display*, o ecrã tem também uma estrutura com as suas propriedades). Dentro dessas propriedades podem-se salientar:

Largura do *display:*

```
Display *display;
Int screen_number;
Int XDisplayWidth(display, screen_number)
```

Altura do *display*:

```
Int XDisplayHeight (display,scren_number)
```

Janela raiz:

```
Window XrootWindow (display, screen_number)
```

Display de um ecrã:

```
Screen *screen;
Display *XdisplayOfScreen (screen);
```

Janela raiz do ecrã:

```
Window XrootWindowOfScreen (screen);
```

Para fechar a conexão a um servidor X, um cliente deve executar a seguinte função:

```
Display *display;
XcloseDisplay (display)
```

Ao executar esta função, o servidor executa as seguintes operações:

- Retira todas as selecções que são propriedade do cliente
- Se o cliente tem o rato e o teclado activos, é lhe retirada essa propriedade (rouba o teclado e o rato)
- Se o cliente roubou o servidor, essa propriedade é-lhe retirada
- Retira ao cliente todos os roubos passivos por ele efectuados
- Marca todos os recursos alocados pelo cliente, quer como permanente quer como temporário, e dependendo do modo de encerramento ser *Retain Permanent* ou *Retain Temporary* .

- **Janela**

Ao criar uma janela, é gerada uma janela de área rectangular que permite a visualização de um *output* pedido a um servidor por um cliente. Esta janela tem um conjunto de propriedades que a caracterizam.

Criar uma Janela

Para criar uma janela com os atributos específicos, é usada a função `XCreateWindow()`, no entanto, se o objectivo for criar uma janela com os mesmos atributos que a sua janela mãe, é usada a função `XCreateSimpleWindow ()`. Ambas as funções criam uma janela não mapeada, logo, as janelas não ficam automaticamente visíveis. De salientar ainda que estas janelas usam o mesmo cursor que as suas mães até outro ser definido usando a função `XDefineCursor ()`.

Demonstra-se agora o uso de funções que permitem a criação de uma janela:

```
Display *display; // Display a ser usado

Window parent; // Especifica a janela mãe

Int x,y; // Representa as coordenadas onde esta janela vai ser criada, não esquecer que a origem se situa no canto superior esquerdo

Unsigned int width, height; // Representa a largura e a altura da janela a ser criada

Unsigned int border_width; // Especifica a largura em pixeis do bordo da janela criada

Int depth; //Especifica a profundidade da janela

Unsigned int class; // Especifica a classe da janela, Pode ser do tipo, InpuOutput, InputOnly ou CopyFromParent

Visual *visual; // Especifica o tipo de visual

Unsigned long valuemask; //Especifica que atributos estão definidos no argumento attributes. Se for zero, os atributos são ignorados e não referenciados

XSetWindowAttributes * attributes;// Especifica a estrutura a partir da qual os valores devem ser usados.

Window XCreateWindow (display,parent, x, y, width, height, border_width, depth,class,visual,valuemask,attributes)

Unsigned long border;// Especifica o valor de pixel do bordo da janela

Unsigned long background; //Especifica o valor de pixel do fundo da janela

XCreateSimpleWindow (display, parent, x, y, width, height, border_width, border, background)
```

Mapear uma Janela

Como já foi referido anteriormente, após ser criada uma janela esta não aparece no ecrã. Para tal acontecer, a janela criada necessita de ser mapeada e, uma vez mapeada, é gerado um evento de exposição (*Expose*). Existem três tipos de mapeamentos:

- Usando a função `XMapWindow()`, que permite mapear a janela e todas as suas sub janelas que requereram mapeamento. No caso de uma janela ter um ascendente que não foi mapeado, é marcada como ilegível quando o ascendente for mapeado. Esse tipo de janela é chamada "não visível". Quando todos os seus ascendentes estão mapeados, a janela torna-se passível de ser visível e passa a ser visível no ecrã, caso não seja ocultada por outra janela.

- Usando a `XMapRaised()` que é uma função similar a `XMapWindow()`, já que mapeia a janela e todas as suas subjanelas que requereram mapeamento. No entanto, também leva a janela especificada para o topo da *stack*.
- Usando `XMapSubwindows()` que mapeia todas as sub janelas referentes a uma janela específica na *stack top-to-bottom*.

Demonstra-se agora como usar as funções:

```
Display *display;
Window w;
XMapWindow (display, w);
XMapRaised (display, w);
XmapSubwindows (display, w);
```

Configuração de uma janela

É possível, com esta biblioteca, mover, redimensionar e modificar a largura do bordo. Para tal, é necessário alterar o membro apropriado da estrutura `XWindowChanges` e/ou a `value mask` da função `XConfigureWindow()`.

Bits do `value mask`:

```
#define CWX   (1<<0)
#define CWY   (1<<1)
#define CWWidth  (1<<2)
#define CWHeight (1<<3)
#define CWBorderWidth (1<<4)
#define CWSibling (1<<5)
#define CWStackMode (1<<6)
```

Valores da estrutura `XWindowChanges`:

```
typedef struct {
          int x, y;
          int width, height;
          int border_width;
```

```
            Window sibling;
            int stack_mode;
} XWindowChanges;
```

A maioria destes valores foi já referido anteriormente, portanto importa agora apenas referir os valores não descritos. O membro *sibling* permite seleccionar a janela *sibling* para operações de *stack*. Uma janela *sibling* é uma janela que tem a mesma janela mãe que outra, ou seja, é irmã. O membro `stack_mode` é usado para seleccionar o modo como a janela é colocada na *stack*, que pode ser `Above, Below, TopIf, BottomIf,` ou `Opposite`.

Assim, para configurar o tamanho de uma janela, a sua localização, o seu bordo ou a *stacking*, é usado o seguinte comando:

```
XWindowChanges * values;
XConfigureWindow (display, w, value_mask,values);
```

Para mover uma janela sem alterar o seu tamanho, é necessário usar o *display* usado, a janela a mover e as coordenadas da nova posição:

```
XMoveWindow (display, w, x, y);
```

Para modificar o tamanho da janela sem alterar a sua posição actual, bem como o seu bordo, é necessário usar o *display* actual, a janela a dimensionar, a largura e altura desejada:

```
XResizeWindow (display, w, width, height)
```

Para mudar o tamanho e a localização de uma janela, são necessários os argumentos usados nas duas funções anteriores:

```
XMoveResizeWindow (display, w, x, y, width, height)
```

Para modificar apenas a largura do bordo da janela é necessário especificar o *display*, a janela e a largura desejada:

```
XSetWindowBorderWidth (display, w, width)
```

Alterar os atributos de uma janela

É importante modificar os atributos de uma janela, de modo a proporcionar o aspecto desejado. Para tal, pode ser usada uma função mais generalista, `XChangeWindowAttributes()`, ou funções específicas para o atributo que se pretende alterar. Assim, para usar a função `XChangeWindowAttributes()` é necessário mexer na estrutura `XSetWindowAttributes`. Os atributos disponíveis diferem consoante se trate de uma janela `InputOnly` ou de uma janela `InputOutput`. As janelas `InputOnly` são invisíveis e são usadas para controlar eventos de entrada em situações em que janelas `InputOutput` são desnecessárias e apenas podem ser usadas para controlar cursores, eventos de entrada e o roubo de propriedades (roubo do cursor, do teclado). Estas janelas não podem ter janelas `InputOutput` como descendentes, nem responder à requisição de gráficos.

É importante agora identificar os atributos que se aplicam a janelas `InputOnly` e a janelas `InputOutput` bem como a estrutura que permite modificar os atributos.

Tabela 2 Atributos InputOutput e InputOnly[25]

Atributo	Default	InputOutput	InputOnly
background-pixmap	Nenhum	Sim	Não
background-pixel	Indefinido	Sim	Não
border-pixmap	Copiado do Pai	Sim	Não
border-pixel	Indefinido	Sim	Não
bit-gravity	Esquecer Gravidade	Sim	Não
win-gravity	Gravidade Noroeste	Sim	Sim
backing-store	Não necessário	Sim	Não
backing-planes	Todos	Sim	Não
backing-pixel	Zero	Sim	Não
save-under	False	Sim	Não
event-mask	Vazio	Sim	Sim
do-not-propagate-mask	Vazio	Sim	Sim
override-redirect	False	Sim	Sim
colormap	Copiado do Pai	Sim	Não
cursor	Nenhum	Sim	Sim

Bits da variável `valuemask`:

```
#define CWBackPixmap        (1L<<0)
#define CWBackPixel         (1L<<1)
#define CWBorderPixmap      (1L<<2)
#define CWBorderPixel       (1L<<3)
#define CWBitGravity        (1L<<4)
#define CWWinGravity        (1L<<5)
#define CWBackingStore      (1L<<6)
#define CWBackingPlanes     (1L<<7)
#define CWBackingPixel      (1L<<8)
#define CWOverrideRedirect  (1L<<9)
#define CWSaveUnder         (1L<<10)
#define CWEventMask         (1L<<11)
#define CWDontPropagate     (1L<<12)
#define CWColormap          (1L<<13)
#define CWCursor            (1L<<14)
```

Variáveis da estrutura `XSetWindowAttributes`:

```
Typedef struct {

          Pixmap background_pixmap; // com background, nenhum, ou
relativo a janela mãe

          Unsigned long background_pixel; // pixel de background

          Pixmap border_pixmap; // Bordo fora da janela ou
CopyFromParent

          Unsugned long border_pixel; // Valor do pixel do bordo

          Int bit_gravity;

          Int win_gravity;

          Int backing_store; //NotUseful,WhenMapped,Always

          Unsigned long backing_planes; //planos a serem preservados

          Unsigned long backing_pixel; // valor a usar na recuperação
de planos

          Bool save_under; // gravar bits abaixo 1, não gravar bits
abaixo 0
```

```
          Long event_mask; // seleccionar os eventos a ser gravados

          Long do_not_propagate_mask; // seleccionar os bits que não
devem ser propagados

          Bool override_redirect; // valor booleano para
override_redirect

          Colormap colormap; // Mapa de cores a ser associado à janela

          Cursor cursor; // Cursor a ser mostrado, ou nenhum se
desejado

          }XSetWindowAttributes;
```

Tal como referido anteriormente, para mudar um ou mais atributos de uma dada janela, é usada a função `XChangeWindowAttributes()`:

```
Unsigned long valuemask;

XSetWindowAttributes *attributes;

XchangeWindowAttributes (display, w, valuemask,attributes)
```

Para definir apenas o fundo de uma janela para um dado *pixel*, é necessário o *display*, a janela desejada e o *pixel* desejado:

```
Unsigned long background_pixel,

XSetWindowBackground(display, w, background_pixel);
```

Já para mudar o fundo para um dado mapa de *pixeis* é usada a seguinte função, que necessita do *display*, da janela desejada e do mapa de *pixeis* desejado:

```
Pixmap background_pixmap;

XSetWindowBackgroundPixmap(display, w, background_pixmap)
```

Tal como para o *background*, é possível mudar a cor do bordo para um determinado *pixel* ou para um mapa de *pixeis*:

```
Unsigned long border_pixel;

Pixmap border_pixmap;

XSetWindowBorder (display, w, border_pixel)

XsetWindowBorderPixmap(display, w, border_pixmap)
```

Para definir o mapa de cores de uma janela, é usada a função `XSetWindowColormap()`, que necessita das variáveis *display*, da janela desejada e do mapa de cores desejado.

```
Colormap colormap;

XSerWindowColormap (display, w, colormap);

É ainda possível definir e retirar o cursor a uma janela.

Cursor cursor;

XDefineCursor(display, w, cursor);

XUdefineCursor( display, w);
```

Ocultar e destruir uma Janela

Num programa, torna-se por vezes, por vários motivos, necessário ocultar uma janela ou as suas sub janelas. Com o Xlib isso é possível usando as seguintes funções:

```
XUnmapWindow (display,w); //para ocultar uma janela

XUnmapSubwindows (display,w); //para ocultar apenas as sub janelas da
janela seleccionada
```

De salientar que com a função `XUnmapWindow()`, não só a janela seleccionada é ocultada como também as suas sub janelas (se existirem).

Por vezes é necessário destruir uma janela específica (por exemplo no fim de um programa, quando esta já não é necessária). Tal como para ocultar janelas, para destruir também existem duas opções: destruir uma janela e as suas sub janelas (`XDestroyWindow()`) ou apenas destruir as suas sub janelas (`XDestroySubwindows()`):

```
XDestroyWindow(display,w);//para destruir uma janela e as suas
subjanelas

XDestroySubwindows(display,w); // para destruir apenas as sub janelas
```

- ***Graphics Context* (GC)**

Quando se pretendem realizar inúmeras operações de desenho, é necessário especificar várias opções para chegar à forma desejada (fundo, cor, ponteiro, texto, etc.). Para tal é usada uma estrutura com o contexto gráfico desejado (uma estrutura GC), na qual se definem as várias opções desta estrutura, e de seguida passa-se um apontador desta estrutura para a rotina de desenho. Assim, não é necessário definir as opções para todas as

funções de desenho, pois caso se deseje manter sempre o mesmo "estilo", basta chamar sempre a mesma estrutura GC. As operações gráficas podem ser realizar quer em janelas quer em mapas de *pixeis*, que são colectivamente chamados *drawables*. Convém no entanto referir que cada *drawable* existe num único ecrã, ou seja, uma estrutura GC é criada para um ecrã específico.

Criar uma GC

Para editar um atributo específico de uma GC, é necessário editar o membro correcto da estrutura `XGCValues` e/ou o valor da *bitmask* correspondente na chamada à função `XCreateGC ()`.

Bits correspondentes na *bitmask*:

```
#define GCFunction      (1L<<0)
#define GCPlaneMask     (1L<<1)
#define GCForeground    (1L<<2)
#define GCBackground    (1L<<3)
#define GCLineWidth     (1L<<4)
#define GCLineStyle     (1L<<5)
#define GCCapStyle      (1L<<6)
#define GCJoinStyle     (1L<<7)
#define GCFillStyle     (1L<<8)
#define GCFillRule      (1L<<9)
#define GCTile          (1L<<10)
#define GCStipple       (1L<<11)
#define GCTileStipXOrigin   (1L<<12)
#define GCTileStipYOrigin   (1L<<13)
#define GCFont              (1L<<14)
#define GCSubwindowMode     (1L<<15)
#define GCGraphicsExposures (1L<<16)
#define GCClipXOrigin       (1L<<17)
#define GCClipYOrigin       (1L<<18)
```

```
#define GCClipMask          (1L<<19)

#define GCDashOffset        (1L<<20)

#define GCDashList          (1L<<21)

#define GCArcMode           (1L<<22)
```

Estrutura GC:

```
typedef struct {

          int function;                  /* logical operation */

          unsigned long plane_mask;      /* plane mask */

          unsigned long foreground;      /* foreground pixel */

          unsigned long background;      /* background pixel */

          int line_width;                /* line width (in pixels) */

          int line_style;                /* LineSolid, LineOnOffDash,
LineDoubleDash */

          int cap_style;                 /* CapNotLast, CapButt,
CapRound, CapProjecting */

          int join_style;                /* JoinMiter, JoinRound,
JoinBevel */

          int fill_style;                /* FillSolid, FillTiled,
FillStippled FillOpaqueStippled*/

          int fill_rule;                 /* EvenOddRule, WindingRule */

          int arc_mode;                  /* ArcChord, ArcPieSlice */

          Pixmap tile;       /* tile pixmap for tiling operations*/

          Pixmap stipple;    /* stipple 1 plane pixmap for stippling */

          int ts_x_origin;   /* offset for tile or stipple operations */

          int ts_y_origin;

          Font font;               /* default text font for text
operations */

          int subwindow_mode;            /* ClipByChildren,
IncludeInferiors */

          Bool graphics_exposures; /* boolean, should exposures be
generated */

          int clip_x_origin;       /* origin for clipping */

          int clip_y_origin;

          Pixmap clip_mask;        /* bitmap clipping; other calls for
rects */
```

```
        int dash_offset;        /* patterned/dashed line information
*/
        char dashes;
} XGCValues;
```

Para criar um GC é necessária a função `XCreateGC()`:

```
Display *display;
Drawable d;
unsigned long valuemask;
XGCValues *values;
GC XCreateGC(display, d, valuemask, values)
```

Como se pode verificar, esta função necessita de quatro argumentos:

- *Display*, que representa a conexão ao servidor X;
- *Drawable*, que representa o ID de um *pixmap* ou de uma janela; apesar de representar o ID de um *pixmap* ou de uma janela, esta variável serve para identificar o ecrã a ser usado;
- *Valuemask,* que representa os elementos da estrutura `XGCValues` que vão ser lidos (bit a 1);
- *Values,* que representa a estrutura `XGCValues`.

Funções GC

Definir a cor do primeiro plano:

```
XSetForeground(display, gc, foreground)
Display *display;
GC gc;
unsigned long foreground;
exemplo: XSetForeground(display, gc, WhitePixel(display, screen_num));
```

Definir a cor de *background*:

```
XSetBackground(display, gc, background)
unsigned long background;
```

```
exemplo: XSetBackground(display, gc, BlackPixel(display, screen_num));
```

Definir os atributos relativos às linhas:

```
XSetLineAttributes(display, gc, line_width, line_style, cap_style,
join_style)

unsigned int line_width; // largura da linha

int line_style;// estilo da linha: LineSolid, LineOnOffDash, ou
LineDoubleDash.

int cap_style; //estilo da linha e da cap: CapNotLast, CapButt,
CapRound, or CapProjecting

int join_style; //estilo da junção: JoinMiter, JoinRound, or JoinBevel

exemplo: XSetLineAttributes(display, gc,line_width, line_style,
cap_style, join_style);
```

Definir o estilo de preenchimento:

```
XSetFillStyle(display, gc, fill_style)

int fill_style; // FillSolid, FillTiled, FillStippled, or
FillOpaqueStippled

exemplo: XSetFillStyle(display, gc, FillSolid);
```

Definir a fonte:

```
XSetFont(display, gc, font)

Font font; //especifica a fonte

Exemplo:  XSetFont(display, gc, font_info->fid);

          XFontStruct* font_info;

          char* font_name = "*-helvetica-*-24-*";
```

- **Funções gráficas**

Esta é uma área importante pois permite limpar, criar áreas , desenhar pontos, linhas, texto, etc. De seguida demonstram-se as principais funções.

Limpar uma área rectangular

```
XClearArea(display, w, x, y, width, height, exposures)

      Window w;

      int x, y; //coordenadas de origem da janela
```

```
      unsigned int width, height; // largura e altura do rectângulo a
limpar

      Bool exposures;// Flag que define se evento de exposição é gerado
```

Limpar uma janela inteira

```
XClearWindow(display, w)
```

Desenhar um ponto

```
XDrawPoint(display, d, gc, x, y)

Drawable d; //janela ou pixmap

int x, y;//coordenadas do ponto a desenhar
```

Desenhar vários pontos

```
XDrawPoints(display, d, gc, points, npoints, mode)

Drawable d; //janela ou pixmap

XPoint *points;//uma estrutura de arrays de pontos

int npoints;//número de pontos

int mode; //modo de coordenadas : CoordModeOrigin ou CoordModePrevious
```

Desenhar uma linha

```
XDrawLine(display, d, gc, x1, y1, x2, y2)

int x1, y1, x2, y2;//especifica os pontos (x1,y1) e (x2,y2) a serem
ligados
```

Desenhar várias linhas

```
XDrawLines(display, d, gc, points, npoints, mode)

XPoint *points;//especifica um array de pontos

int npoints;//número de pontos no array

int mode; // modo de coordenadas : CoordModeOrigin or CoordModePrevious
```

Desenhar um rectângulo

```
XDrawRectangle(display, d, gc, x, y, width, height)

int x, y;//coordenadas de origem do rectângulo

unsigned int width, height;//largura e altura do rectângulo
```

Desenhar vários rectângulos

```
XDrawRectangles(display, d, gc, rectangles, nrectangles)

XRectangle rectangles[]; //Especifica um array de rectângulos

int nrectangles;//Número de rectângulos no array
```

Desenhar um arco circular ou elíptico

```
XDrawArc(display, d, gc, x, y, width, height, angle1, angle2)

int x, y; //Especifica as coordenadas x e y

unsigned int width, height; //Especifica a largura e altura, que são o
maior e o menor eixo do arco

 int angle1, angle2;//angle1: especifica o início do arco relativamente
à posição das 3 horas num relógio, em graus*64; angle2: especifica o
caminho e extensão do arco relativamente ao inicio do arco, em graus*64
```

Listar as fontes disponíveis

```
char **XListFonts(display, pattern, maxnames, actual_count_return)

char *pattern;//especifica a string de padrão não terminado que pode
conter caracteres wildcard

int maxnames;//especifica o número máximo de nomes a serem retornados

int *actual_count_return; //retorna o número actual de nomes de fontes
```

Desenhar texto numa janela

```
XDrawText(display, d, gc, x, y, items, nitems)

int x, y;//origem do texto a ser desenhado

XTextItem *items;//um array de itens de texto (estrutura)

int nitems;//Número de itens de texto no array
```

Desenhar uma *string* numa janela

```
XDrawString(display, d, gc, x, y, string, length)

int x, y;//origem da string a desenhar

char *string;//string a desenhar

int length; //tamanho da string
```

- **Eventos**

Uma vez que já foi referida a importância dos eventos num sistema de janelas X, importa agora referir os vários tipos de eventos existentes, bem como a forma de os tratar. Um evento é gerado no servidor X como resultado da actividade de um dispositivo ou como efeito colateral de um pedido enviado por uma função Xlib (por exemplo, `XMapWindow()` gera um evento de exposição). É portanto importante conhecer os vários tipos de eventos existentes e as categorias através das quais estes se dividem. A tabela seguinte demonstra essa divisão.

Tabela 3 Categorias de eventos e tipos de eventos[25]

Categorias de Eventos	Tipos de Eventos
Keyboard events	KeyPress, KeyRelease
Pointer events	ButtonPress, ButtonRelease, MotionNotify
Window crossing events	EnterNotify, LeaveNotify
Input focus events	FocusIn, FocusOut
Keymap state notification event	KeymapNotify
Exposure events	Expose, GraphicsExpose, NoExpose
Structure control events	CirculateRequest, ConfigureRequest, MapRequest, ResizeRequest
Window state notification events	CirculateNotify, ConfigureNotify, CreateNotify, DestroyNotify, GravityNotify, MapNotify, MappingNotify, ReparentNotify, UnmapNotify, VisibilityNotify
Colormap state notification event	ColormapNotify
Client communication events	ClientMessage, PropertyNotify, SelectionClear, SelectionNotify, SelectionRequest

Normalmente, o servidor X só envia um evento para um cliente se este pediu especificamente para o receber. Para tal, o cliente deve modificar o atributo `event-mask` de uma janela. No entanto, geralmente, todos os eventos de mapeamento são enviados para todos os clientes.

Importante na operação com eventos é a estrutura de eventos. Assim sendo, todos os eventos têm a seguinte estrutura:

```
typedef struct {

          int type; //tipo de evento, id do evento

          unsigned long serial;    // último pedido processado pelo
servidor

          Bool send_event; //caso tenha sido enviado usando XsendEvent,
tem o valor TRUE

          Display *display;  //o display no qual foi lido o evento

          Window window; //janela inerente ao evento

} XAnyEvent;
```

De salientar que para além das estruturas para cada evento, existe uma estrutura, a `XEvent`, que é uma união das estruturas individuais declaradas para cada evento.

Para seleccionar um evento, é necessário conhecer a sua máscara bem como as circunstâncias para as quais este deve ser usado, algumas das quais são demontradas na tabela seguinte.

Tabela 4 Máscaras de eventos e suas circunstâncias[25]

Máscara de Eventos	Circunstâncias
NoEventMask	Nenhum evento desejado
KeyPressMask	Detectar tecla pressionada no teclado
KeyReleaseMask	Detectar tecla solta no teclado
ButtonPressMask	Detectar click no rato
ButtonReleaseMask	Detectar botão do rato solto
EnterWindowMask	Detectar entrada do rato na janela
LeaveWindowMask	Detectar saída do rato na janela
PointerMotionMask	Detectar movimento do rato
PointerMotionHintMask	Detectar *hints* do movimento do rato
Button1MotionMask	Detectar movimento do rato enquanto botão 1 pressionado
Button2MotionMask	Detectar movimento do rato enquanto botão 2 pressionado
Button3MotionMask	Detectar movimento do rato enquanto botão 3 pressionado
Button4MotionMask	Detectar movimento do rato enquanto botão 4 pressionado
Button5MotionMask	Detectar movimento do rato enquanto botão 5 pressionado

Tratar os Eventos

Existem várias formas de seleccionar os eventos que se pretende conhecer, tal pode ser realizado no instante em que se cria a janela (`XCreateWindow()`), ao alterar os atributos de uma janela (`XChangeWindowAttributes()`ou com uma função específica, a função `XSelectInput()`. Uma vez que as duas primeiras já foram referidas, demonstra-se agora como usar a terceira função.

```
XSelectInput(display, w, event_mask)
```

```
        Display *display;
        Window w;
        long event_mask; // identificador do evento desejado
exemplo: XSelectInput (display,w,KeyPressMask);
```

Uma vez que os eventos ficam guardados numa fila, é necessário tratar os eventos existentes nessa fila. Para tal podem ser usadas as funções `XFlush()` e `XSync()`. Enquanto a primeira "lê" o *buffer* de saída, a segunda "lê" o *buffer* de saída e aguarda até todos os pedidos serem processados. De salientar que, dependendo do desejado, a função `XSync()` permite descartar o conteúdo do *buffer*. Assim:

```
XSync(display, discard)
        Display *display;
        Bool discard; // Se estiver a 1 limpa a fila de eventos
```

Para obter os eventos presentes na fila, é possível usar as funções `XNextEvent()` ou `XPeekEvent()`: a primeira copia o evento para estrutura seleccionada e remove-o da fila, a segunda copia o evento para a estrutura seleccionada mas não o remove da fila.

```
XEvent *event_return; //estrutura de um evento
XNextEvent(display, event_return)
XPeekEvent(display, event_return)
```

Estas funções obrigam a determinar se o evento recebido corresponde ao evento desejado, no entanto existem funções que permitem seleccionar apenas os eventos desejados. Assim, para verificar se o evento desejado se encontra na fila e, em caso positivo o remove-lo, é usada a função `XIfEvent();` para verificar se o evento desejado se encontra na fila, sem o bloquear, é usada a função `XCheckIfEvent()`; por fim, para verificar se o evento desejado se encontra na fila de eventos, sem o remover da fila, é usada a função `XPeekIfEvent()`.

Para colocar um evento de volta na fila de eventos, basta seleccioná-lo, assim como o seu *display*.

```
XPutBackEvent(display, event)
        XEvent *event;
```

Uma função bastante útil, `XSendEvent()`, permite o envio de um evento para uma janela específica.

```
Status XSendEvent(display, w, propagate, event_mask, event_send)
      Display *display;
      Window w;//neste caso, é a janela para a qual se quer enviar
      Bool propagate;//modo de propagação. TRUE ou FALSE
      long event_mask;//máscara do evento a enviar
      XEvent *event_send;//estrutura do evento a enviar
Exemplo:XSendEvent(display,w,TRUE,KeyPressMask,&event)
```

- **Entradas de Dados**

Com esta biblioteca é possível trabalhar com os dispositivos de entrada, sendo possível roubar o rato (`XGrabPointer()`), retirar o rato (`XUngrabPointer()`), roubar o botão do rato (`XGrabButton()`) e retirar o botão de rato roubado (`XUngrabButton()`).

O mesmo processo também é possível para o teclado: `XGrabKeyboard()` para roubar o teclado, `XUngrabKeyboard()` para retirar o teclado roubado, `XGrabKey()` para roubar uma única tecla do teclado, `XUngrabKey()` para retirar a tecla roubada.

Outras das funções importantes desta biblioteca são as que permitem saber quem tem o foco, bem como atribuí-lo. Para saber que janela tem o foco:

```
XGetInputFocus(display, focus_return, revert_to_return)
      Window *focus_return;//retorna a janela que tem o foco
      int *revert_to_return//retorna o estado do foco: RevertToParent,
RevertToPointerRoot, ou RevertToNone
```

Já para atribuir o foco:

```
XSetInputFocus(display, focus, revert_to, time)
   Window focus;// janela a que se quer atribuir o foco
   int revert_to; //especifica para onde reverter se a janela não se
tornar visível: RevertToParent, RevertToPointerRoot, ou RevertToNone
   Time time; // específica o tempo, normalmente vale CurrentTime
```

[15][25][26]

Anexo C. Libxml

A Libxml é uma biblioteca, escrita em C, que permite trabalhar ficheiros XML (ler, criar e manipular dados). No entanto, para a presente Tese, as características mais importantes da Libxml são: permitir parcelar um documento XML, extrair o texto dentro de um elemento específico e extrair o valor de um atributo.

- **Tipos de Dados**

Para ser possível trabalhar com esta biblioteca é importante conhecer os tipos de dados envolvidos, sendo estes:

- `xmlChar,` uma substituição para *chars*;
- `xmlDoc`, uma estrutura que contém a árvore de atributos criada por um documento parcelado;
- `xmlDocPtr`, um apontador para a estrutura anteriormente referida;
- `xml-Node`, uma estrutura que contem um único nó;
- `xmlNodePtr`, um apontador para a estrutura anteriormente referida e usado para viajar ao longo da árvore do documento (por exemplo, para descer um nível na árvore: `xmlNodePtr sub; sub=sub->next;`)

- **Funções**

As funções nesta biblioteca são antecedidas pela mnemónica "xml". Seguidamente serão apresentadas as principais funções existentes na Libxml.

Escolher ficheiro a parcelar

```
xmlDocPtr doc;
doc=xmlParseFile(nomedodocumento);
```

Obter o elemento raiz do documento

```
xmlNodePtr cur;
cur=xmlDocGetRootElement(doc);
```

Comparação do nó actual com o nó desejado

```
xmlStrcmp(cur->name,(const xmlChar*)"tv")
```

Onde `cur->name` representa o nome do nó actual e `(const xmlChar*)"tv"` um *cast* para tornar a *string* "tv" numa `xmlChar`.

Obter o valor de uma propriedade

```
xmlChar *txt;
txt=xmlGetProp (sub, (const xmlChar *)"channel");
```

Onde `sub`, representa o nó actual e `(const xmlChar*)"channel"` um *cast* para tornar a *string* "channel" numa `xmlChar`. Esta função retorna o valor desta propriedade.

Descer na Árvore

```
xmlNodePtr sub;
Sub= cur->children;
```

Obter conteudo de um nó

```
Txt=xmlNodeListGetString (doc-cur->xmlCgildrenNode,1);
```

Uma vez que esta função aloca memória para a *string* que retorna, é necessário usar a função `XmlFree(txt)` para a libertar.

Libertar o documento

Por fim, é necessário libertar o documento que está a ser usado.

```
xmlFreeDoc(doc);
```

[32]