Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informáica
Departamento Engenharia Elétrica

TRABALHO DE CONCLUSÃO DE CURSO

Interface de Programação para Desenvolvimento de Aplicações Baseadas em Linux para Dispositivos Móveis

Aluno: Miguel Augusto de Souza Falcão
Matrícula: 20411225
Professor Orientador: Marcos Ricardo Morais

Campina Grande
Fevereiro de 2010

Sumário

1 Introdução

Ao longo dos últimos anos, sistemas móveis embarcados, como aparelhos celulares, PDAs e Internet *tablets*, agregaram funcionalidades de outros dispositivos, tornando-se equipamentos de múltiplo propósito. Neste sentido, é possível encontrar celulares que além de realizar chamadas telefônicas, também fornecem suporte a gravação e reprodução de áudio e vídeo, tiram fotos, armazenam grandes quantidades de dados, etc. Ao passo que novas funcionalidades são agregadas, *softwares* mais complexos são requeridos. Desta maneira, desenvolvedores devem saber como acessar os recursos do dispositivo e lidar com suas restrições, como tamanho reduzido de tela, níveis de bateria, capacidade de armazenamento e processamento [1].

Plataformas de desenvolvimento existentes para dispositivos móveis apresentam algumas particularidades no tocante ao acesso e utilização dos recursos do sistema, requerendo o entendimento de suas tecnologias mais complexas ou de baixo nível. Grande parte destas plataformas são relativamente recentes e com uma documentação raramente satisfatória. Estes aspectos adicionam novos desafios ao desenvolvimento de aplicações móveis. A falta de informação é um problema recorrente enfrentado pelos desenvolvedores da plataforma Maemo [2], tornando a implementação de aplicações para dispositivos móveis um desafio para desenvolvedores.

Este Trabalho de Conclusão de Curso (TCC) apresenta a Easy, uma biblioteca que serve de interface de programação para facilitar o desenvolvimento de aplicações baseadas em Linux para dispositivos móveis que utilizam Maemo. Easy tem como objetivo prover acesso simplificado aos recursos específicos da Maemo, construindo uma camada de abstração no topo de sua arquitetura. Para ilustrar as vantagens de se utilizar a Easy, é apresentada uma comparação entre a biblioteca e outras tecnologias amplamente utilizadas em aplicações móveis. Easy é distribuída sob licença *Lesser General Public License* (LGPL) [3], é implementada em linguagem de programação Python [4] e faz parte do projeto PerComp [5] do Laboratório de Sistemas Embarcados e Computação Pervasiva (*Embedded*) [6].

2 Metodologia

Na execução do projeto, o processo de desenvolvimento de *software* utilizado foi XP [7] que segue uma metodologia de desenvolvimento leve. XP enfatiza o uso de práticas que se mostram funcionais, como revisão de código, teste, simplicidade e ciclo curto e iterativo [8]. Ele visa alcançar duas metas desejadas pelas indústrias de tecnologia da informação: desenvolvimento rápido e consistente com as reais necessidades do cliente e uma fácil manutenibilidade permitindo que o *software* seja modificado é medida que as necessidades se alterem.

Existiram três papéis desempenhados pelos membros da equipe. O gerente de projeto foi o mestrando Mario Hozano. O papel do cliente foi desempenhado pelo INdT (Instituto Nokia de Tecnologia) e o papel dos desenvolvedores foi desempenhado por André Hora, Emanuel Dantas e Miguel Falcão, cabendo a este último membro o desenvolvimento dos módulos de áudio, vídeo, rádio e VoIP. As estórias foram definidas pelo cliente.

As reuniões entre desenvolvedores e gerente ocorriam semanalmente. Entretanto, o contato entre cliente, gerente e desenvolvedores ocorria diariamente através da web, visto que o cliente se encontrava em outra cidade.

O projeto Easy utilizou o ambiente GForge [9] como forma de interaçõo entre a equipe de desenvolvimento e cliente. Tal ambiente é uma ferramenta corporativa para desenvolvimento de *software* de código aberto. O projeto apresentado utiliza de forma intensiva vários recursos disponíveis no ambiente, como fóruns, listas de discussões, notícias, etc.

O sistemas de controle de versão de código-fonte (SVN) é utilizado, além da disponibilidade de distribuição dos arquivos do projeto através de pacotes de instalação. Tais recursos são de fundamental importância para o desenvolvimento do projeto, pois o cliente obtém realimentação diária de tudo que está sendo implementado e interage de forma mais efetiva com os membros da equipe.

3 Plataforma Maemo

Maemo é uma plataforma de desenvolvimento de código aberto para dispositivos móveis baseados em Linux, como os Nokia Internet *tablets* 770, N800 e N810 (Figura 1). Algumas das funcionalidades apresentadas pelos dispositivos Maemo incluem tela de alta resolução sensível a toque, câmera integrada, microfone, alto-falantes, receptor GPS, chip sintonizador de rádio FM, conectividade sem fio através de Wi-fi e Bluetooth, etc.

Figura 1: Internet *tablets* Nokia 770, N800 and N810

A plataforma Maemo está atualmente em sua versão 5 e é constituída de componentes largamente utilizados em computadores convencionais e sistemas móveis, como GTK+ [10], D-Bus [11], Hildon [12], etc. Para acessar os seus recursos é requerido o conhecimento de componentes de baixo nível. Basicamente, o desenvolvimento de aplicações para a plataforma é feito através da linguagem C, uma característica das plataformas Linux. Entretanto, aplicações em Python podem ser implementadam por desenvolvedores através da utilização de componentes de ligações (*bindings*) fornecidas pelo projeto PyMaemo [13].

4 Biblioteca Easy

Easy é uma biblioteca escrita em Python que visa fornecer uma camada de abstração de acesso aos recursos disponíveis em dispositivos Maemo, como câmera, microfone, alto-falante, sintonizador de rádio e Bluetooth.

A decisão de utilizar Python no projeto Easy se deu devido à sua extensa biblioteca padrão, grande apoio da comunidade de desenvolvedores e desenvolvimento rápido de aplicações. No mais, Python é considerado uma ótima alternativa para prototipagem rápida e programação em ambiente Maemo, sendo, após a linguagem C, a mais utilizada nessa plataforma [13].

A arquitetura da biblioteca Easy (Figura 2) define os módulos que acessam os recursos da plataforma através de ligações fornecidas pelo PyMaemo. Enquanto PyMaemo dá acesso a funções oferecidas por componentes como BlueZ [14], LibOSSO, D-Bus, etc., Easy simplifica a utilização dos mesmos, fornecendo métodos diretos e intuitivos divididos em 10 módulos: *Audio*, Bluetooth, *Camera*, *Contacts*, E-mail, GPS, *Radio*, UI (*User Interface*), UPnP (*Universal Plug and Play*) e VoIP.

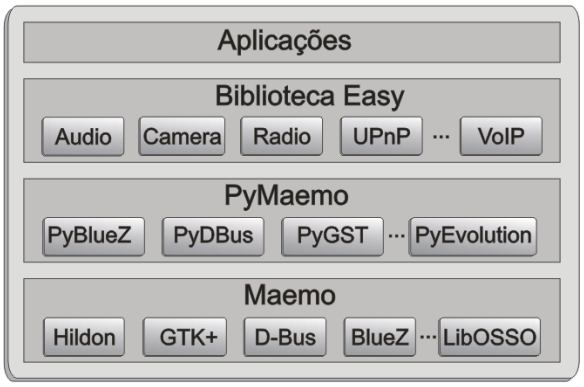

Figura 2: Arquitetura da biblioteca Easy.

Ao utilizar a biblioteca Easy, os desenvolvedores podem aumentar sua produtividade, desenvolvendo aplicações de uma maneira mais rápida e escrevendo menos código. O principal objetivo da biblioteca não é de apenas facilitar o desenvolvimento de *software* para programadores experientes, mas também de tornar possível a implementação de aplicações por programadores com apenas conhecimento básico da linguagem de programação Python. Os módulos da biblioteca e alguns exemplos ilustrativos são descritas a seguir. A documentação completa e exemplos adicionais de todos os módulos podem ser encontrada no site do projeto em [15].

4.1 *Audio*

O módulo *Audio* fornece métodos diretos para a gravação e processamento de arquivos de áudio, além de incluir funções para controle de volume, busca e obtenção de informações sobre a duração e a posição atual do arquivo em reprodução.

Ele encapsula um conjunto de funcionalidades do GStreamer [16], uma biblioteca liberado sob licença LGPL que permite a construção de aplicações multimídia através de uma abstração de *pipelines* com conexão de elementos manipuladores de mídia, onde as aplicações podem tirar proveito de avanços nas tecnologias de codecs e filtros de uma forma transparente. Através desta biblioteca, os desenvolvedores podem adicionar novos componentes, escrevendo *plugins* genéricos com uma interface limpa. A motivação para a sua criação foi a necessidade das aplicações multimídia de uma padronização visto que implementavam as mesmas funcionalidades de maneiras diferentes, onde os esforços eram duplicados e não havia cooperação. As aplicações só poderiam ser implementadas quando toda infra-estrutura estivesse funcionando. GStreamer fornece uma padronização através de uma camada de abstração sobre componentes de manipulação de qualquer formato para o qual haja um *plugin* instalado. Na Figura 3 pode-se observar o posicionamento da plataforma no desenvolvimento de aplicações.

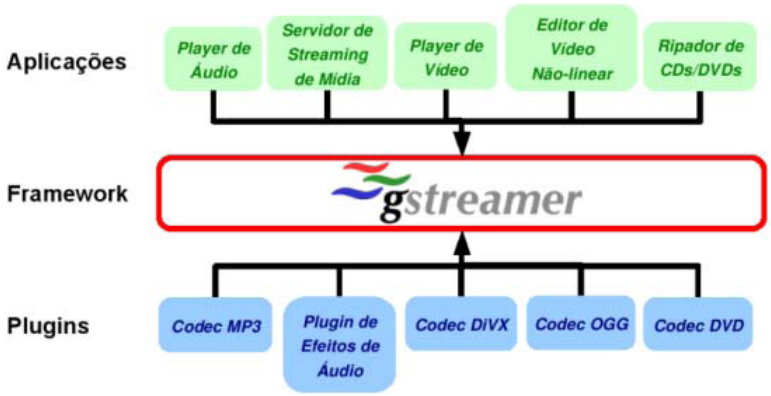

Figura 3: Posicionamento da plataforma GStreamer. [17]

Um exemplo de aplicação usando o módulo de áudio é mostrado na Listagem 1.

```
from easy import audio                                    1
audio.record('audio.wav', 10)                             2
audio.play('audio.wav')                                   3
```

Listagem 1: Gravação e execução com o módulo *Audio*.

Usando três linhas de código é possível gravar o áudio capturado pelo microfone do dispositivo e reproduzi-lo nos alto-falantes. Após carregar os módulos necessários (Linha 1), o áudio é gravado durante 10 segundos (Linha 2) e, em seguida, esse arquivo é reproduzido (Linha 3).

4.2 *Camera*

No módulo *Camera*, as funções relacionadas com a câmera e manipulação de vídeo são fornecidas. O módulo contém métodos para gravação e reprodução de arquivos de vídeo, bem como a captura de fotos. Outras funções são a busca no vídeo, obtenção de informações de comprimento e a posição atual do arquivo em execução, além de permitir alterar as propriedades de vídeo como: brilho, contraste, cor e saturação.

O módulo encapsula a plataforma GStreamer para acessar a câmera do dispositivo. Na Listagem 2 é ilustrado o código fonte utilizado para exibir a imagem capturada pela câmera do aparelho em uma resolução de 640x480 (Linha 2). Um vídeo de 10 segundos é gravado (Linha 3). Em seguida, tal vídeo é reproduzido na tela (linha 4) e uma foto é capturada (Linha 5).

```
from easy import camera                                    1
camera.start(camera.RESOLUTION_640x480)                    2
camera.record('video.avi', 10)                             3
camera.play('video.avi')                                   4
camera.click('photo.jpg')                                  5
```

Listagem 2: Gravação, execução e captura de foto com o módulo *Camera*.

4.3 *Radio*

O módulo de rádio fornece uma camada de abstração para o uso do chip sintonizador de FM disponível em alguns dispositivos Maemo, como o N800. As funções providas no módulo permitem escutar emissoras de rádio FM através de métodos de sintonia de frequência manual e automática. Fornece também função para controle de volume. Foi criado um elemento especial com a plataforma GStreamer para acessar o chip sintonizador FM.

```
from easy import radio                                     1
radio.start()                                              2
radio.tune(93100)                                          3
radio.next_station()                                       4
```

Listagem 3: Sintonização de estação FM utilizando o módulo *Radio*.

Na Listagem 3 é mostrado o código fonte para inicializar o chip de rádio (Linha 2), ajustá-lo à frequência de 93.1 MHz (Linha 3) e, em seguida realizar a sintonia automática para a próxima estação disponíveis (Linha 4).

4.4 UPnP

O módulo UPnP (*Universal Plug and Play*) fornece uma interface para criar dispositivos UPnP [18] de uma forma fácil. O módulo usa o projeto BRisa, um plataforma UPnP que permite aos utilizadores descobrir dispositivos multimídia, compartilhar, procurar e processar conteúdos multimídia através de redes locais ou remotamente através da Internet. O BRisa implementa a especificação UPnP, que provê serviços pervarsivos como configuração zero e auto-descoberta de dispositivos e serviços [19].

O módulo UPnP da Easy cria uma camada de abstração sobre a plataforma BRisa utilizando *annotations* de Python [20] para definir os parâmetros e as ações dos dispositivos e serviços UPnP. Ao usar o módulo, não é necessário para o usuário lidar com *parsers* XML ou especificação UPnP.

A Listagem 4 ilustra o código fonte que implementa um dispositivo UPnP simples, uma calculadora. Esta implementação provê a operação de soma (Linhas 11 a 14), a maneira de fornecer a descrição UPnP do dispositivo (linhas 8 a 10) e os anúncios e serviços do dispositivo UPnPCalculator (Linha 18).

```
class UPnPCalculator:                                                    1
                                                                         2
    from easy.upnp_utils import *                                        3
                                                                         4
    device_name = "Calculator"                                           5
    service_name = "Operations"                                          6
                                                                         7
    @action("addition")                                                  8
    @input("a:int", "b:int:")                                            9
    @output("result:int")                                                10
    def add(self, *args, **kwargs):                                      11
        a = int(args[0])                                                 12
        b = int(args[1])                                                 13
        return {'addResponse': {"result":a+b}}                           14
                                                                         15
from easy import upnp                                                    16
upnp_calc = UPnPCalculator()                                             17
upnp.start(upnp_calc)                                                    18
```

Listagem 4: Implementação de dispositivo UPnP com biblioteca Easy.

4.5 VoIP

O módulo VoIP fornece métodos simples para desenvolver aplicações com voz sobre IP, provendo funções para realizar ou aguardar chamadas VoIP. Este módulo é baseado no projeto Telepathy [21], uma plataforma usada para fazer aplicações de comunicações interpessoais, tais como mensagens instantâneas, voz sobre IP e videoconferência.

A Listagem 5 ilustra um exemplo de como usar o módulo VoIP da Easy. Para fazer uma chamada VoIP, o usuário invoca o método de chamada passando como parâmetros o *login* VoIP, a senha e o identificador da conta que deseja se comunicar (Linha 2). O gerenciador de conexões, bem como o protocolo usado, é encapsulado pelo módulo.

```
1 from easy import voip
2 voip.call('myself@voip.com', 'abcde', '1000')
```

Listagem 5: Fazendo chamadas VoIP com biblioteca Easy.

4.6 E-mail

O módulo E-mail é composto por funções para o desenvolvimento de aplicações para correio eletrônico. Possui fácil manipulação, exigindo apenas informar o servidor SMTP, *login* e senha para se conectar. Este módulo encapsula a biblioteca smtplib [22], ocultando detalhes de baixo nível do protocolo MIME [23]. Todas as funcionalidades básicas da composição de mensagem de correio eletrônico são suportadas: copia para destinatário (*cc*), cópia oculta (*bcc*) e arquivos em anexo.

Um exemplo da utilização deste módulo é visto na Listagem 6. Na linha 2 ocorre a conexão com o servidor de e-mail, informando o *login*, senha e servidor. Na linha 3 uma foto é anexada e na 4 o e-mail é enviado para o destinatário.

```
1 from easy import email
2 email.connect('login', 'password', 'servidor_smtp')
3 email.attach('myphoto.jpg')
4 email.send_mail('destino', 'titulo', 'mensagem vai neste campo!')
```

Listagem 6: Enviando e-mails com a biblioteca Easy.

4.7 Outros módulos

A biblioteca Easy define ainda módulos para acessar outros recursos da Maemo como Bluetooth, contatos, receptor GPS e interface gráfica. Estes módulos são descrito abaixo.

- **Bluetooth:** define métodos para o uso de Bluetooth no Maemo. Fornece funções para descoberta de dispositivos e de serviços, além de transferência de arquivos. Este módulo utiliza LightBlue [24], uma API baseada em Python que permite um acesso simplificado ao Bluetooth;

- **Contatos:** permite a manipulação de contatos armazenados no dispositivo Maemo. É possível adicionar, remover e alterar contatos, utilizando para tanto uma camada de abstração que se situa acima da biblioteca Python-Evolution;

- **GPS:** possibilita o acesso às informações fornecidas por um receptor GPS pareado por Bluetooth com o dispositivo Maemo, para a versão N800, ou diretamente pelo GPS embutido em dispositivos que o possuam, tal como o N810. As informações que podem ser conseguidas incluem: latitude, longitude, altitude e velocidade;

- **UI:** simplifica o processo de desenvolvimento de interfaces gráficas baseadas em GTK+ utilizando o projeto Eagle, uma biblioteca Python liberada sob licença LGPL que ajuda no desenvolvimento de GUIs, fornecendo uma camada de abstração para facilitar a criação de interfaces gráficas [25].

5 Comparação Entre a Biblioteca Easy e Outras Abordagens

O escopo da biblioteca Easy não é resumido às implementações de *scripts* simples como os mostrados na seção anterior. Projetos mais elaborados, a exemplo de *players* de áudio e gravadores de vídeo com interface gráfica, podem também ser construídos. A Figura 4 e Figura 5 ilustra tais aplicações, implementados por completo utilizando apenas a biblioteca Easy, tornando suas elaborações mais simplificadas. Como resultado final, os desenvolvedores podem obter um código fonte mais claro e menor, o que facilita a manutenção e oferece melhor suporte para a evolução quando novas funcionalidades são necessárias.

Figura 4: *Player* de áudio.

Para ilustrar estas vantagens, esta seção provê algumas comparações úteis entre a biblioteca Easy e outras abordagens comumente utilizadas no desenvolvimento de *software* para dispositivos móveis.

5.1 Gravador de áudio

Na Listagem 7 é ilustrada uma aplicação que grava o áudio capturado pelo microfone do dispositivo e salva-o em um arquivo no formato *wave*. Este aplicativo é implementado com linguagem de programação C e faz uso da biblioteca GStreamer (gst.h) diretamente para lidar com o áudio capturado. O fluxo de áudio é captado pelo elemento *dsppcmsrc* (Linha 11),

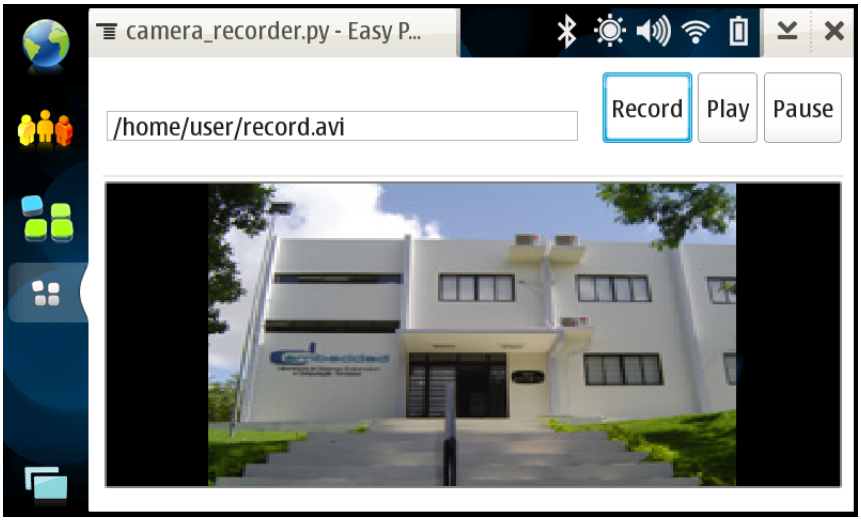

Figura 5: Gravador de vídeo.

convertendo o áudio recebido em dados brutos (*raw*). Posteriormente, os dados são codificados no formato *wave* através do elemento *wavenc* (Linha 12) e registrados em arquivo pelo elemento *gnomevfssink* (Linha 13).

Um *pipeline* (Linha 10) é necessário para agrupar todos esses elementos, que devem ser conectados corretamente para funcionar da maneira esperada (Linha 22). O nome do arquivo é definido alterando a propriedade *"local"* do elemento *gnomevfssink* (Linha 20). Para iniciar a gravação, é necessário alterar o estado do *pipeline* para *GST_STATE_PLAYING* (Linha 23). O código fonte desta listagem é baseado em um tutorial acessível em [26].

```
#include <gst/gst.h>                                                    1
#include <time.h>                                                       2
                                                                        3
int main(int argc,char *argv[])                                         4
{                                                                       5
  gst_init(&argc,&argv);                                                6
                                                                        7
  GstElement *pipeline, *src, *enc, *sink;                              8
                                                                        9
  pipeline=gst_pipeline_new("audio-recorder");                          10
  src=gst_element_factory_make("dsppcmsrc","src");                      11
  enc=gst_element_factory_make("wavenc","encoder");                     12
  sink=gst_element_factory_make("gnomevfssink","filesink");             13
                                                                        14
  if(!pipeline || !src || !enc || !sink){                               15
    g_print("Could not create element!\n");                            16
    return -1;                                                          17
  }                                                                     18
```

```
                                                                    19
    g_object_set(G_OBJECT(sink),"location","audio.wav",NULL);       20
    gst_bin_add_many(GST_BIN(pipeline),src,enc,sink,NULL);          21
    gst_element_link_many(src,enc,sink,NULL);                       22
    gst_element_set_state(pipeline,GST_STATE_PLAYING);              23
    sleep(10);                                                      24
                                                                    25
    gst_element_set_state(pipeline,GST_STATE_NULL);                 26
    gst_object_unref(GST_OBJECT(pipeline));                         27
                                                                    28
    return 0;                                                       29
}                                                                   30
```

Listagem 7: Gravador de áudio implementado em C com GStreamer.

A Listagem 8 ilustra o mesmo gravador de áudio, desenvolvido, entretanto, utilizando a biblioteca Easy. É significante o nível de abstração da plataforma GStreamer utilizando a Easy. *Bins*, estados, *pipelines* e conexões são todos encapsulados. Como resultado, o desenvolvedor não tem necessidade de se preocupar com questões mais complexas da utilização do GStreamer, ele só precisa importar o módulo de áudio (Linha 1) e chamar o método de gravação, passando o nome do arquivo e o tempo total como parâmetros (Linha 2).

```
from easy import audio                  1
audio.record('audio.wav', 10)           2
```

Listagem 8: Gravador de áudio implementado com biblioteca Easy.

5.2 *Player* de vídeo com GUI

A Listagem 9 apresenta o código fonte de um gravador de vídeo em Python. Utiliza-se o GTK+ para construir a interface gráfica e o GStreamer para implementar as funcionalidades multimídia. A implementação da GUI requer uma janela principal (Linha 6), que funciona como um recipiente para adicionar outros elementos, chamados de *widgets*. Entre estes elementos, há botões para iniciar e parar o vídeo (Linha 12 e 14), um campo de texto onde o usuário pode especificar a localização e nome do vídeo (Linha 10) e também um elemento especial, o *DrawingArea*, que é utilizado para mostrar o vídeo na janela (Linha 16).

O tocante à multimídia é tratado pelo elemento *playbin* (Linha 25) e pelo *bin*, que converte o vídeo capturado para o formato *ffmpeg* (Linhas 31 a 39). Depois disso, o vídeo é exibido no tela através do *xvimagesink* (Linha 42). Além de exigir um significativo esforço de programação para implementar a

interface gráfica do usuário (considerando a simplicidade da aplicação proposta), o desenvolvedor deve ter conhecimento sobre bases do GStreamer, como *bins*, *pads* e *signals* para que o aplicativo funcione corretamente.

```
import pygtk, gtk, pygst, gst, hildon

class Video_Player:

 def __init__(self):
  window = hildon.Window()
  window.connect("destroy", gtk.main_quit)
  vbox = gtk.VBox()
  hbox = gtk.HBox()
  self.entry = gtk.Entry()
  self.entry.set_text("/home/user/video.avi")
  play_button = gtk.Button("Play")
  play_button.connect("clicked", self.play)
  stop_button = gtk.Button("Stop")
  stop_button.connect("clicked", self.stop)
  self.movie_window = gtk.DrawingArea()
  window.add(vbox)
  vbox.pack_start(hbox, False)
  hbox.add(self.entry)
  hbox.pack_start(play_button, False)
  hbox.pack_start(stop_button, False)
  vbox.add(self.movie_window)
  window.show_all()

  self.player = gst.element_factory_make('playbin', 'pb')
  bus = self.player.get_bus()
  bus.add_signal_watch()
  bus.enable_sync_message_emission()
  bus.connect('message', self.on_message)

  self.bin = gst.Bin('bin')
  videoscale = gst.element_factory_make('videoscale')
  self.bin.add(videoscale)
  pad = videoscale.get_pad('sink')
  ghostpad = gst.GhostPad('sink', pad)
  self.bin.add_pad(ghostpad)
  videoscale.set_property('method', 1)
  conv = gst.element_factory_make('ffmpegcolorspace', 'conv');

  self.bin.add(conv)
  self.videosink = gst.element_factory_make('xvimagesink')
  self.bin.add(self.videosink)
  gst.element_link_many(videoscale, conv, self.videosink)
  self.player.set_property('video-sink',self.bin)

```

```
  def play(self, widget):                                                    46
    self.videosink.set_xwindow_id(self.movie_window.window.xid)              47
    self.player.set_property('uri','file://'+self.entry.get_text())          48
    self.player.set_state(gst.STATE_PLAYING)                                 49
                                                                             50
  def stop(self, widget):                                                    51
    self.player.set_state(gst.STATE_NULL)                                    52
                                                                             53
  def on_message(self, bus, message):                                        54
    type = message.type                                                      55
    if type == gst.MESSAGE_EOS:                                              56
    self.player.set_state(gst.STATE_NULL)                                    57
                                                                             58
Video_Player()                                                               59
gtk.gdk.threads_init()                                                       60
gtk.main()                                                                   61
```

Listagem 9: *Player* de vídeo implementado em Python com GTK+ e GStreamer.

É notável a redução do tamanho do código, assim como a complexidade envolvida quando compara-se esta implementação com a feita utilizando a Easy (Listagem 10). A GUI é implementada com 5 linhas de código (Linhas 9 a 13), enquanto que as funcionalidades multimídia são implementadas com mais 5 (Linhas 3 a 7).

```
from easy import camera, ui                                                  1
                                                                             2
def play(app, button):                                                       3
 camera.play(filename=app['name'])                                           4
                                                                             5
def stop(app, button):                                                       6
 camera.stop_recording()                                                     7
                                                                             8
app=ui.App(title='Easy Player',                                              9
 top=(ui.Entry(id='name', value='./video.avi'),                              10
      ui.Button(id='play', label='Play', callback=play),                     11
      ui.Button(id='stop', label='Stop Record', callback=stop)),             12
 center=ui.XWindow(id='xw'))                                                 13
                                                                             14
camera.set_window_id(app['xw'].get_window_id())                              15
ui.run()                                                                     16
```

Listagem 10: *Player* de vídeo implementado com a biblioteca Easy.

5.3 Sintonizador de rádio FM

Um dos métodos para utilizar o chip sintonizador de rádio FM dos dispositivo Maemo é através da Video4Linux [27], uma biblioteca criada para lidar com vídeo para plataformas Linux. *Webcams*, sintonizadores de TV e outros dispositivos também são suportados pela Video4Linux. Ela é altamente integrada e dependente do *kernel* do Linux.

A Listagem 11 ilustra um exemplo de como utilizar a Video4Linux para sintonizar o chip de rádio FM em estações disponíveis. O código é implementado em linguagem de programação Python e acessa *drivers* do *kernel* para controlar o chip.

Uma vez que a Video4Linux usa função de nível muito baixo, especifica-se definições para o *kernel* do Linux através de chamadas de sistema do tipo funções *ioctl* (Linhas 3 a 12). O símbolos *_IOW*, *_IOR* e *_IOWR* (Linhas 14 a 19) definem macros em linguagem C utilizadas para acessar e controlar o *driver* do sintonizador.

O chip é abstraído por um *device* que pode ser manipulado através do arquivo */dev/radio* (Linha 35). O rádio é sintonizado em 107,6 MHz (Linha 44) chamando a função *fcntl.ioctl*, passando como parâmetros variáveis não intuitivas, como a estrutura *data* (Linha 42) ou a macro *_VIDIOC_S_FREQUENCY* (Linha 25). Este código fonte é baseado em um exemplo disponível em [28].

```
import os, struct, fcntl                                                    1
                                                                            2
_IOC_NRBITS = 8                                                             3
_IOC_TYPEBITS = 8                                                           4
_IOC_SIZEBITS = 14                                                          5
_IOC_DIRBITS = 2                                                            6
_IOC_NRSHIFT = 0                                                            7
_IOC_TYPESHIFT = _IOC_NRSHIFT + _IOC_NRBITS                                 8
_IOC_SIZESHIFT = _IOC_TYPESHIFT + _IOC_TYPEBITS                             9
_IOC_DIRSHIFT = _IOC_SIZESHIFT + _IOC_SIZEBITS                              10
_IOC_WRITE = 1                                                              11
_IOC_READ = 2                                                               12
                                                                            13
_IOC = lambda d,t,nr,size: (d<< _IOC_DIRSHIFT) |                            14
       (ord(t)<<_IOC_TYPESHIFT)|(nr<<_IOC_NRSHIFT) |                        15
       (size << _IOC_SIZESHIFT)                                             16
_IOW = lambda t,nr,size: _IOC(_IOC_WRITE, t, nr, size)                      17
_IOR = lambda t,nr,size: _IOC(_IOC_READ, t, nr, size)                       18
_IOWR = lambda t,nr,size: _IOC(_IOC_READ | _IOC_WRITE,t,nr,size)            19
                                                                            20
_VIDIOC_G_TUNER = _IOWR('V', 29, 84)                                        21
_VIDIOC_G_AUDIO = _IOR ('V', 33, 52)                                        22
_VIDIOC_S_AUDIO = _IOW ('V', 34, 52)                                        23
_VIDIOC_G_FREQUENCY = _IOWR('V', 56, 44)                                    24
```

```
_VIDIOC_S_FREQUENCY = _IOW ('V', 57, 44)
_VIDIOC_G_CTRL = _IOWR('V', 27, 8)
_VIDIOC_S_CTRL = _IOWR('V', 28, 8)
_V4L2_CTRL_CLASS_USER = 0x00980000
_V4L2_CID_BASE = _V4L2_CTRL_CLASS_USER | 0x900
_V4L2_CID_AUDIO_MUTE = _V4L2_CID_BASE + 9
_SOUND_MIXER_FMRADIO = 0x06
_SOUND_MIXER_READ = 0x80044D00
_SOUND_MIXER_WRITE = 0xC0044D00

radio = os.open('/dev/radio', os.O_RDONLY)
info = struct.pack('84x')
data = fcntl.ioctl(radio, _VIDIOC_G_TUNER, info)
fields = struct.unpack("L32sLLLLLLl14L", data)
tuner_index = fields[0]
tuner_type = fields[2]

data=struct.pack('LLL8L',tuner_index, tuner_type, 107600*16,
                 0, 0, 0, 0, 0, 0, 0, 0)
fcntl.ioctl(radio, _VIDIOC_S_FREQUENCY, data)
data = struct.pack('Ll', _V4L2_CID_AUDIO_MUTE, 0)
fcntl.ioctl(radio, _VIDIOC_S_CTRL, data)
mixer_fd = os.open('/dev/mixer', os.O_RDONLY)
data = struct.pack('bb', 50, 50)
fcntl.ioctl(mixer, _SOUND_MIXER_WRITE |
            _SOUND_MIXER_FMRADIO, data)
os.close(mixer_fd)

data = struct.pack('Ll', _V4L2_CID_AUDIO_MUTE, 1)
fcntl.ioctl(radio, _VIDIOC_S_CTRL, data)
os.close(radio)
```

Listagem 11: Sintonizador de rádio FM implementado em Python com Video4Linux.

Em comparação, a Listagem 12 ilustra o código fonte para a implementação do aplicativo de sintonia de rádio FM desenvolvido com a biblioteca Easy. Detalhes sobre o funcionamento e manuseio da API Video4Linux não são mais requisitos para poder sintonizar e ouvir uma estação de rádio. O usuário precisa apenas iniciar o chip de rádio (Linha 2) e depois ajustá-lo para a frequência desejada (Linha 3).

```
from easy import radio
radio.start()
radio.tune(107600)
```

Listagem 12: Sintonizador de rádio FM implementado com a Easy.

6 Considerações Finais

O presente Trabalho de Conclusão de Curso apresentou a Easy, uma biblioteca baseada em Python para o desenvolvimento de aplicações para dispositivos móveis que utilizam a plataforma Linux. Deu-se ênfase a sua arquitetura, API e módulos, mostrando suas vantagens quando comparada com outras soluções para a manipulação de áudio, vídeo, chip de rádio e outros recursos dos dispositivos Maemo. Para tanto, foram ilustrados alguns exemplos de implementações mais complexas utilizando a Easy e as comparando com tecnologias amplamente utilizadas no desenvolvimento para dispositivos móveis baseados em Linux.

Foi mostrado que a biblioteca apresentada torna mais simples o desenvolvimento de programas para dispositivos Maemo, permitindo a construção de *software* através de uma camada de abstração.

Como conclusão, é importante ressaltar que o uso de APIs e bibliotecas como a Easy são relevantes para a manutenção do código fonte e a sua evolução quando novas funcionalidades devem ser adicionadas, uma vez que traz mais clareza e simplicidade ao desenvolvimento de aplicações.

É importante também ressaltar que encapsular tecnologias complexas ou de mais baixo nível aumenta a produtividade do desenvolvedor, porém limita o acesso as funcionalidades providas pela camada de abstração, cabendo ao programador decidir se o que ela fornece é suficiente para seu aplicativo ou se deve-se utilizar uma abstração de nível mais baixo para atender aos requisitos necessários.

Atualmente a biblioteca Easy está disponível em sua versão 0.4 e pode ser executado na plataforma Maemo 4.x. Informações sobre contribuições, *bugs*, listas de discussão e notícias podem ser obtidas no site do projeto.

Como trabalhos futuros que podem ser desenvolvidos para a biblioteca, pode-se citar: módulo de mensagem instantânea utilizando a plataforma Telepathy, interface gráfica com usuário utilizando Qt [29], suporte a recepção de TV digital através do Ginga [30], além de portar a biblioteca para plataformas i386 e telefones móveis, permitindo que se tenha essa mesma camada de abstração para *desktop* e celulares.

Referências

[1] G. H. Forman and J. Zahorjan, "The challenges of mobile computing," *Computer*, vol. 27, no. 4, pp. 38–47, 1994.

[2] Maemo, "Maemo is the application development platform for internet tablets," http://maemo.org, 2010, acessado em Janeiro de 2010.

[3] GNU, "Gnu lesser general public license," http://www.gnu.org/copyleft/lesser.html, 2010, acessado em Janeiro de 2010.

[4] M. Lutz and D. Ascher, *Learning Python*, 2nd ed. O'Reilly, 2004.

[5] "Percomp," http://wiki.percomp.org, 2010, acessado em Janeiro de 2010.

[6] "Embedded systems and pervasive computing lab," http://embedded.ufcg.edu.br, 2010, acessado em Janeiro de 2010.

[7] XP, "Extreme programming: A gentle introduction," http://extremeprogramming.org, 2010, acessado em Janeiro de 2010.

[8] K. Beck, *Extreme Programming Explained: Embrace Change*, 2nd ed. Addison - Wesley, 2000.

[9] GForge, "Gforge collaborative development environment," http://gforge.org, 2010, acessado em Janeiro de 2010.

[10] A. Krause, "Foundations of GTK+ Development," http://gtk.org, 2010, acessado em Janeiro de 2010.

[11] D-Bus, "D-Bus," http://freedesktop.org/wiki/Software/dbus, 2010, acessado em Janeiro de 2010.

[12] Hildon, "Hildon," http://live.gnome.org/Hildon, 2010, acessado em Janeiro de 2010.

[13] PyMaemo, "Python for maemo," http://pymaemo.garage.maemo.org, 2010, acessado em Janeiro de 2010.

[14] BlueZ, "Bluez," http://bluez.org/, 2010, acessado em Janeiro de 2010.

[15] Easy, "Easy project," http://easy.garage.maemo.org/, 2010, acessado em Janeiro de 2010.

[16] GStreamer, "Gstreamer: open source multimedia framework," http://gstreamer.freedesktop.org/, 2010, acessado em Janeiro de 2010.

[17] CInLUG, "Apresentando o gstreamer," http://cin.ufpe.br/ cinlug/wiki/index.php/Apresentando_o_GStreamer, 2010, acessado em Janeiro de 2010.

[18] UPnP, "Upnp forum," http://upnp.org/, 2010, acessado em Janeiro de 2010.

[19] A. Guedes, D. Santos, J. do Nascimento, L. Sales, A. Perkusich, and H. Almeida, "Set your multimedia application free with brisa framework: An open source upnp implementation for resource limited devices," *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pp. 1257–1258, 10-12 Jan. 2008.

[20] C. Winter, "Pep 3107 – function annotations," http://www.python.org/dev/peps/pep-3107/, 2010, acessado em Janeiro de 2010.

[21] Collabora, "Telepathy," http://telepathy.freedesktop.org/wiki, 2010, acessado em Janeiro de 2010.

[22] smtplib, "Smtp protocol client," http://docs.python.org/lib/module-smtplib.html, 2010, acessado em Janeiro de 2010.

[23] N. Borenstein and N. Freed, "Mime: Multipurpose internet mail extensions," Technical Report RFC 1341, 1992, iETF Network Working Group, http://www.ietf.org/rfc/rfc1341.txt.

[24] LightBlue, "Lightblue: a cross-platform python bluetooth api," http://lightblue.sourceforge.net/, 2010, acessado em Janeiro de 2010.

[25] Eagle, "Eagle is a python library to help GUI development," http://gustavobarbieri.com.br/eagle/, 2010, acessado em Janeiro de 2010.

[26] W. Taymans, S. Baker, A. Wingo, R. S. Bultje, and S. Kost, "Gstreamer application development manual (0.10.23.1)," http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/, 2010, acessado em Janeiro de 2010.

[27] Video4Linux, "Video4linux," linuxtv.org/v4lwiki/index.php/Main_Page, 2010, acessado em Janeiro de 2010.

[28] maemo, "How to control the fm tuner chip in the n800," http://wiki.maemo.org/Programming_FM_radio, 2010, acessado em Janeiro de 2010.

[29] Qt, "Qt - a cross-platform application and ui framework," http://qt.nokia.com/products/, 2010, acessado em Janeiro de 2010.

[30] Ginga, "Ginga digital tv middleware specification," http://www.ginga.org.br/, 2010, acessado em Janeiro de 2010.