Survey on Multimedia Technologies for Mobile Learning Applications

Paul POCATILU, Cătălin BOJA
Economic Informatics Department,
Academy of Economic Studies, Bucharest, Romania
ppaul@ase.ro, catalin.boja@ie.ase.ro

Mobile technologies are developing very fast. This paper presents a survey on multimedia technologies for mobile learning applications, focusing on multimedia programming techniques for Windows Mobile, Symbian, and Java ME.
Keywords: *multimedia, mobile devices, mobile applications, mobile learning*

1 Introduction

Mobile learning is implemented using different technologies and platforms. Each implementation has specific characteristics in terms of user interface and content and influences the quality management process.

The software required for mobile learning process is a Web browser or a specific application, that can be standalone or a client application.

Web-based platforms are very common in mobile learning process. Figure 1 depicts the architecture of this platform.

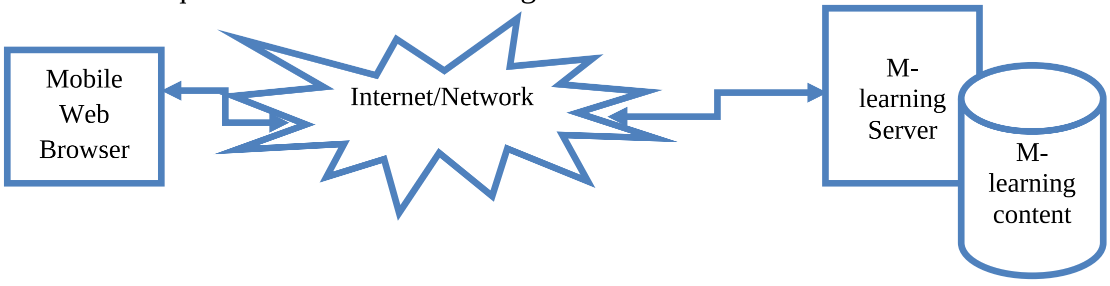

Fig. 1. Web-based mobile learning platforms architecture

The advantages of using Web-based mobile learning platform are:

- User's minimal effort to set up the application;
- The user doesn't need to learn how to use a new application; the client is a simple mobile Web browser (like Pocket Internet Explorer of Opera Mobile);
- E-learning content can be easily updated on the server.

There are also some disadvantages:

- Not so rich user interface (browser-based);
- Multimedia content is limited;
- Possible additional costs for traffic usage.

Distributed platforms have a similar architecture as Web-based platforms, but the client application is a rich application and not a simple mobile Web browser. The advantages of this platform are:

- Rich user interface;
- Support for multimedia content;
- E-learning content can be easily updated on the server;

There are also some disadvantages:

- The user need to install and setup the client application;
- The user have to learn how to use the application;
- Possible additional costs for traffic usage.

Standalone applications do not require network access. They are similar to the client applications for distributed platforms. The main disadvantage is that the update of content needs access to device.

Errors and applications failures influence the learning process. Distributed applications and Web-based requires more test because of network access. The test plans need to include use cases on various hardware platforms, operating systems, networks bandwidths in order to cover a large number of scenarios.

Almost all mobile operating systems have support for multimedia applications development. Some technologies and libraries exist in desktop versions (like Open GL, DirectX etc.), other are specific to some operating system [2], [3], [4], [5].

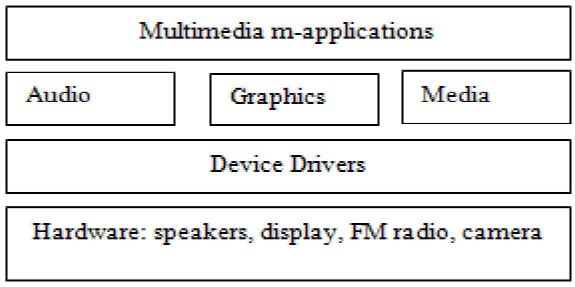
Multimedia m-applications
Audio
Graphics
Media
Device Drivers
Hardware: speakers, display, FM radio, camera

Fig. 2 Multimedia system [1]

Multimedia m-applications (figure 2) access audio, graphics and media API through operating system or third party libraries. These API communicate with multimedia related hardware through device drivers.
The display resolution, audio and video system, 3D support, and operating system influence the multimedia application development for mobile devices.
The audio and video resources involve playback and recording capabilities.

2 Java Micro Edition (Java ME)

Java Micro Edition platform is intended for all mobile devices that have limited hardware and software resources. In order to allow the development of application for these devices, this platform has limited framework support comparing to its big brother, Java Standard Development Kit. Despite the early characteristics of devices, present and future mobile technologies allow software developers to design and implement rich multimedia content. In order to support this kind of resources, Java ME platform includes a package, the Mobile Media API (MMAPI), [6-7], that is flexible enough to run with many types of protocols and multimedia formats, as MP3, MIDI for audio, MPEG-4 for video and JPEG, PNG, BMP, SVG for pictures. The MMAPI framework has been designed to run on any J2ME-based virtual machine, including the CDC and CLDC virtual machines. This standard Java specification has been defined by the Java Community Process (JCP) in JSR 135, [7].
At the core of the MMAPI package there are two concepts that describe multimedia processing:

- *Protocol handling* refers to the process of reading data from a source, as local files, from a streaming server or from device components, into a media-processing system;
- *Content handling* refers to the process of decoding the media data and rendering it to an output interface.

Figure 3 describes the three high-level object types, Data Source, Player and Manager, that the API provides, in order to implement the two concepts.

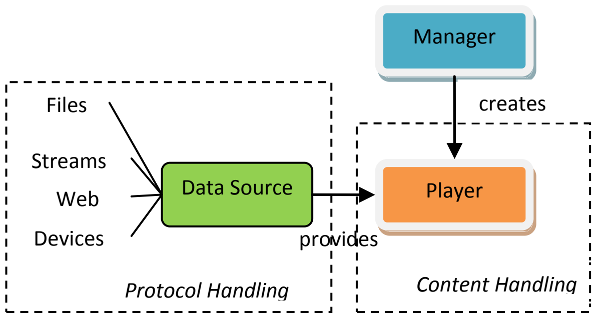

Fig. 3 The MMAPI Architecture.

The relation between these components is based on the fact that the *Manager* object is used to create *Player* entities from *DataSources* or *Input Streams*. Once created, *Player* class provides methods that will give control over the multimedia resource.

2.1 Audio

Different audio formats are managed by the MMAPI using:

- *Manager playTone()* method that generates simple tones, characterized by frequency and duration; this is the simple audio media format that can be managed and its role is to provide basic audio capabilities to mobile devices; the following sequence play a tone for 2000 milliseconds at volume 50

```
...
try {
  Manager.playTone()(ToneControl.C4,
4000, 100);
}
```

```
catch(MediaException me) {
}
...
```

- *Player* object, created to access audio files stored on the device or on an Web server; the next code sequence gets an MP3 file over the network and plays it:

```
try{
  String mp3="http://myserver.com/" +
"media/music.mp3";
  Player p = Manager.createPlayer(mp3);
  p.start();
}
catch (IOException x) {
  ...
} catch (MediaException x) {
  ...
}
```

- InputStreams and Player classes to access device stored audio resources, as MP3 files; media can be stored as a resource within the JAR file or in a RecordStore:

```
//from JAR file
InputStream is1 =
getClass().getResourceAsStream("m.mp3");
Player p1 = Manager.createPlayer(is,
"audio/mpeg");
P1.start();

//from RMS
InputStream is2 =
  new ByteArrayInputStream(
  myRecordStore.getRecord(id));
Player p2 = Manager.createPlayer(
  is2, "audio/mpeg");
p2.start();
```

2.2 Graphics and Animation

The concept of graphics and animation management is direct related to the framework elements that allow developers to control device display. One core element of the J2ME platform used in this scope is the *Canvas* class, [9], defined in the *javax.microedition.lcdui* package, that lets an application draw screens using the low-level user-interface API.

The next code sequence is used to draw an image on the device display, creating a splash screen.

```
protected void paint(Graphics g) {
    Image logo;
     try{
            logo    =    Image.createImage(
"/images/logo.jpg");
```

```
           g.setColor(0,0, 0);
           g.fillRect(0, 0,
           g.drawImage(logo, 0, 0, 0);
           g.setColor(200,0,0);

g.setFont(Font.getFont(Font.FACE_SYSTEM,
Font.STYLE_BOLD,Font.SIZE_LARGE));

         g.drawString("My Agenda v1.0", 5,
30,0);
         g.setColor(0,0,0);
         g.drawLine(5,50,  this.getWidth(),
50);
        }
        catch(Exception err){
              ...
        }
    }
```

The image is load from the JAR and figure 4 shows the result.

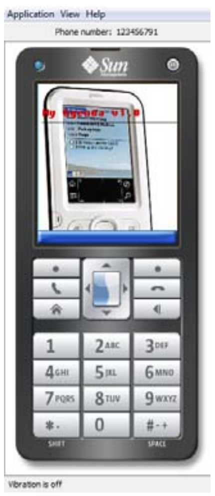

Fig. 4 Drawing image on mobile device display

The limitations of *Canvas* class is generated by a lack of control over when a canvas repaints itself or over how quickly key and pointer events are received by the object. In order to overcome this drawback and to provide support for software gaming industry, it was designed the *GameCanvas* class, that extends the *Canvas* class.

The role of the *GameCanvas* class is to provide efficient resource management for animation and event handling. Basic elements of an animation and their classes are:

- animation background implemented by the *TiledLayer* class; the background is viewed as a grid with custom dimensions; this provides an efficient and simple way to manage interactions between game components and dynamic generation of the environment; based on few resources, as the background elements in figure 5, resources are multiplied and put in the grid; the following code sequence, creates a 6x7 grid and populates it with 32x32 pixels graphical elements, that were extracted from a PNG image

```
 Image imgBack, imgButterfly;
 int background[][]= {
        {0,4,4,0,0,0,4},
        {4,0,3,0,0,4,4},
        {0,0,0,4,0,4,0},
        {1,0,2,2,0,1,1},
        {2,1,0,0,2,1,1},
        {0,0,1,1,2,1,0}
    };
  TiledLayer tlBackground;
  imgBack        =     Image.createImage(
"/images/bk.png");
  tlBackground   =   new   TiledLayer(6,7,
imgBack, 32, 32);
  for(int i=0;i<6;i++)
     for(int j = 0;j<7; j++)
 background.setCell(j, i, fundal[i][j]);
```

- animated elements, defined as *Sprite* objects; this class is designed to manage animation sequences independently from the main thread; the next sequence creates a Sprite object based on butterfly.png, described in figure 5; the sprite animates the butterfly by changing in a predefined sequence the four images;

```
Image    imgButterfly=Image.createImage("
/images/fluture.png");
Sprite    sButterfly;    =    new    Sprite(
imgFluture, 32,32);

int flight[] = {1, 2, 3, 3, 2, 1, 0};

sButterfly.setFrameSequence(flight);
sButterfly.setRefPixelPosition(8, 8);
```

- *LayerManager* class that provides game developer methods to manage animation elements; usign a *LayerManager* object, animation elements are interconnected and it is decided how and in which context they are made visible; next sequence puts together, in a GameCanvas, both the background and the sprite;

```
LayerManager lm = new LayerManager();
lm.append(sButterfly);
sFluture.setPosition(100,150);
lm.append(tlBackground);
lm.paint(this.getGraphics(), 0, 0);
lm.setViewWindow(0,0,latime,inaltime-
20);
flushGraphics();
```

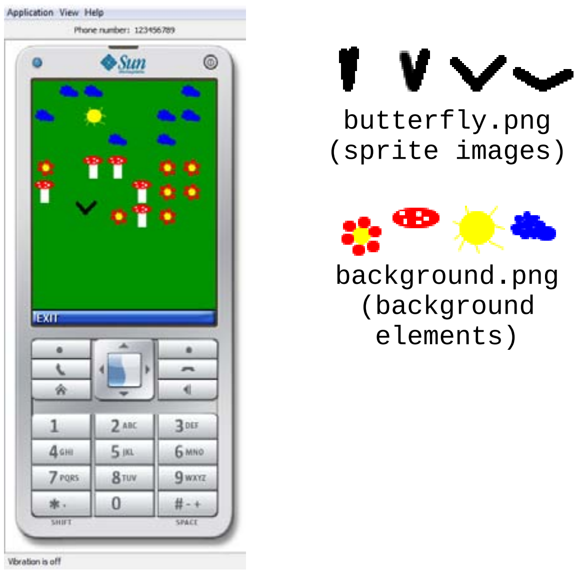

Fig. 5 Game application in J2ME and its resources

On advanced mobile devices that incorporate a camera, MMAPI includes support for accessing it from code. For that the player is created using a *capture://video* locator. By default, the image format used to store the camera capture is PNG. The next code sequence allows the user to capture an image by camera:

```
try {
player =
Manager.createPlayer("capture://video");
player.realize();
videoControl = (VideoControl)
(player.getControl("VideoControl"));
if (videoControl != null) {
videoControl.initDisplayMode(VideoContro
l.USE_DIRECT_VIDEO, this);
}
}
catch (IOException ioe) {
...
}
catch (MediaException me) {
...
}
catch (SecurityException se) {
...
}
if(player!=null){
try{
```

```
byte[] pngImage =
videoControl.getSnapshot(null);
image=Image.createImage(pngImage,0,pngIm
age.length);
}catch(MediaException me){
...
}
}
```

The key elements from the above sequence are:

- create Player object

```
player =
Manager.createPlayer("capture://video");
```

- *getControl* over the resource

```
videoControl = (VideoControl)
(player.getControl("VideoControl"));
```

- getSnapShot to get the captured image.

```
byte[]            pngImage             =
videoControl.getSnapshot(null);
```

2.3 Video

J2ME provides video management with the *VideoControl* interface. A reference to the video resource is provided by the *Player* class and its *getControl()* method.
The below sequence plays a video file on mobile device:

```
...
Player p;
VideoControl vc;
try {
   p = Manager.createPlayer("http://
myserver.com/movie.mpg");
   p.realize();
   // get video control
   vc = (VideoControl)p.getControl("
VideoControl");

   //get a GUI to display the video
   Item videoItem = (Item)video.initDisp
layMode(
   VideoControl.USE_GUI_PRIMITIVE,null);

  //append the GUI to a form
   videoForm.append(videoItem);
   p.start();
}
catch(IOException ioe) {
}
catch(MediaException me) {
}
...
```

3 Windows Mobile

DirectShow is available starting with Windows Mobile 5. This is technology is based on COM. It is highly recommended to play the media files in a separate thread.

3.1 Audio

Audio files can be played using:

- standard API function,
- a shell function (Windows Mobile 6 and up only)
- DirectShow
- third-party API

The *PlaySound* function is limited to PCM WAV formats. This call plays the specified file. If an error occurs, bResult will be set to FALSE:

```
BOOL bResult =
PlaySound(L"\\Windows\\Alarm5.wav",
NULL, SND_FILENAME);
```

For other formats, *SndPlaySync* function can be used. This function is related to Windows Mobile 6 shell. A sample call of this function is:

```
HRESULT hr =
SndPlaySync(L"\\Windows\\Ring-
WindowsMobile .wma", 0);
```

DirectShow can be used to play audio files. Depending on installed codecs, various formats can be played. The following code is used to play an audio file using DirectShow:

```
CComPtr<IGraphBuilder>   pGraphBuilder;
CComPtr<IMediaControl>   pMediaControl;
CComPtr<IMediaEvent>     pMediaEvent;

LPCWSTR fisAudio = L"\\My
Documents\\audio2.wav";

HRESULT hr = S_OK;
CoInitialize(NULL);

//Here we create the filter graph
manager
hr =
pGraphBuilder.CoCreateInstance(CLSID_Fil
terGraph);

if (hr != S_OK)
{
       // error handling
       return;
}

hr = pGraphBuilder->QueryInterface(
IID_IMediaControl, (void
**)&pMediaControl);
hr = pGraphBuilder-> QueryInterface(
IID_IMediaEvent, (void **)&pMediaEvent);

//Render the media file
hr = pGraphBuilder->RenderFile(fisAudio,
NULL);
```

```
if (hr == S_OK)
{
   hr = pMediaControl->Run();

   if (hr = S_OK)
   {
    long lEvCode = 0;
    pMediaEvent->WaitForCompletion(
       INFINITE, &lEvCode);
   }
}
else
{
   TCHAR temp[50];
   swprintf(temp, L"%s : %x", L"Eroare
redare audio", hr);
   MessageBox(hWnd, temp, L"Eroare!",
       MB_OK | MB_ICONERROR);
}

CoUninitialize();
```

The thread waits until the audio file ends so that this code needs to be in a separate thread.
There is a third-party API developed that allows playing sound.

3.2 Graphics and Animation

The basic drawing primitives are provided by the GDI. Usually the drawing is handled on WM_PAINT message.
The following listing demonstrates how to use them to draw shapes and text using GDI API:

```
Rectangle(hdc,r.left,r.top,r.right,r.bot
tom);
```

```
//draw a rectangle
//fill the rectangle
FillRect (hdc,&r, g_hBrush);
//set background color
SetBkColor (hdc, 0x0000ff00);
//draw text
DrawText(hdc,L"DrawText Sample", -1, &r,
DT_CENTER | DT_VCENTER);
//select another brush
SelectObject(hdc, g_hBrush);
//draw a circle
Ellipse(hdc, (r.right - r.left - 50)/2,
r.top, (r.right - r.left - 50)/2 + 50,
50);
//set text align relative to reference
//point for ExtTextOut
SetTextAlign(hdc, TA_CENTER);
//draw text
ExtTextOut(hdc, (r.right - r.left)/2,
190, 0, NULL, text, lstrlen(text),
NULL);
```

There is support for 3D graphics using Direct3D Mobile. There are limitations of Direct3D Mobile compared to Direct3D due to hardware constraints.
First, a device has to be initialized and its properties to be set. The device will be used to render the 3D graphics based on objects properties (coordinates and color). The distance to the objects, the view angle, the light, the view window and other properties can be controlled.
The following code initializes the device and the pyramid that will be rendered:

```
PresentParameters presentParams = new PresentParameters();
presentParams.Windowed = true;
presentParams.SwapEffect = SwapEffect.Discard;

device = new Device(
                0,                      // numar device
                DeviceType.Default,     // configuratia
                this,                   // fereastra parinte
                CreateFlags.None,       // parametri de creare
                presentParams);         // parametri de prezentare


// camera settings
device.Transform.View = Matrix.LookAtLH(
                new Vector3(0.0f, 0.0f, -2.0f), // camera position
                new Vector3(0.0f, 0.0f, 0.0f),  // camera focus
                new Vector3(0.0f, 0.5f, 0.0f)   // camera up
                );

// set up the projection
device.Transform.Projection =
                Matrix.PerspectiveFovLH(
                (float)Math.PI / 4,     // angle
                1.0f,                   // scaling factor
                1.0f,                   // near Z plane
                100.0f                  // further Z plane
                );
//display objects even on the hidden side
```

```
device.RenderState.CullMode = Cull.None;
//manually control the light
device.RenderState.Lighting = false;

//pyramid initialization (coordinates and color)
vertices[0] = new CustomVertex.PositionColored(0.0f, 0.0f, -0.5f,
Color.Yellow.ToArgb());
vertices[1] = new CustomVertex.PositionColored(0.5f, 0.5f, 0.0f,
Color.Green.ToArgb());
vertices[2] = new CustomVertex.PositionColored(-0.5f, 0.5f, 0.0f,
Color.Red.ToArgb());
vertices[3] = new CustomVertex.PositionColored(0.0f, 0.0f, -0.5f,
Color.Yellow.ToArgb());
vertices[4] = new CustomVertex.PositionColored(0.0f, -0.5f, 0.0f,
Color.Blue.ToArgb());
vertices[5] = new CustomVertex.PositionColored(-0.5f, 0.5f, 0.0f,
Color.Red.ToArgb());
vertices[6] = new CustomVertex.PositionColored(0.0f, 0.0f, -0.5f,
Color.Yellow.ToArgb());
vertices[7] = new CustomVertex.PositionColored(0.0f, -0.5f, 0.0f,
Color.Blue.ToArgb());
vertices[8] = new CustomVertex.PositionColored(0.5f, 0.5f, 0.0f,
Color.Green.ToArgb());
vertices[9] = new CustomVertex.PositionColored(0.0f, -0.5f, 0.0f,
Color.Blue.ToArgb());
vertices[10] = new CustomVertex.PositionColored(0.5f, 0.5f, 0.0f,
Color.Green.ToArgb());
vertices[11] = new CustomVertex.PositionColored(-0.5f, 0.5f, 0.0f,
Color.Red.ToArgb());
// initialize the buffer (3D points)
vertBuffer = new VertexBuffer(
                typeof(CustomVertex.PositionColored), // tip buffer
                vertices.Length,                       // number of points
                device,                                // device
                Usage.WriteOnly,                       // access type
                CustomVertex.PositionColored.Format,   // point types
                Pool.SystemMemory);                    // memory used

// fill the buffer with data
vertBuffer.SetData(vertices, 0, LockFlags.None);
}
```

The code that draws and rotates the pyramid is:

```
//set black background
device.Clear(ClearFlags.Target,
Color.Black, 1.0f, 0);
device.BeginScene();

// set the buffer
device.SetStreamSource(0, vertBuffer,
0);

// increase the rotation angle
unghiRotire += pasRotire;

device.Transform.World =
Matrix.RotationYawPitchRoll(0f,
unghiRotire, 0f);

//draw using triangles
device.DrawPrimitives(PrimitiveType.Tria
ngleList, 0, 3);
device.EndScene();
device.Present();
```

Figure 6 shows the capture of a mobile device screen running the Direct3D Mobile code presented in this section.

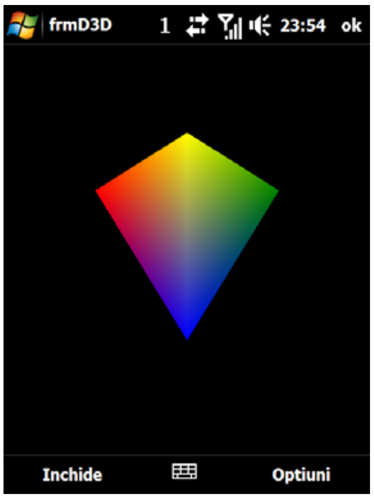

Fig. 6 Direct3D Mobile example

3.3 Video

Video can be played on Windows Mobile based

devices using DirectShow or Windows Media Player control. These methods require the use of COM technologies.
The code is similar to audio playback using DirectShow, for video playback a window is required. The following code initializes the filter graph manager and the player window and plays a video file in full screen:

```
CComPtr<IGraphBuilder>   pGraphBuilder;
CComPtr<IMediaControl>   pMediaControl;
CComPtr<IVideoWindow>    pVidWindow;
CComPtr<IMediaEvent>     pMediaEvent;

LPCWSTR fisVideo = L"\\My Documents\\video4.3gp";

HRESULT hr;

CoInitialize(NULL);

// Initialize the filter graph manager
hr = pGraphBuilder.CoCreateInstance(CLSID_FilterGraph);

if (hr != S_OK)
{
       TCHAR temp[50];
       swprintf(temp, TEXT("%s : %x"), TEXT("Eroare initializare filtru!"), hr);
       MessageBox(hWnd, temp, TEXT("Eroare!"), MB_OK | MB_ICONERROR);
       return;
}
//initialize the media control
hr = pGraphBuilder->QueryInterface(IID_IMediaControl, (void **)&pMediaControl);
//initialize the video window
hr = pGraphBuilder->QueryInterface(IID_IVideoWindow, (void **)&pVidWindow);
//initialize the media event
hr = pGraphBuilder->QueryInterface(IID_IMediaEvent, (void **)&pMediaEvent);

//video file rendering
hr = pGraphBuilder->RenderFile(fisVideo, NULL);

if (hr == S_OK)
{
       // Video window settings
       pVidWindow->put_Owner((OAHWND)hWnd);
       pVidWindow->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS);
       //Play in full screen
       pVidWindow->put_FullScreenMode(OATRUE);

       //start playing the video
       hr = pMediaControl->Run();

       if (hr == S_OK)
       {
              long lEventCode = 0;
              //wait until video end
              pMediaEvent->WaitForCompletion(INFINITE, &lEventCode);
       }

}
else
{
       TCHAR temp[50];
       swprintf(temp, TEXT("%s : %x"), TEXT("Eroare redare video"), hr);
       MessageBox(hWnd, temp, TEXT("Eroare!"), MB_OK | MB_ICONERROR);
}

CoUninitialize();
```

The video playback has to take place in a separate thread in order to keep the user interface functionally.

4 Symbian OS

4.1 Audio

The Symbian audio component allows playing and recording audio files, depending on installed codecs.

Audio files can be played using *CMdaAudioPlayerUtility* class. Applications that use it need a helper class that implements *MMdaAudioPlayerCallback* interface in order to receive notification about the player.

Audio streams can be played using the *CMdaAudioOutputStream* class.

The following code implements a player class that can be used for audio files:

```
#include <MdaAudioSamplePlayer.h>

enum TState {EIsNotReady, EIsReady, EIsPlaying };

class CAudioPlayer: public CBase, public MMdaAudioPlayerCallback
{
      void CAudioPlayerL(const TDesC& aFisier);

      TState iStare;
      CMdaAudioPlayerUtility* iPlayer;
public:
      void ConstructL(const TDesC& aFisierAudio);
      static CAudioPlayer* NewL(const TDesC& aFisierAudio);
      static CAudioPlayer* NewLC(const TDesC& aFisierAudio);
      ~CAudioPlayer();
      void Play();
      //void Stop();
      //void Pause();
      //callback functions
      void MapcInitComplete(TInt aEroare, const TTimeIntervalMicroSeconds& aDur);
      void MapcPlayComplete(TInt aEroare);
};

CAudioPlayer* CAudioPlayer::NewL(const TDesC& aFisierAudio)
{
      CAudioPlayer* self = NewLC(aFisierAudio);
      CleanupStack::Pop(self);
      return self;
}

CAudioPlayer* CAudioPlayer::NewLC(const TDesC& aFisierAudio)
{
      CAudioPlayer* self = new (ELeave) CAudioPlayer();
      CleanupStack::PushL(self);
      self->ConstructL(aFisierAudio);
      return self;
}

void CAudioPlayer::ConstructL(const TDesC& aFisierAudio)
{
      iPlayer = CMdaAudioPlayerUtility::NewFilePlayerL(aFisierAudio, *this);
}
//the audio will play if the player is ready
void CAudioPlayer::Play()
{
      if(iStare == EIsReady)
      {
            iStare = EIsPlaying;
            iPlayer->Play();
      }
}
//Not shown Stop, Pause, SetPosition, GetPosition, volume control
void CAudioPlayer::MapcPlayComplete(TInt aEroare)
{
      iStare = aEroare ? EIsNotReady : EIsReady;
}

void CAudioPlayer::MapcInitComplete(TInt aEroare, const TTimeIntervalMicroSeconds&)
{
        iStare = aEroare ? EIsNotReady : EIsReady;
}
//releas the player
CAudioPlayer::~CAudioPlayer()
{
        delete iPlayer;
        iPlayer = NULL;
}
```

The audio file can be stopped, paused and resumed. Also it is possible to position in the audio file. The following code is used to play an audio file using the class above:

```
iPlayer = CAudioPlayer::NewL(_L
("C:\\System\\Data\\audio2.wav"));

if (iPlayer)
        iPlayer->Play();
```

4.2 Graphics and Animation

Drawing is similar to other programming platforms. It is necessary a device context that will be used to draw on the display. The following code is used to draw some basic shapes and text on the application main window:

```
CWindowGc& gc = SystemGc();

gc.SetPenStyle(
CGraphicsContext::ENullPen );

gc.SetBrushColor( KRgbYellow );
gc.SetBrushStyle(
CGraphicsContext::ESolidBrush );
gc.DrawRect( aRect );

TRect rect = aRect;
rect.Shrink(20, 20);
gc.SetBrushColor( KRgbRed );
gc.DrawEllipse(rect);

TPoint point(10, 10);
gc.UseFont(CEikonEnv::Static()-
>NormalFont());
gc.DrawTextVertical(_L("DrawTextVertical
"), point, 0);
gc.DiscardFont();
```

Usually this code is called from the *Draw* function, called when the screen needs to be painted.

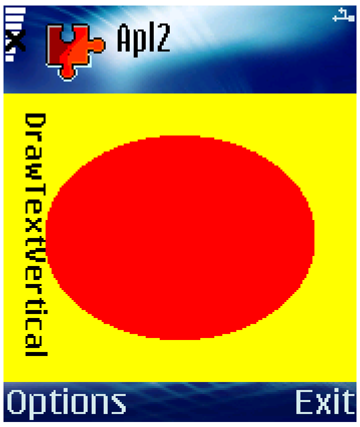

Fig. 7 Using graphics on Symbian phone

Figure 7 depicts s a capture of a Symbian phone screen showing the result of running the code above.

Also multimedia framework has support for image processing.

4.3 Video

Symbian's multimedia framework has support for playing and recording videos. The source can be a file on disk or a stream of data. The architecture is similar to audio playback, an observer class need to be created to receive the framework events. Additionally a window is needed to playback the video. *CVideoPlayerUtility* is the class used for video playback and the interface *MVideoPlayerUtilityObserver* is required to handle the playback events through callback functions.

The following class can be used to play video files:

```
class CVideoPlayer : public CBase ,public MVideoPlayerUtilityObserver
{

        CVideoPlayerUtility* iPlayer;
        CVideoPlayer ();
public:

        void ConstructL(const CCoeControl& aView);
```

```
       static CVideoPlayer * NewL(const CCoeControl& aView);
       void PlayL(const TDesC& aFisierVideo);
       void Stop();
       ~CVideoPlayer();
       //callback methods
       void MvpuoOpenComplete(TInt aError);
       void MvpuoPrepareComplete(TInt aError);
       void MvpuoFrameReady(CFbsBitmap& aFrame,TInt aError);
       void MvpuoPlayComplete(TInt aError);
       void MvpuoEvent(const TMMFEvent& aEvent);

};
void CVideoPlayer::MvpuoPrepareComplete(TInt aErr)
{
       if (KErrNone == aErr)
       {
              iPlayer->Play();
       }
       else
       {
              iPlayer->Close();
       }
}
void CVideoPlayer::MvpuoEvent(const TMMFEvent& )  {}
void CVideoPlayer::MvpuoPlayComplete(TInt aErr)    { }
void CVideoPlayer::MvpuoFrameReady(CFbsBitmap& ,TInt)    {}
void CVideoPlayer::MvpuoOpenComplete(TInt aErr)
{
       if (KErrNone == aErr) { iPlayer->Prepare(); }
}
void CVideoPlayer::Stop()
{
       iPlayer->Stop();
}
void CVideoPlayer::PlayL(const TDesC& aFisierVideo)
{
       iPlayer->Close();
       iPlayer->OpenFileL(aFisierVideo);
}
CVideoPlayer::CVideoPlayer ()   {}
CVideoPlayer::~CVideoPlayer ()
{
       if (iPlayer != NULL)
       {
              delete iPlayer;
              iPlayer = NULL;
       }
}
CVideoPlayer * CVideoPlayer::NewL(const CCoeControl& aView)
{
       CVideoPlayer * self = new (ELeave) CVideoPlayer ();
       CleanupStack::PushL(self);
       self->ConstructL(aView);
       CleanupStack::Pop();
       return self;
}
//initialize the player
void CVideoPlayer ::ConstructL(const CCoeControl& aView)
{
       TRect rect(aView.PositionRelativeToScreen(),
                     TSize(aView.Rect().Width(), aView.Rect().Height()));

       iPlayer = CVideoPlayerUtility::NewL(*this,
                     EMdaPriorityNormal,
                     EMdaPriorityPreferenceNone,
                     aView.ControlEnv()->WsSession(),
                            *(aView.ControlEnv()->ScreenDevice()),
                            *(aView.DrawableWindow()),
                    rect,
                    rect);
}
```

In order to use the video player class, an object needs to be initialized and to call the PlayL method:

```
if (!iVideoPlayer)
iVideoPlayer = CVideoPlayer::NewL(*this->iAppContainer);

if (iVideoPlayer)
   iVideoPlayer->PlayL(_L("C:\\video\\Video000.3gp"));
```

The player objects need to be released when finished.

The camera can be used for video playback and recording, if available.

6 Conclusions

Using operating system native API leads to faster applications but long development curves and difficulties on porting applications to other operating systems.

Acknowledgement

This paper presents some results of the research project IDEI 2673: *Project management methodologies for the development of mobile applications in the educational system*, financed within the framework of IDEI research program.

References

[1] P. Pocatilu and A. Pocovnicu, "Multimedia for mobile Learning," *Proc. of Informatics in Economy Conference*, May 2009, pp. 362-367

[2] Adi Rome et al. *Multimedia on Symbian OS - Inside the Convergence Device*, John Wiley & Sons Ltd, 2008, p. 23

[3] Windows Mobile 5.0 SDK: Multimedia and Games [Online]. Available: http://msdn.microsoft.com/en-us/library/aa454194.aspx (March 2009).

[4] iPhone Reference Library [Online]. Available: http://developer.apple.com/iphone/library/navigation/index.html. (March 2009).

[5] The Developer's Guide|Android Developers [Online]. Available: (http://developer.android.com/guide/index.html (March 2009).

[6] Qusay Mahmoud – *The J2ME Mobile Media API*, Sun Developer Network, 2003, http://developers.sun.com/mobility/midp/articles/mmapioverview/

[7] JSRs: Java Specification Requests – *JSR 135: Mobile Media API*, Final release 3, 2006, http://jcp.org/en/jsr/detail?id=135

[8] Bruce Hopkins – *Comparing Mobile Platforms: Java ME and Adobe Flash Lite, Sun Mobility Tech Tips, http://blogs.sun.com/mobility_techtips/entry/comparing_mobile_platforms_java_me*

[9] Eric Giguere - *Game Canvas Basics*, Sun Developer Network, 2004, http://developers.sun.com/mobility/midp/ttips/gamecanvas/index.html

Paul POCATILU graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 1998. He achieved the PhD in Economics in 2003 with thesis on Software Testing Cost Assessment Models. He has published as author and co-author over 45 articles in journals and over 40 articles on national and international conferences. He is author and co-author of 10 books, (Software Testing Costs, and Object Oriented Software Testing are two of them). He is associate professor in the Department of Economic Informatics of the Academy of Economic Studies, Bucharest. He teaches courses, seminars and laboratories on Mobile Devices Programming, Economic Informatics, Computer Programming and Project Management to graduate and postgraduate students. His current research areas are software testing, software quality, project management, and mobile application development.

Catalin BOJA is Lecturer at the Economic Informatics Department at the Academy of Economic Studies in Bucharest, Romania. In June 2004 he has graduated the Faculty of Cybernetics, Statistics and Economic Informatics at the Academy of Economic Studies in Bucharest. In March 2006 he has graduated the Informatics Project Management Master program organized by the Academy of Economic Studies of Bucharest. He is a team member in various undergoing university research projects where he applied most of his project management knowledge. Also he has received a type D IPMA certification in project management from Romanian Project Management Association which is partner of the IPMA organization. He is the author of more than 40 journal articles and scientific presentations at conferences. His work focuses on the analysis of data structures, assembler and high level programming languages. He is currently holding a PhD degree on software optimization and on improvement of software applications performance.