**Streamline - A Prototype application for video segmentation**

**Author** AnshumanRai

**Company**

Istream

**Team**

Anadi Mishra, Anshuman Rai, Dhanu, Guruprasad, Kamal Govindraj, Khan,Parag

**Inception**

Mar 2012

**DataDesign**

*Input* - Mpeg stream (Avi, Mp4, FLV)  
  
*Internal Data Structures*

*Video*

Frames in a list from mpeg library opencv

*Audio*

Tracks from mpeg library pyffmpeg

*Output*

Graphical output on a gtk based system of video with gdk pixmaps used for display of frames. Output of audio to a an ALSA device.

**Architecture Design**

Python gtk based application that uses an event based model for demonstration of functionalities.

***InterfaceDesign***

Python gtk based application that uses an event based model for demonstration of functionalities. It uses glade which is a framework for xml based representation of gtk based graphic interfaces.

**Procedural Design**

*Main program*

* Initialize instance of VideoEditor
* Call the main Gtk thread with Gtk.main()

*VideoEditor Class*

* \_\_init\_\_ method
* Read Glade File
* Initialize Gtk Builder
* Initialize Gtk Builder from glade file
* Fetch references to gtk data structures like Window, FileChooserDialog and map to local variables
* Instantiate ffmpeg classes like FFMpegReader and AlsaSoundLazyPlayer, tracks and map to local variables
* Initialize an ad dictionary which is a hash table containing key value pair. A hash computed for an ad frame is the key, and, a tuple with ad name and number of frames is value
* Initialize a dictionary of methods for event handling
* Connect the builder signals to the dictionary
* Show the main gtk window
* main\_menu\_bar\_file\_open\_activate method
* set file open mode to true
* set ad open mode to false
* show file chooser dialog box
* main\_menu\_bar\_ad\_load\_activate method
* set ad open mode to true
* set file open mode to false
* show file chooser dialog box
* file\_chooser\_dialog\_file\_activated method
* if file open mode is true
* hide the dialog box
* set local variable currentFileSelectedFullPathName to fileChooserDialog.get\_filename()
* if ad open mode is true
* hide the dialog box
* set local variable currentAdSelectedFullPathName to fileChooserDialog.get\_filename()
* Invoke ProcessAdLoad
* processAdLoad method
* Create an opencv video capture object with the the currentAdSelectedFullPathName
* Initialize variables frames\_per\_second and number\_of\_frames by calling respective opencv get property methods
* Iterate over the first frames equal to the frames per second of the ad video and for each of the frames compute a hash by invoking computeHash method. Create a tuple with currentAdSelectedFullPathName and number of frames. Insert a key value pair of hash computed above and the tuple in the ad dictionary
* computeHash method
* Convert opencv frame to an opencv IplImage
* Scale the IplImage to an 8x8 image and convert to grayscale
* Compute an average of each of the pixels
* Compute a 64 bit hash which is a bitwise or of values. The values themselves are 64 bit for each pixel with 0 or 1 as values depending on whether pixel is above or below average and then bitwise shifted.
* on\_button\_trim\_ads\_clicked
* Initialize local variable videoCaptureFile to cvCreateFileCapture(self.currentFileSelectedFullPathName);
* Set timers for audio and video playback handler
* gobject.timeout\_add(30, self.trim\_ads\_video\_playback\_handler,videoCaptureFile)
* gobject.timeout\_add(30, self.trim\_ads\_audio\_playback\_handler
* trim\_ads\_video\_playback\_handler(capture)
* Capture a frame from the input videoCaptureFile
* Compute a hash from the frame
* Check for the hash in the ad dictionary
* If there is a match, identify the tuple
* Skip the number of frames as identified in the tuple
* Convert the frame to a displayable gdk pixmap and render to the interface
* trim\_ads\_audio\_playback\_handler(capture)
* Instantiate an FFmpeg reader object
* Identify the audio track
* Instantiate an AudioSoundLazyPlayer object
* Set callback for the audio track to the push\_nowait method on the AudioSoundLazyPlayer object

*AudioSoundLazyPlayer Class*

* \_\_init\_\_ method (rate, channels, fps)
* Set various local variables
* self.\_rate=rate
* self.\_channels=channels
* self.\_d = alsaaudio.PCM()
* self.\_d.setchannels(channels)
* self.\_d.setformat(alsaaudio.PCM\_FORMAT\_S16\_LE)
* self.\_d.setperiodsize((rate\*channels)//int(fps))
* self.\_d.setrate(rate)
* push\_nowait(stamped\_buffer)
* Write the stamped buffer
* self.\_d.write(stamped\_buffer[0].data)