NLiVE

NLiVE
Design Document
v 1.0

Authors: Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan

Class:452

		Released	3-11-2005

History

Document History

Version	Date of Issue	Author	Change & Reason of Change
V 0.1	2-14-2005	Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan	Template Creation – Initial Version
V 0.2	2-18-2005	Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan	Included various diagrams.. updated sections
V 0.3	2-22-2005	Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan	Added low level DirectX Interaction design
V 0.4	2-28-2005	Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan	Combined GUI, File Project section, modified sections according to meeting
V 0.5	3-4-2005	Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan	Modified low level DirectX Interaction design and added middle layer interaction design
V 0.6	3-7-2005	Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan	Added additional diagrams in Appendix E, modified for formatting issues, all requirement key included after review. Middle and Lower Level design added. Final update
V 1.0	3-11-2005	Kevin Galloway, Nicholas Goede, James Hess, Brian Lawrence, William Mongan	Updates based of results of final review including flow diagram and moving of various diagrams to different sections.

Table of Contents

Table of Requirement Keys

1.Introduction

1.1. Purpose of the Document

This document identifies functional elements related to the NLiVE application and describes the implementation of these elements. The element description shall be of sufficient detail to verify design goals and thus serve as the guideline to the implementation of the NLiVE application. This document also forms the basis for developing test code for the NLiVE application.

1.2. Scope of the Document

This document describes the functionalities that are present in the NLiVE application. It also describes in detail the various classes that implement these functionalities and the software interfaces associated with this application.

2.General description

2.1. Product Goal

The Goal of the NLiVE application is to provide a non-linear video editor containing many of the features and functionality of other high-end commercially available video editors without the high costs involved.

2.2. Development Profile

This project makes use of the DirectShow library that is part of Microsoft DirectX. The QuickTime libraries from Apple were also considered to fill this role. The major reasons to choose DirectX over QuickTime include limited ability of the QuickTime libraries to output in any other format than QuickTime's proprietary MOV format and DirectX's greater support on Microsoft Windows.

The application is developed in a two layers plus glue architecture. What this means is that their is a lower level library that wraps and simplifies the DirectX library and functionality, a glue layer to make it interoperate correctly with the top layer, and a top layer that acts as a client to the library and actually is the application the user uses. This architecture was chosen because DirectX's DirectShow library has limited interoperability with C# and thusly the lowest level library is done in C++ with the application layer being programmed in C#. The reason for this choice over doing the entire application in C++ is because of C#'s greater tools and ease of use for GUI making, more and better programming tools available to us for C#, and general programmer ease of use over C++.

The lower level library and its attached glue library can be thought of as an Adapter pattern for converting DirectShow's COM interface to a C# compatible .NET interface. It can also be seen as a Facade pattern since it only exposes the functionality of DirectShow needed by the application.

2.3. Interfaces and interactions

The NLiVE application uses the .NET library file input/output mechanisms. The DirectX libraries are interacted with and are used in order to communicate with an external DV based Digital Video Camera.

2.4. Assumptions and Dependencies

The NLiVE application runs under the assumptions and dependencies laid out in the requirements document.

3.System Architecture Overview

The NLiVE application consists of multiple layers to provide the functionality required in a standalone non-linear video editor application. The following diagram provides an overview of those layers.

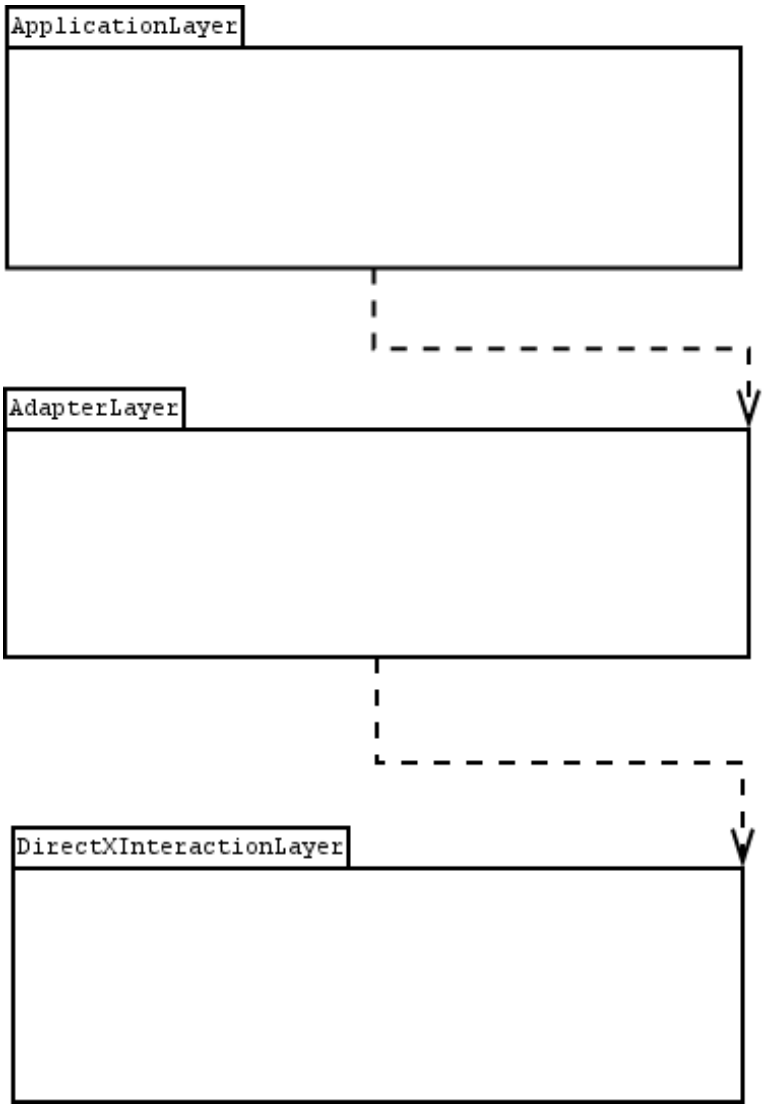

Figure 1 - NLiVE Package Diagram

Each layer provides the platform needed to utilize the functionality that each layer environment provides. For example the DirectXInteractionLayer provides a C++ layer compatible with Direct X's DirectShow while the AdapterLayer acts as a gateway allowing the Application Layer to communicate with the DirectShow functionality through the AdapterLayer.

Functional design

Each section contains the design details including class diagrams laying out the functional classes of the NLiVE application. The NLiVE application consists of multiple layers. Each layer has its own set of classes.

The following flow diagram provides an overview of how all the classes between and within the layers work together.

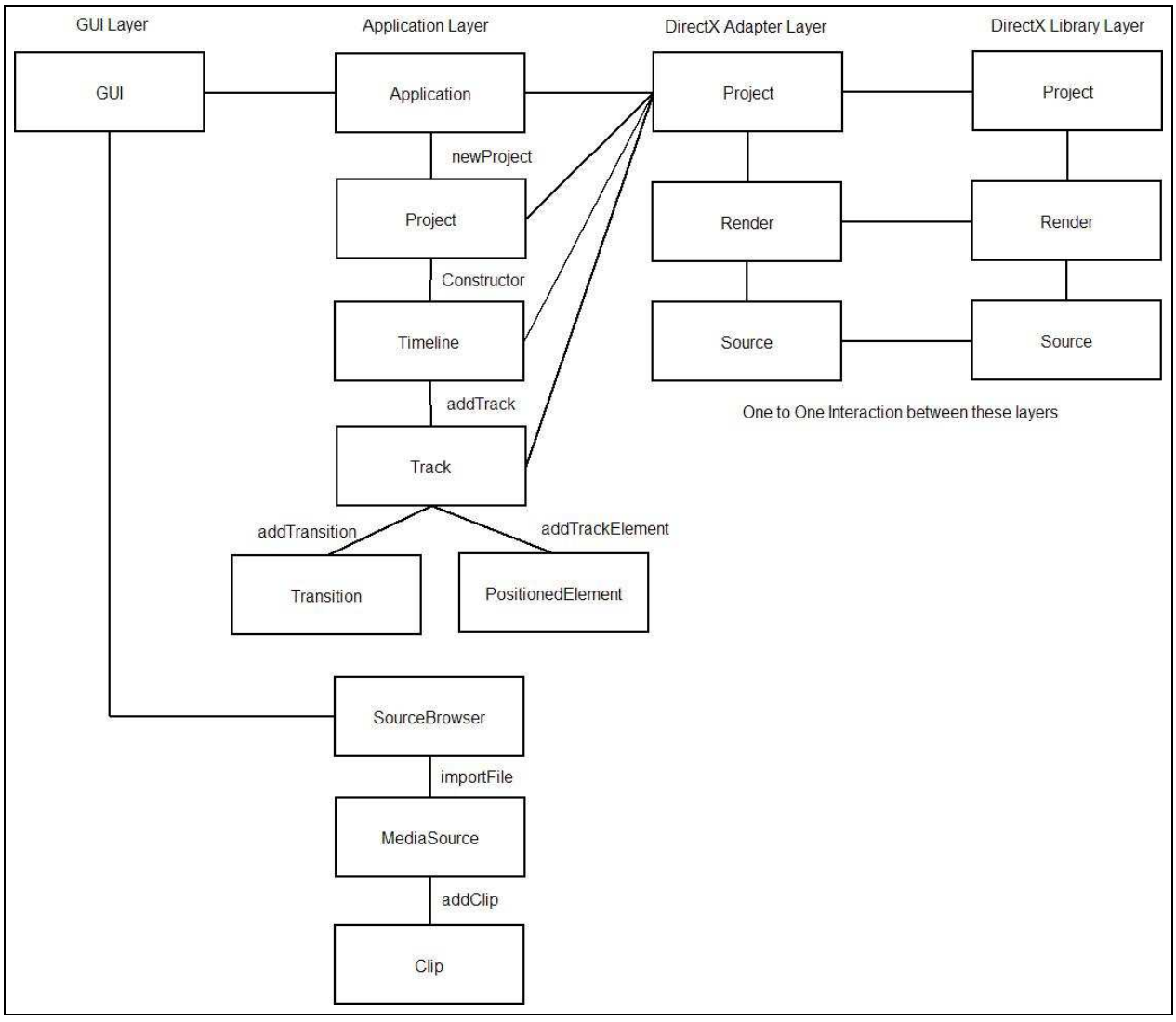

Figure 2 - Flow diagram of the major classes in NLiVE.

3.1. Application Layer

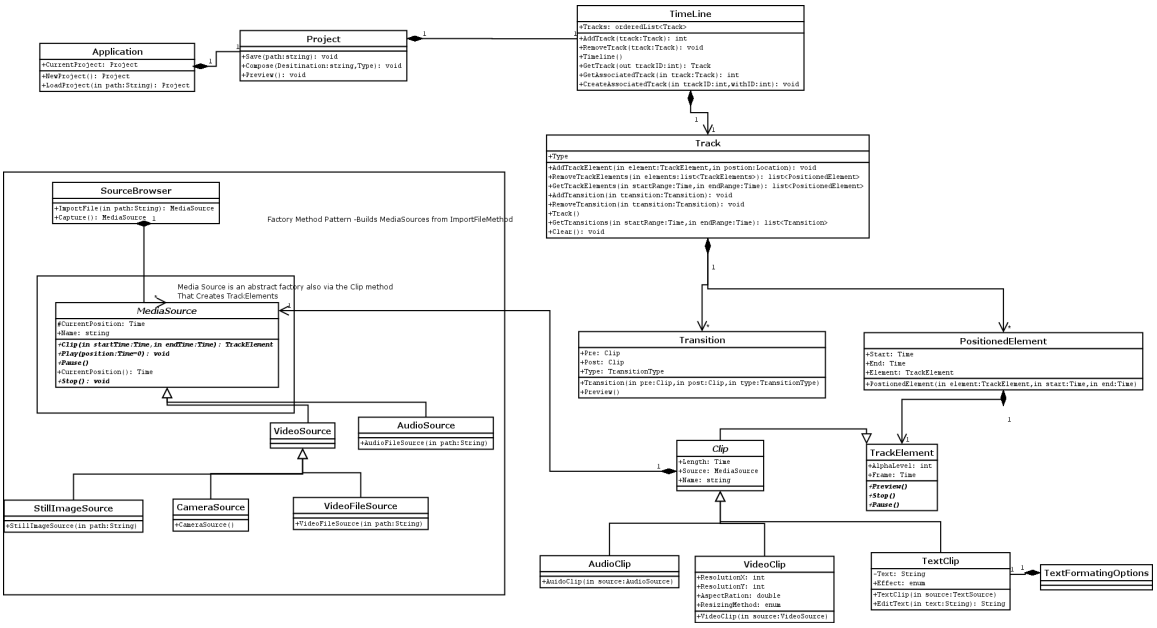

Figure 3 – NLiVE Application Layer Class Diagram

3.1.1. Application

This class is responsible for making new projects and loading existing projects. It also tracks the currently loaded project.

This class implements the following requirements:
nlive_newproject
nlive_loadproject

This class has the following methods:

- NewProject() : Project :
  - Pre-conditions: None
  - Post-conditions: A valid empty Project object is returned
- LoadProject(String path) : Project:
  - Pre-conditions: path is a valid path and points to an existing valid NLiVE project save file.
  - Post-conditions: A valid Project object containing the data in the NLiVE project save file pointed to by path.

3.1.2. Project

This class is responsible for containing the TimeLine and for saving projects to NLiVE project save files as well as previewing and generating the output movie.

This class implements the following requirements:
nlive_compose
nlive_saveproject
nlive_saveprojectas

This class has the following methods:

- Save(String Path) : void:
  - Pre-conditions: path points to a valid location for the NLiVE project save file to be written to.
  - Post-conditions: A file is created or if already present overwritten with an NLiVE project save file created using the data in this Project object.
- Preview() : void:
  - Pre-conditions: None
  - Post-conditions: A video is played by the system that is representative of the output of the Compose() method.
- Compose(string Destination, Type): void:
  - Pre-conditions: Destination is a valid path for the output video file to be written to and Type is a valid video output type.
  - Post-conditions: An output file is created of the type specified and contains the output according to the current TimeLine of this Project object. If the TimeLine is empty then the shortest valid video file of the type specified is created. This output file plays in a media player capable of playing this video.

3.1.3. TimeLine

This class is responsible for containing and administering the various tracks. It also must maintain an association between a video track and its 'default' audio track.

This class implements the following requirements:
nlive_track_add_track
nlive_track_remove_track
nlive_default_audio

This class has the following methods:

- AddTrack(Track track) : unsigned int
  - Pre-conditions: track is a valid track.
  - Post-conditions: The Track object track is added to the TimeLine and a valid ID for that track is returned.
- RemoveTrack(int trackID) : void
  - Pre-conditions: trackID represents a track stored in the TimeLine.
  - Post-conditions: The track that has an ID of trackID is removed from the TimeLine.
- GetTrack(int trackID) : Track
  - Pre-conditions: trackID represents a track stored in the TimeLine.

- Post-conditions: Returns the Track object represented in the TimeLine by trackID.

3.1.4. Track

This class represents a track or a time ordered set of TrackElement objects and Transition objects. It contains the functionality to manipulate this set as well as obtain elements from it.

This class implements the following requirements:
nlive_selected_text_clip_change_track
nlive_selected_text_clip_move
nlive_clip_timeline_placement
nlive_clip_movement
nlive_blendclip
nlive_audio_clear_track
nlive_texttrack_delete
nlive_blendimage

This class has the following methods:

- AddTrackElement(TrackElement element, Location position) : void
  - Pre-conditions: element is a valid element and position is a valid position.
  - Post-conditions: The element is placed on the track in the position specified.
- RemoveTrackElements(in elements:list<TrackElements>) : list<PostionedElement>
  - Pre-conditions: elements is a valid list. Every contained TrackElement object is valid.
  - Post-conditions: Every TrackElement object in the list is removed from the Track and the corresponding PostionedElements returned.
- GetTrackElements(Time startRange, Time endRange) : list<PositionedElements>
  - Pre-conditions: startRange and endRange are valid times.
  - Post-conditions: Returns all PositionedElements that lie within the time range given by startRange and endRange as a list.
- AddTransition(Transition transition) : void
  - Pre-conditions: transition is a valid Transition object and is for clips currently on this Track object.
  - Post-conditions: Adds transition to this Track object.
- RemoveTransition(Transition transition) : void
  - Pre-conditions: transition is in the Track.
  - Post-conditions: transition is removed from the Track.
- GetTransition(Time startRange, Time endRange) : list<Transition>
  - Pre-conditions: startRange and endRange are valid times.
  - Post-conditions: Returns a list of all Transtions between the time of startRange and endRange
- Clear() : void
  - Pre-conditions: None.
  - Post-conditions: All elements and transitions are removed from this Track.

3.1.5. Transition

This class represents a Transition and its before and after clips as well as its type. If either the start or end clip is null then the transition is at the beginning of the track or end of the track Transition.

This class implements the following requirements:
nlive_transition
nlive_transition_preview

This class has the following methods:

- Preview() : void :
    - Pre-conditions: None
    - Post-conditions: The transition is displayed as a preview that approximates the result that would be outputted via Preview.Compose()

3.1.6. PositionedElement

This class represents a TrackElement object with an associated position in time.

This class implements the following requirements:
nlive_clip_start
nlive_clip_end

3.1.7. TrackElement

This class represents a non-transition object that can appear on a track.

This class implements the following requirements:
nlive_preview_clip_command
nlive_preview_stop
nlive_preview_pause
nlive_change_frame
nlive_clip_alpha

This class has the following methods:

- Preview() : void :
    - Pre-conditions: None
    - Post-conditions: The TrackElement is displayed as a preview that approximates the result that are outputted via Preview.Compose() for this TrackElement object.
- Stop() : void:
    - Pre-conditions: None
    - Post-conditions: Sets current state of preview for this TrackElement to stop and resets the frame to the starting position.
- Pause() : void:

- Pre-conditions: None
- Post-conditions: Sets current state of preview for this TrackElement to stop.

3.1.8. Clip

Represents a clip of some part of a media source or a TextClip.

This class implements the following requirements:
nlive_name_media_clip

3.1.9. AudioClip

Represents a clip of an AudioSource.

This class implements the following requirements:
nlive_audio_clip

3.1.10. VideoClip

Represents a clip of a VideoSource and associated properties.

This class implements the following requirements:
nlive_clip_resolution
nlive_resize
nlive_resize_display

3.1.11. TextClip

Represents a piece of Text in clip form and tracks possible effects.

This class implements the following requirements:
nlive_texttrack_add
nlive_edit_text_for_clip

- EditText(String text) : String :
  - Pre-conditions: None.
  - Post-conditions: Sets TextClip object's text string to text and returns the current value.

3.1.12. TextFormatingOptions

Stores and allows the changing of Text Formatting Options

3.1.13. SourceBrowser

Represents a set of MediaSources and has a Factory Method to make them from file paths or to create a CameraSource from a DV camera.

This class implements the following requirements:
nlive_source_browser
nlive_import

- ImportFile(String path) : MediaSource
  - Pre-conditions: path is a valid path and is a valid media file.
  - Post-conditions: Uses information from the path to create the correct type of MediaSource and returns it.
- Capture() : MediaSource
  - Pre-conditions: a functioning camera is connected and ready to import video.
  - Post-conditions: A valid CameraSource is created and returned.

3.1.14. MediaSource

Represents a media source and acts as an Abstract Factory for creating Track Elements via the Clip method. Also allows for playing MediaSources.

This class implements the following requirements:
nlive_cut

- Play() : void :
  - Pre-conditions: None
  - Post-conditions: The MediaSource is played for the user.
- Stop() : void:
  - Pre-conditions: None
  - Post-conditions: Sets current state of playing for this MediaSource to stop and resets the frame to the starting position.
- Pause() : void:
  - Pre-conditions: None
  - Post-conditions: Sets current state of playing for this MediaSource to stop.
- Clip(Time startTime, Time endTime) : TrackElement:
  - Pre-conditions: startTime and endTime are ordered such that endTime is later and startTime and endTime both lie within the bounds of the length of the MediaSource.
  - Post-condition: Returns a valid TrackElement object.

3.1.15. VideoSource

Parent Class for all Video based MediaSources

This class implements the following requirements:

nlive_auto_audio_clip

3.1.16. AudioSource

Class that represents an Audio file.

This class implements the following requirements:
nlive_file_mp3
nlive_file_wav

3.1.17. StillImageSource

Class that represents a still image file.

This class implements the following requirements:
nlive_still_image_as_video

3.1.18. VideoFileSource

Class that represents a video file.

This class implements the following requirements:
nlive_file_avi support
nlive_file_mpeg1
nlive_file_dv

3.1.19. CameraSource

Class that represents a DV camera.

This class implements the following requirements:
nlive_capture

3.2. Adapter Layer

The adapter layer is functionally very similar to the DirectX library layer, per the specifications of the Managed Wrapper. One notable difference is that the return values of modules in the Adapter layer are members of the Application layer. At the DirectX library layer, the return values are DirectX COM objects of similar names.

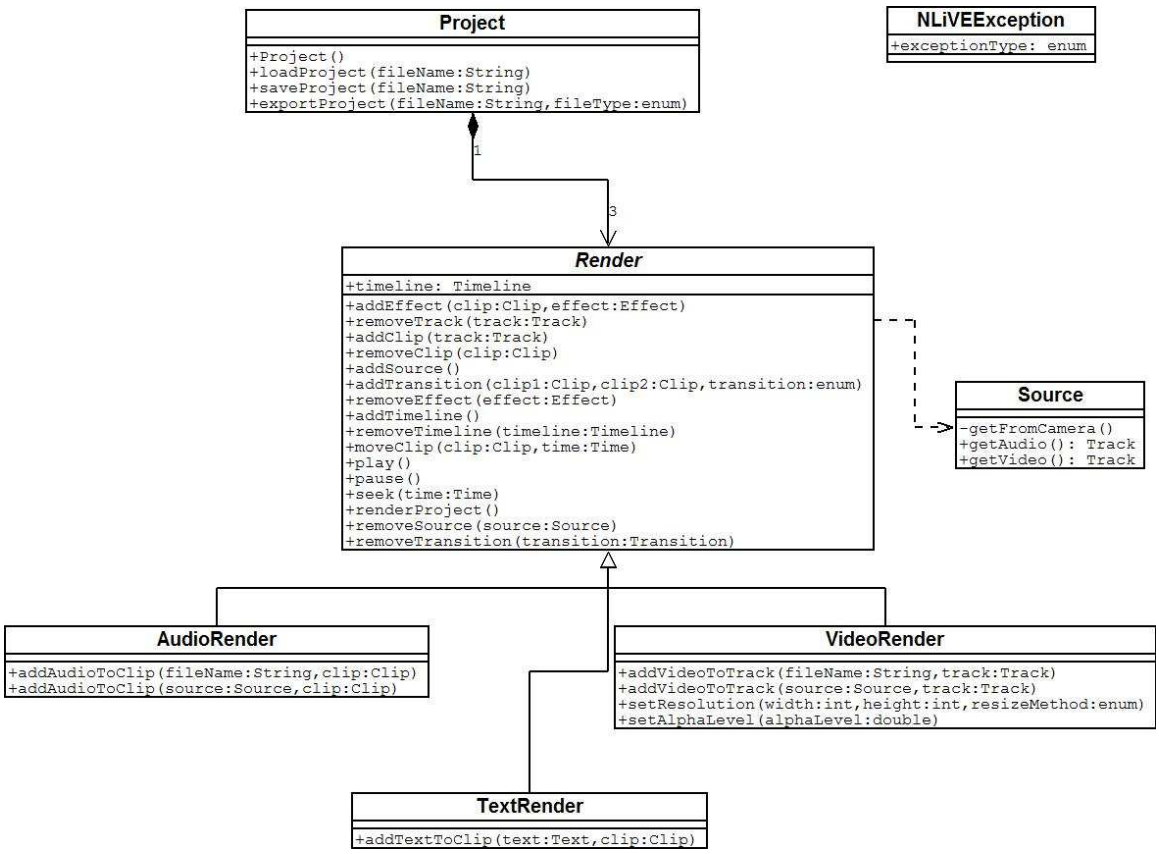

Figure 4 - NLiVE Adapter Layer Class Diagram

3.2.1. Project

Project represents the liaison between the application layer and the DirectX library layer. It directs requests to all rendering modules of this layer.

This module corresponds most directly with the following application layer modules:
Project
Timeline

- Project() : void :
    - Pre-conditions: A Lower layer project has not been created before.

    - Post-conditions: A Project with an empty timeline are created for the user.
- loadProject(filename: String) : void:
    - Pre-conditions: None
    - Post-conditions: The specified project name is loaded from disk and the low level representation is created.
- saveProject(filename: String) : void:
    - Pre-conditions: A project exists in memory.
    - Post-conditions: The project is serialized and saved to disk.

3.2.2. Source

Source is responsible for interfacing with the DV camera and retrieving audio and video which can be converted into tracks.

This module corresponds most directly with the following application layer modules:
CameraSource

- getFromCamera() : void :
    - Pre-conditions: The DV Camera has been hooked up to the system and has created a project.
    - Post-conditions: The video is read from the camera to an internal representation. The audio or video are extracted by getAudio() and getVideo(), respectively.
- getAudio() : void:
    - Pre-conditions: The DV Camera has been hooked up to the system and has created a project.
    - Post-conditions: getFromCamera() is called if it has not been called already, and the audio is extracted as a Track.
- getVideo() : void:
    - Pre-conditions: The DV Camera has been hooked up to the system and has created a project.
    - Post-conditions: getFromCamera() is called if it has not been called already, and the video is extracted as a Track.

3.2.3. Render

Render is responsible for overseeing the rendering and video previewing process. It is an abstract class, implemented by AudioRender, VideoRender and TextRender to complete the various specific tasks.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addEffect(clip : Clip, effect : enum) : void :
    - Pre-conditions: A project with clips has been created.

    - Post-conditions: The given effect has been added to the specified clip.
- removeTrack(track:Track) : void:
    - Pre-conditions: None
    - Post-conditions: The specified track is removed, leaving an empty default track.
- addClip() : void:
    - Pre-conditions: None
    - Post-conditions: A clip is added to the track associated with the module (text, audio or video).
- removeClip(clip: Clip) : void:
    - Pre-conditions: None
    - Post-conditions: The specified clip is removed from the track.
- addSource() : void:
    - Pre-conditions: None
    - Post-conditions: A file or camera source is opened for extraction.
- addTransition(clip1:Clip, clip2:Clip, transition:enum) : void:
    - Pre-conditions: At least one clip exists on the track (if only one track exists, then a fade to or from black is desired).
    - Post-conditions: The given transition is applied to the clip(s).
- removeEffect(effect:Effect)
    - Pre-conditions: The given effect exists on the timeline.
    - Post-conditions: The given effect is removed.
- moveClip(clip:Clip, time:Time)
    - Pre-conditions: The given clip exists on the timeline, and the given destination time space is available on the timeline.
    - Post-conditions: The given clip is moved to the desired place.
- play()
    - Pre-conditions: None
    - Post-conditions: The video is rendered for preview via a call to renderProject().
- pause()
    - Pre-conditions: The video is playing
    - Post-conditions: The video is stopped.
- seek(time:Time)
    - Pre-conditions: None
    - Post-conditions: The playback position is moved to the given time, if time is in [0, project.length].
- renderProject()
    - Pre-conditions: None
    - Post-conditions: The video is rendered.
- removeSource(source:Source)
    - Pre-conditions: The given source exists in the project.
    - Post-conditions: The given source is removed.
- removeTransition(transition:Transition)
    - Pre-conditions: The given transition exists on the timeline.
    - Post-conditions: The given transition is removed.

3.2.3.1. Audio Render

AudioRender is responsible for handling audio rendering tasks delegated from the Render class.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addAudioToClip(filename:String, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given file source audio is loaded into the clip.
- addAudioToClip(source:Source, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given camera source audio is loaded into the clip.

3.2.3.2. Video Render

VideoRender is responsible for handling video rendering tasks delegated from the Render class.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addVideoToClip(filename:String, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given file source video is loaded into the clip.
- addVideoToClip(source:Source, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given camera source video is loaded into the clip.

3.2.3.3. Text Render

TextRender is responsible for handling text rendering tasks delegated from the Render class.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addTextToClip(text:Text, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.

- Post-conditions: The given text is loaded into the clip with its corresponding formatting flags.

3.2.4. NLiVE Exception

The NLive Exception class is responsible for interpreting exceptions thrown from the DirectX library layer and, if necessary, passing them along to the application layer for graceful handling.

3.3. Direct X Interaction Layer

The adapter layer is functionally very similar to the Adapter layer, per the specifications of the Managed Wrapper. One notable difference is that the return values of modules in the Adapter layer are members of the Application layer. At the DirectX library layer, the return values are DirectX COM objects of similar names.

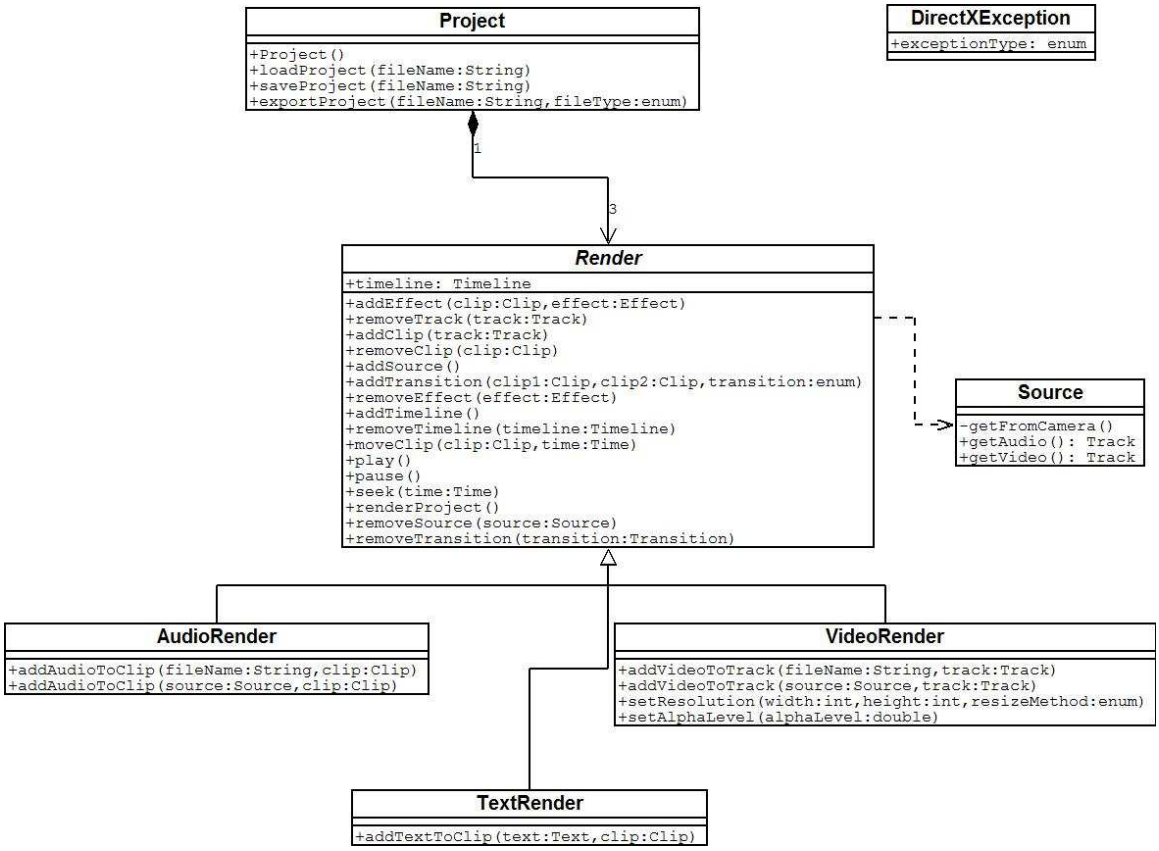

Figure 5 - NLiVE Direct X Layer Class Diagram

3.3.1. Project

Project represents the low level DirectX COM timeline and project data. It directs requests to all rendering modules of this layer as they are initiated from the Adapter.

This module corresponds most directly with the following application layer modules:
Project
Timeline

- Project() : void :
    - Pre-conditions: A Lower layer project has not been created before.

    - Post-conditions: A Project with an empty timeline are created for the user.
- loadProject(filename: String) : void:
    - Pre-conditions: None
    - Post-conditions: The specified project name is loaded from disk and the low level representation is created.
- saveProject(filename: String) : void:
    - Pre-conditions: A project exists in memory.
    - Post-conditions: The project is serialized and saved to disk.

3.3.2. Source

Source is responsible for interfacing with the DV camera and retrieving audio and video which can be converted into tracks.

This module corresponds most directly with the following application layer modules:
CameraSource

- getFromCamera() : void :
    - Pre-conditions: The DV Camera has been hooked up to the system and has created a project.
    - Post-conditions: The video is read from the camera to an internal representation. The audio or video are extracted by getAudio() and getVideo(), respectively.
- getAudio() : void:
    - Pre-conditions: The DV Camera has been hooked up to the system and has created a project.
    - Post-conditions: getFromCamera() is called if it has not been called already, and the audio is extracted as a Track.
- getVideo() : void:
    - Pre-conditions: The DV Camera has been hooked up to the system and has created a project.
    - Post-conditions: getFromCamera() is called if it has not been called already, and the video is extracted as a Track.

3.3.3. Render

Render is responsible for overseeing the rendering and video previewing process. It is an abstract class, implemented by AudioRender, VideoRender and TextRender to complete the various specific tasks.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addEffect(clip : Clip, effect : enum) : void :
    - Pre-conditions: A project with clips has been created.

    - Post-conditions: The given effect has been added to the specified clip.
- removeTrack(track:Track) : void:
    - Pre-conditions: None
    - Post-conditions: The specified track is removed, leaving an empty default track.
- addClip() : void:
    - Pre-conditions: None
    - Post-conditions: A clip is added to the track associated with the module (text, audio or video).
- removeClip(clip: Clip) : void:
    - Pre-conditions: None
    - Post-conditions: The specified clip is removed from the track.
- addSource() : void:
    - Pre-conditions: None
    - Post-conditions: A file or camera source is opened for extraction.
- addTransition(clip1:Clip, clip2:Clip, transition:enum) : void:
    - Pre-conditions: At least one clip exists on the track (if only one track exists, then a fade to or from black is desired).
    - Post-conditions: The given transition is applied to the clip(s).
- removeEffect(effect:Effect)
    - Pre-conditions: The given effect exists on the timeline.
    - Post-conditions: The given effect is removed.
- moveClip(clip:Clip, time:Time)
    - Pre-conditions: The given clip exists on the timeline, and the given destination time space is available on the timeline.
    - Post-conditions: The given clip is moved to the desired place.
- play()
    - Pre-conditions: None
    - Post-conditions: The video is rendered for preview via a call to renderProject().
- pause()
    - Pre-conditions: The video is playing
    - Post-conditions: The video is stopped.
- seek(time:Time)
    - Pre-conditions: None
    - Post-conditions: The playback position is moved to the given time, if time is in [0, project.length].
- renderProject()
    - Pre-conditions: None
    - Post-conditions: The video is rendered.
- removeSource(source:Source)
    - Pre-conditions: The given source exists in the project.
    - Post-conditions: The given source is removed.
- removeTransition(transition:Transition)
    - Pre-conditions: The given transition exists on the timeline.
    - Post-conditions: The given transition is removed.

3.3.3.1. Audio Render

AudioRender is responsible for handling audio rendering tasks delegated from the Render class.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addAudioToClip(filename:String, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given file source audio is loaded into the clip.
- addAudioToClip(source:Source, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given camera source audio is loaded into the clip.

3.3.3.2. Video Render

VideoRender is responsible for handling video rendering tasks delegated from the Render class.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addVideoToClip(filename:String, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given file source video is loaded into the clip.
- addVideoToClip(source:Source, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.
  - Post-conditions: The given camera source video is loaded into the clip.

3.3.3.3. Text Render

TextRender is responsible for handling text rendering tasks delegated from the Render class.

This module corresponds most directly with the following application layer modules:
TrackElement
Clip

- addTextToClip(text:Text, clip:Clip)
  - Pre-conditions: The given clip exists on the timeline.

- Post-conditions: The given text is loaded into the clip with its corresponding formatting flags.

3.3.4. DirectX Exception

The DirectX Exception class is responsible for receiving and interpreting errors thrown by the DirectX COM library. If an exception can be handled at this layer, it is done silently. Otherwise, it is passed to the Adapter layer for further processing or passing to the Application layer for user interaction.

3.3.5. User Interface

The NLiVE application provides a windows based dockable User Interface. The following figure provides a brief outline of the Main GUI.

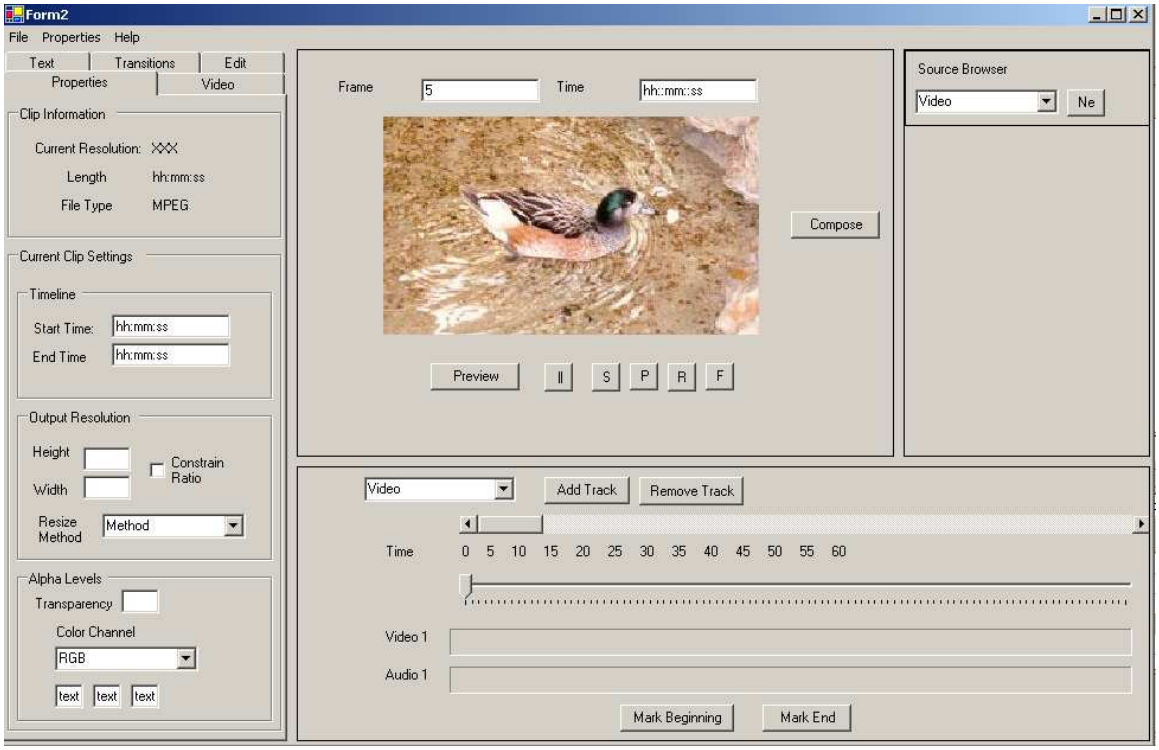

Figure 6 –NLiVE application Main GUI overview

The NLiVE application contains the following Visual Components in the GUI.

- Preview area
  - Play, stop, pause. Etc
    nlive_preview_stop
    nlive_preview_pause
  - Frame stepping buttons
    nlive_change_frame
  - Frame and time display
    nlive_change_frame
  - Preview window
    nlive_clip_preview
    nlive_transition_preview
    nlive_preview_output_command
    nlive_preview_output_display
    nlive_preview_clip_display
  - Preview slider bar
    nlive_clip_end

nlive_clip_start
- Compose button
  nlive_compose_command
- Toggle switch between clip and project
  nlive_preview_output_display
  nlive_preview_clip_display

- Source browser
  nlive_source_browser
  - File type differential (drop down)
    Non-required widget used for sorting.
  - Thumbnails of media
    Allows user to select media sources for: nlive_name_media_clip, and nlive_clip_preview.
  - Import new sources button
    nlive_import_command
    nlive_import_command_dialog
- Time line
  - Add track, delete track
    nlive_track_add_track
    nlive_audio_clip_removal
  - Tracks
    nlive_clip_timeline_placement
    nlive_clip_timeline_view
    nlive_clip_preview
    nlive_transition_postion
    nlive_text_clip_select
  - Track type list
    nlive_track_add_track
  - Graduations of time
    Non-required, used to denote time associated with location on timeline slider bar.
  - Timeline slider bar
    nlive_clip_timeline_placement
  - Change scale of timeline zoom in/zoom out
    nlive_transition_postion
  - Mark Begin/End
    nlive_clip_start
    nlive_clip_end
    nlive_track_select_clip
- Option pane
  nlive_option_pane
  nlive_option_pane_texttrack
  nlive_transition_view
  nlive_clip_properties_resolution
  - Tabs of modes/functions (text, video, sound)
    nlive_option_pane_texttrack_effects

- Properties of actions to be taken for video, pictures, sound, and text
  - Blend image button
    nlive_blendimage_command
    nlive_blendimage_command_dialog
  - Blend clip button
    nlive_blendclip_command
    nlive_blendclip_command_dialog
  - Add transition before clip
    nlive_transition_command
    nlive_transition_dialog
    nlive_transition_timeline_beginning
  - Add transition after clip
    nlive_transition_command
    nlive_transition_dialog
    nlive_transition_timeline_ending
  - New output resolution
    nlive_clip_resolution
  - Constrain aspect ratio
    Non required, allows user to automatically maintain aspect ratio while changing resolution
  - Alpha transparency
    nlive_clip_properties_alpha
  - Alpha channels settings
    nlive_clip_properties_alpha
  - Alpha channel selector
    nlive_clip_properties_alpha
  - Clip start time/end time
    Location of clip along timeline.
  - Text time begin/end
    Required for setting duration of text overlay.
  - Text position start/end
    Required for functionality of animated text, and positioning of text on screen.
  - Text Richtext box
    nlive_edit_text_for_clip
    nlive_texttrack_add_command
    nlive_texttrack_respond
  - Selectors for font, font size, and font style (bold, italic, underline), font color.
    Not required, but useful for formatting text to appear on screen.
  - Edit start time and start frame
    nlive_clip_start
  - Edit end time and end frame
    nlive_clip_end
  - Cut button

nlive_cut_command
- Text effects (fade in/out, blink)
  nlive_option_pane_texttrack_effects
- Resize method selector
  nlive_resize_display
- Transitions tab
  Facilitates the selection of available transitions.

- Menu bar
  - All file operations
    nlive_import
    nlive_name_media_source
    nlive_capture_command
    nlive_capture_command_dialog
  - All project operations
    nlive_newproject_command
    nlive_newproject_command_dialog
    nlive_loadproject_command
    nlive_loadproject_command_dialog
    nlive_saveproject_command
    nlive_saveproject_command_dialog
    nlive_saveprojectas_command
    nlive_saveprojectas_command_dialog
  - Exit
    Gives user ability to exit NLiVE.
  - Export operations
    nlive_compose_command
    nlive_compose_dialog
    nlive_compose_progress
  - Help

The Preview area GUI component exclusively implements the following requirements:
nlive_preview_output_command
nlive_preview_output_display
nlive_preview_clip_display
nlive_clip_preview

The Source browser GUI component exclusively implements the following requirements:
nlive_import_command
nlive_import_command_dialog

The Time line GUI component exclusively implements the following requirements:
nlive_audio_clip_removal
nlive_clip_timeline_view
nlive_transition_postion
nlive_track_select_clip
nlive_text_clip_select

The Option pane GUI component exclusively implements the following requirements:

nlive_option_pane
nlive_option_pane_texttrack
nlive_option_pane_texttrack_effects
nlive_blendimage_command
nlive_blendimage_command_dialog
nlive_blendclip_command
nlive_blendclip_command_dialog
nlive_transition_command
nlive_transition_timeline_beginning
nlive_transition_dialog
nlive_transition_timeline_ending
nlive_clip_properties_alpha
nlive_clip_properties_resolution
nlive_cut_command
nlive_texttrack_add_command
nlive_texttrack_respond
nlive_transition_view

The Menu bar GUI component exclusively implements the following requirements:
nlive_name_media_source
nlive_compose_command
nlive_compose_dialog
nlive_compose_progress
nlive_capture_command
nlive_capture_command_dialog
nlive_newproject_command
nlive_newproject_command_dialog
nlive_loadproject_command
nlive_loadproject_command_dialog
nlive_saveproject_command
nlive_saveproject_command_dialog
nlive_saveprojectas_command
nlive_saveprojectas_command_dialog

The following sequence diagrams provide an overview of how the GUI components interact with the user and various classes defined in the application layer.

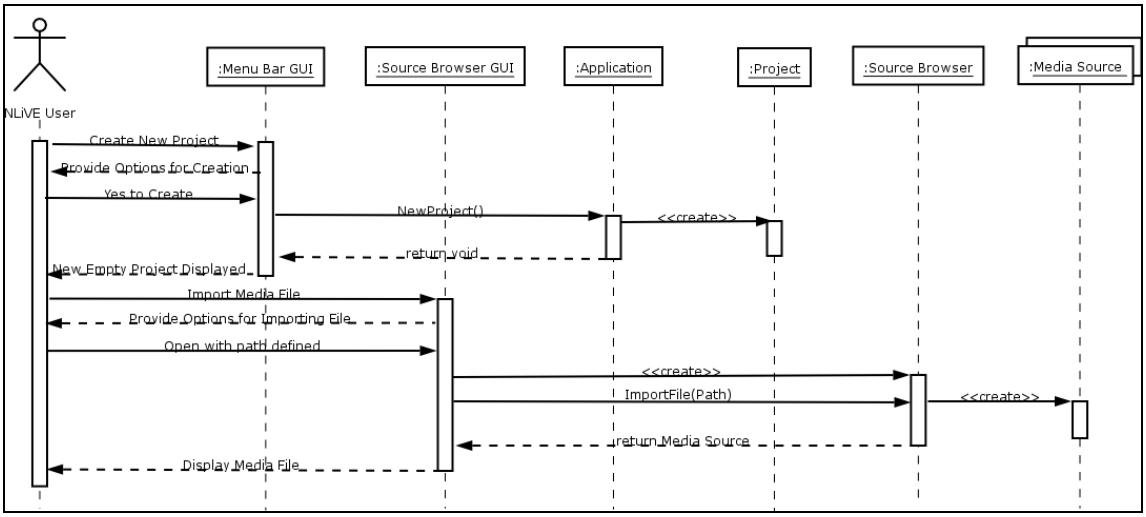

Figure 7- NLiVE New Project / Import Media File Sequence Diagram

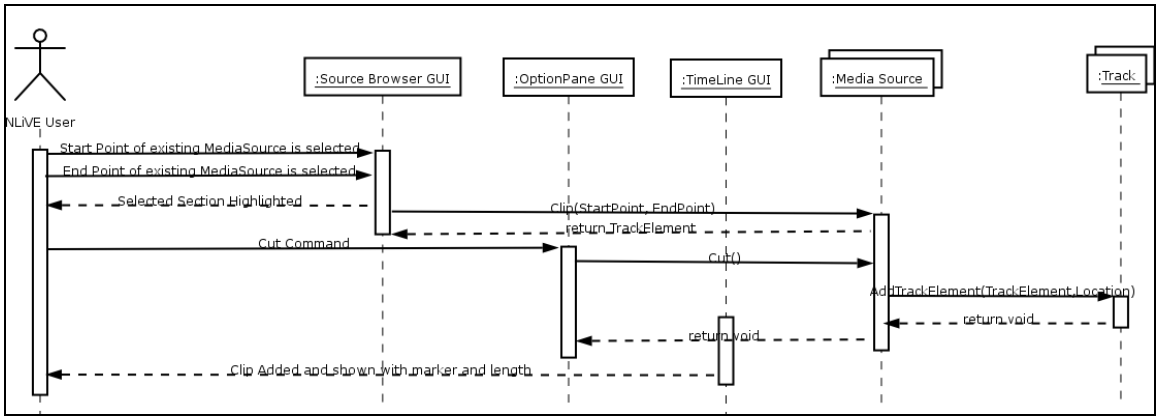

Figure 8 - NLiVE Cut to Timeline Sequence Diagram

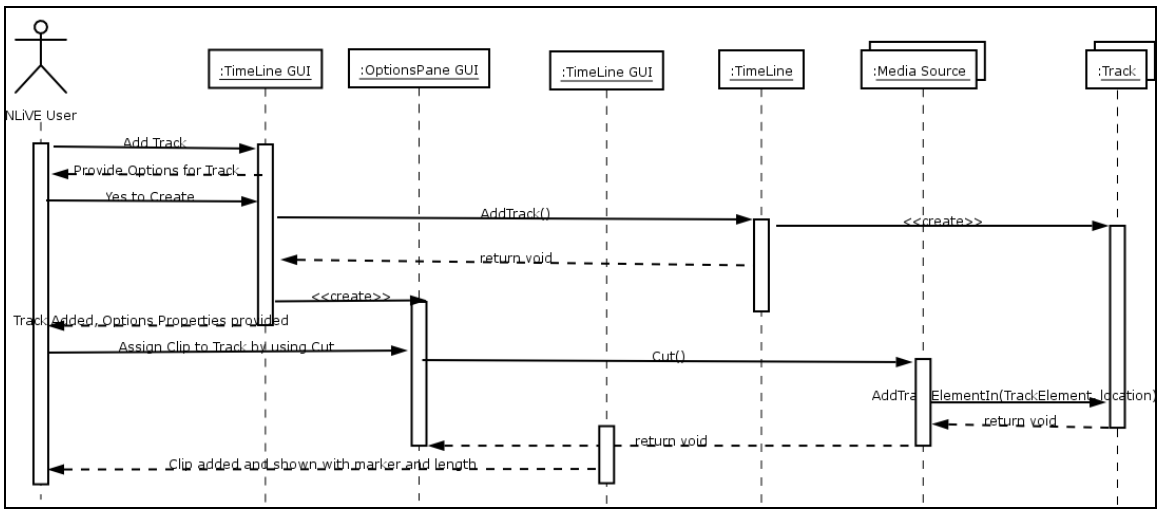

Figure 9 - NLiVE Add Track/Assign Clip Sequence Diagram

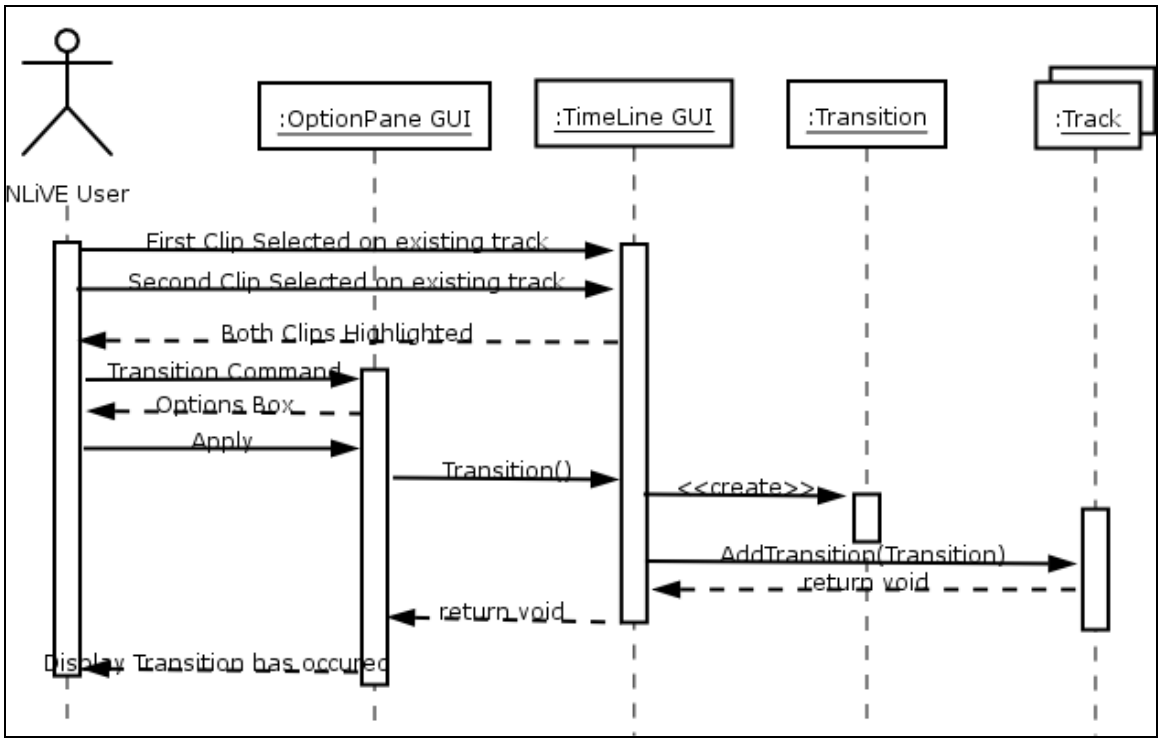

Figure 10 – NLiVE Transition Between Clips Sequence Diagram

The following diagram provides an overview of the interactions a user would take to perform various tasks with detail related to the project (ie save, load, etc..).

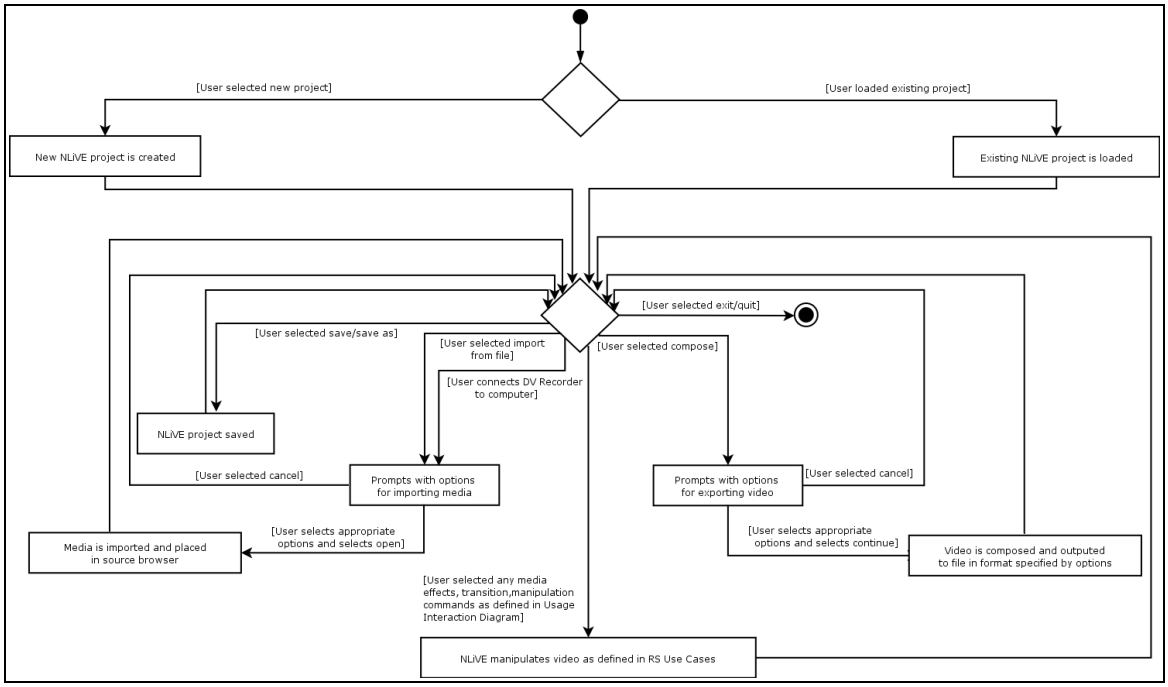

Figure 11 - NLiVE Project Interaction Diagram

3.3.6. Hardware Interface

The NLiVE application requires that the hardware satisfies the minimum requirements specified by the following:

1. Functional Computer including Monitor, Mouse, Keyboard and audio output device
2. Minimum 1.5 GHz processor (Pentium 4 or equivalent)
3. Minimum 256 MB Ram
4. DirectX 7 compatible video card
5. Minimum 200 MB of free space

If this list is satisfied the following requirements will be fulfilled:
nlive_requirements_hardware
nlive_requirements_hardware_camera_support
nlive_requirements_hardware_harddrive
nlive_requirements_hardware_memory
nlive_requirements_hardware_processor
nlive_requirements_hardware_video

3.3.7. Operating system interfaces

The NLiVE application makes no direct calls to the Operating System.

It is assumed that NLiVE will be run under a supported operating system as defined by the following requirement:
nlive_requirements_os

3.3.8. Software Interface

The NLiVE application utilizes Microsoft's DirectX 9.0c and is designed to run with this and later versions. The details of the class interactions that utilize this API are described in the Direct X Layer functional design section.

The following requirement is fulfilled through this interaction:
nlive_requirements_software_directx

3.4. Performance

The NLiVE application will not consume any unnecessary processor or memory resources.

The NLiVE application is implemented in a way that the following requirement will be fulfilled:
nlive_system_stability

Appendix A - Definitions

Media Source: A video, image, or audio file that has been imported into the project.
Source Browser: A viewing and organizational area of the application containing project specific, user imported media source.
Clip: A segment of a media source specified by the user.
NLiVE: Non-Linear Video Editor – name of this application and project
Track: A chronological container of clips that allows per specification of duration.
Timeline: This is the overview representation of the project. The timeline shows the media project as a combination of video, audio and text tracks as defined above.

Appendix B - Abbreviations

GUI – Graphical User Interface

Appendix C – References

1. NLiVE: *NLiVE Requirement Specification.* V 1.0, February 16th, 2005.

2. Tom Pender: *UML Bible*. John Wiley & Sons, 2003.

3. Andrew Filev: *Professional UML with Visual Studio .NET*. Wrox Press, 2002.

4. DirectX XTL Reference – (XML Schema)
   http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directshow/htm/xmlreference.asp

Appendix D – NLiVE File Format Definition

The NLiVE project file format is based on an Extensible Markup Language (XML) persistence format, called XTL. The NLiVE application utilizes the base elements provided by XTL and adds additional elements and attributes as defined in table 1. Please Appendix C for the location of the XTL Reference that provides the XML Schema for the base XTL provided by Microsoft's DirectShow.

XTL provides the following base elements: (Children are shown)	XTL provides the following base attributes: Attribute [element applies to]

• at
• clip
  ○ effect
• composite
  ○ composite
  ○ track
  ○ effect
  ○ transition
• effect
  ○ param
• group
  ○ composite
  ○ effect
  ○ track
• linear
• param
  ○ at
  ○ linear
• timeline (Must be root)
  ○ group
• track
  ○ clip
  ○ effect
  ○ transition
• transition
  ○ param
	• bitdepth [group]
	• buffering [group]
	• clsid [at, effect, transition]
	• cutpoint [transition]
	• cutsonly [transition]
	• defaultfx [timeline]
	• defaulttrans [timeline]
	• enablefx [timeline]
	• enabletrans [timeline]
	• framerate [clip, group, timeline]
	• height [group]
	• lock [clip, composite, effect, group, timeline, transition]
	• mlength [clip]
	• mstart [clip]
	• mstop [clip]
	• mute [clip, composite, effect, group, timeline, transition]
	• name [group,param]
	• previewmode [group]
	• samplingrate [group]
	• src [clip]
	• start [clip, effect, transition]
	• stop [clip, effect, transition]
	• stream [clip]
	• stretchmode [clip]
	• swapinputs [transition]
	• time [at,linear]
	• type [group]
	• userdata [clip, composite, effect, group, timeline, transition]
	• *userid [clip, composite, effect, group, timeline, transition]*
	• *username [clip, composite, effect, group, timeline, transition]*
	• value [at, linear, param]
	• width [group]

The NLiVE application adds these additional elements in addition to utilizing base elements provided by	The NLiVE application adds these additional attributes in addition to utilizing base attributes provided by

XTL.	XTL.

• Source Browser
  ○ clip
• Project
  ○ Source Browser
  ○ Timeline
  ○ Preview Window
• Preview Window
	• open [Project, Source Browser]
	• closed [Project, Source Browser]
	• xpos [Source Browser, Preview Window]
	• ypos [Source Browser, Preview Window]

Table 1 - NLiVE Project File Format