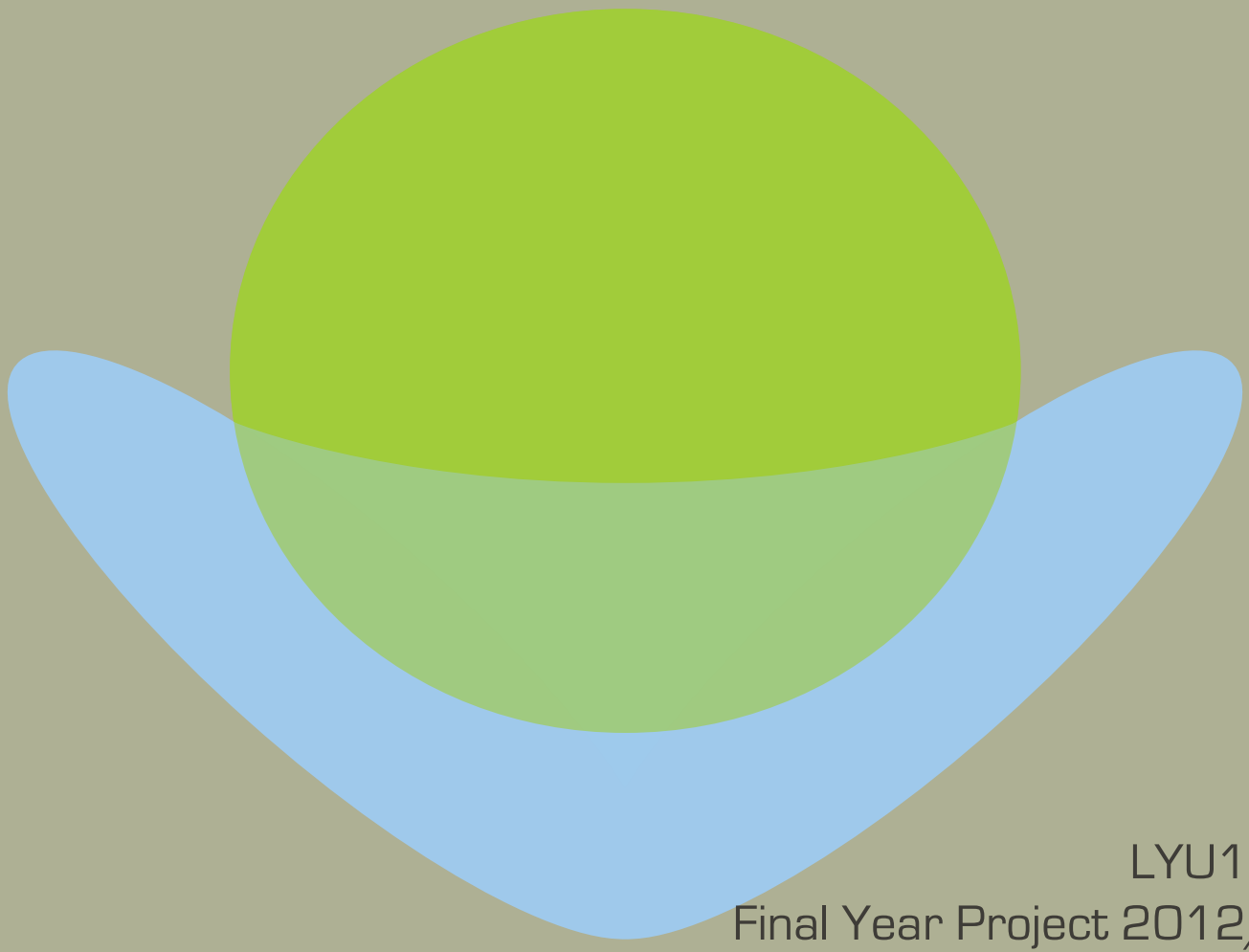
Department of Computer Science and Engineering

The Chinese University of Hong Kong

vPresent

Collaborative Presentation System on Mobile Devices



LYU1203

Final Year Project 2012/13

Spring 2013

Report

Supervisor: Professor Michael R. Lyu

Prepared by Jimmy, Sinn Lok Tsun (1155002358)

Abstract

vPresent is a project promoting a new concept of presentation Collaborative presentation. Collaborative presentation is the concept of presentation allow multiple presenters to make contribution to presentation, and making the presentation more interesting and unique. With raising popularity of mobile operating systems and mobile applications, we implement applications on mobile platform for promoting collaborative presentation. By introducing more features that are familiar to the general public, it helps promoting collaborative presentation in a more friendly way. The concept of collaborative presentation do not limit or require form of presentation by the application. It points to a new direction for improving presentation style.

Contents

Chapter 1

Introduction

1.1. Background and Objective

Presentation systems was a large market for both education, business etc. In education aspect, teachers, lectures and professors use slides show and presentation software for lecture. Other than lecture, presentation system is also used in conferences. Presenting a thesis, researchers and scholars would also use slides to present their thesis in conference. Presentation is widely used in transferring knowledge on educational purpose.

In traditional slide-based presentation, usually only one presenter dominate the whole presentation. If there are two or more presenters responsible for the presentation, taking turn would introduce inconveniences and interruptions towards presentation flow.

Therefore, we are trying to promote a new concept of presentation, namely collaborative presentation. Collaborative presentation aimed at preventing single presenter dominate whole presentation by allowing others join and contribute to presentation, and the ultimate goal is to vague the boundary between presenters and viewers. Promoting the new concept of presentation, we are going to develop an application on iOS,

demonstrating characteristics of collaborative presentation.

1.2. Summary of Fall 2012

1.2.1. Implemented Features

This project is started from Summer 2012, and we have achieved few milestones by Fall 2012.

Firstly, we have made two prototype applications for collaborative presentation, for presenters and moderators respectively. Moderator app and presenter app are connected via wireless network. Moderator app act as server, while presenter app act as client. Client-server model of at most 255 client (restrict by protocol) is support.

Apart from network communication, we have also worked on synchronization of multiple views with external display. External display is connected with an VGA or HDMI adapter, and the presentation could be display in both device display and external display.

With both network communication and external display synchronization, presenter could control moderator wirelessly. While moderator is connected to external display, presenter could control slides and images show in external display such as projector. The following types of media supported in the prototype with wireless synchronization with external display:

- PNG images as slides
- Drawing Pad with arbitrary path drawing

1.3. Highlight in Spring 2013

In Spring 2013, we have decided to work on two main directions, refining scenario of collaborative presentation, completing presentation work flow and enriching presentation content by supporting more types of media.

1.3.1. Refinement of Collaborative Presentation

Collaborative presentation is a good idea in reinventing presentation. However, it is not applicable to any scenario, as well as causing disturbance during presentation. Therefore, we have made refinement on the scenario and concept, to make collaborative presentation more practical in real world presentation.

From the feedback of previous prototype, we have refine some concept and scenario of collaborative presentation.

Focus in Meeting and Conference

We have chosen two scenarios as our main focus: meeting and conference, and designing the interface and user experience focusing on those scenarios. The former scenario require strong communication and collaboration, and would fit to collaborative presentation concept. The latter one is our originate, which is similar to meeting with large group of people. Detail of deployment scenario for Group Meeting and Conference are discussed in subsection 2.3.

Focus in Moderator and Presenter

Under the scenarios mentioned, we decided to only focus in collaboration between

presenters. In those scenario, viewers is a minority and they would not have the right to speak in most cases. In case viewers need to express their opinion, they can still register with moderator and act as presenter in a short period of time.

1.3.2. Completion of Presentation Flow

A presentation work flow could be divided into three stages: preparing materials, distributing materials and presenting materials.

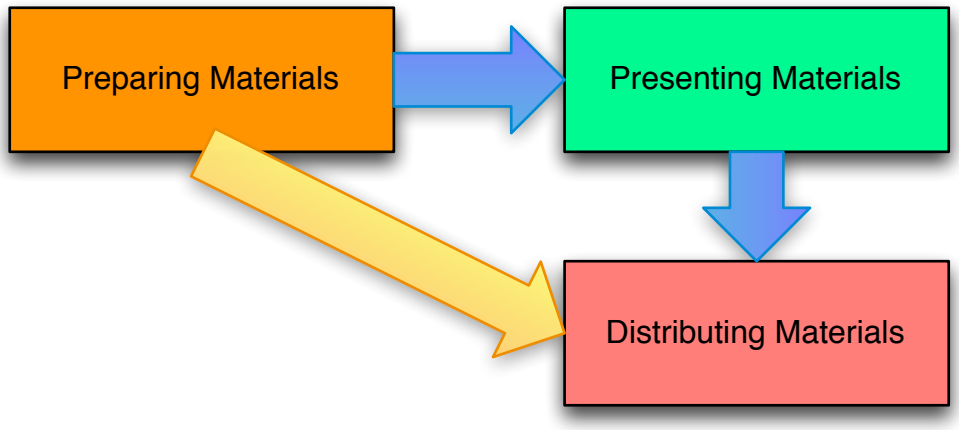


Figure 1.1.: Work Flow of a Presentation

We have been focusing on the stage of presenting materials for a long time. However, we did not address of preparing the presentation material, as well as distributing it. In the previous semester, the presentation materials are composed of multiple image files together with an index file, which is difficult to distribute or manage. Therefore, we are going to consider how to create the materials, as well as distributing the materials in a convenient way.

We have thought about developing a new format of presentation content for our application, but it is not friendly our existing presentations. Therefore, we start explore some existing presentation format and supporting the format in our application.

Mainstream presentation content creation software are Microsoft PowerPoint, Apple iWork Keynote and Google Docs presentation. Those softwares are having their own format, such as `.ppt` , `.pptx` for Microsoft PowerPoint, and `.key` for Apple Keynote, while Google Docs also support exporting to `.ppt` . In addition, all of those could export presentation content to `.pdf` . Considering `.ppt` and `.pptx` format, both of them are in binary and Office Open XML format, which need a large effort for parsing without sufficient of documentation. On the other hand, despite PDF file are binary, iOS API could parse them and render each page into image. There are also external libraries and engines that could parse PDF files.

Apart from creation of presentation, PDF could have annotation which could be in form of text, path drawing, regular shapes etc. Those annotation are useful for expressing idea during meeting and presentation, which is similar to the drawing pad from the previous prototype. Moreover, the annotation is able to save within the PDF file, and allow further distribution and review afterward.

As `.pdf` is easy to create, with common presentation creation software, and handy to distribute, as well as popular for exchanging nowadays. Therefore, we decided to support PDF file in the applications.

1.3.3. Enrichment of Presentation Content

Portable Document Format (PDF)

A PDF file could include the following types of element:

- Text
  - Embed with font

  - In different encoding

- Raster graphics
- Vector graphics
- Annotation

Supporting PDF as presentation content allow users to enrich slides with the above elements, as well as enhancing the quality of slides.

In order to speed up the implementation process, we used PSPDFKit, a third party framework, as PDF parser and renderer. The PSPDFKit provide functions for decoding, parsing of a PDF document, a view controller that show the rendered document and control of page view, as well as annotation editing functions. We are using a free trial of the framework in our app prototype.

Using the PSPDFKit, we then focus on integrating the view controller and those functionalities with our app, demonstrating collaborative presentation concept with PDF presentation. The main focus of integration including sending and exchanging the PDF documents, synchronization with moderator's device as well as external display. The implementation detail will be discussed in chapter 5.

Video Content

Another enrichment feature is to integrate video playback into the presentation content. As it is a common feature provided by common presentation formats, it it proved that video is an essential element in nowadays presentation. Our goal is to allow presenters to display video clips prepared by presenters on the projector screen, while they can

remotely control the playback of the clips. Challenges exist when introducing such kind of feature in collaborative presentation.

Exploring PSPDFKit API, we found it is possible to embed video link inside the PDF file. However, it is using PSPDFKit own protocol and format which is not compatible or readable by other application. Therefore, we decide not to use the built-in PSPDFKit function to play video. Thus, we have to design our own implementation in order to introduce this feature into our application.

Since video files are generally large in file size, while handling them are resource demanding, both in terms of networking and computational power. Moreover, the presentation content will not be available on the moderator side until the actual presentation is carried out, video playback on the remote side will not an easy task if we have to ensure smooth video playback while minimize any performance effects to the presentation flow. New controlling mechanisms and video content delivery methods are essential to complete this task.

Chapter 2

Collaborative Presentation

2.1. Concept and Terminology

Collaborative presentation is trying to make more people could shout their voice out during the presentation, as well as providing a simple way for multiple people taking turn in presenting their materials.

Firstly, we categorized people presence in a presentation into three major types: moderator, presenter and viewer.

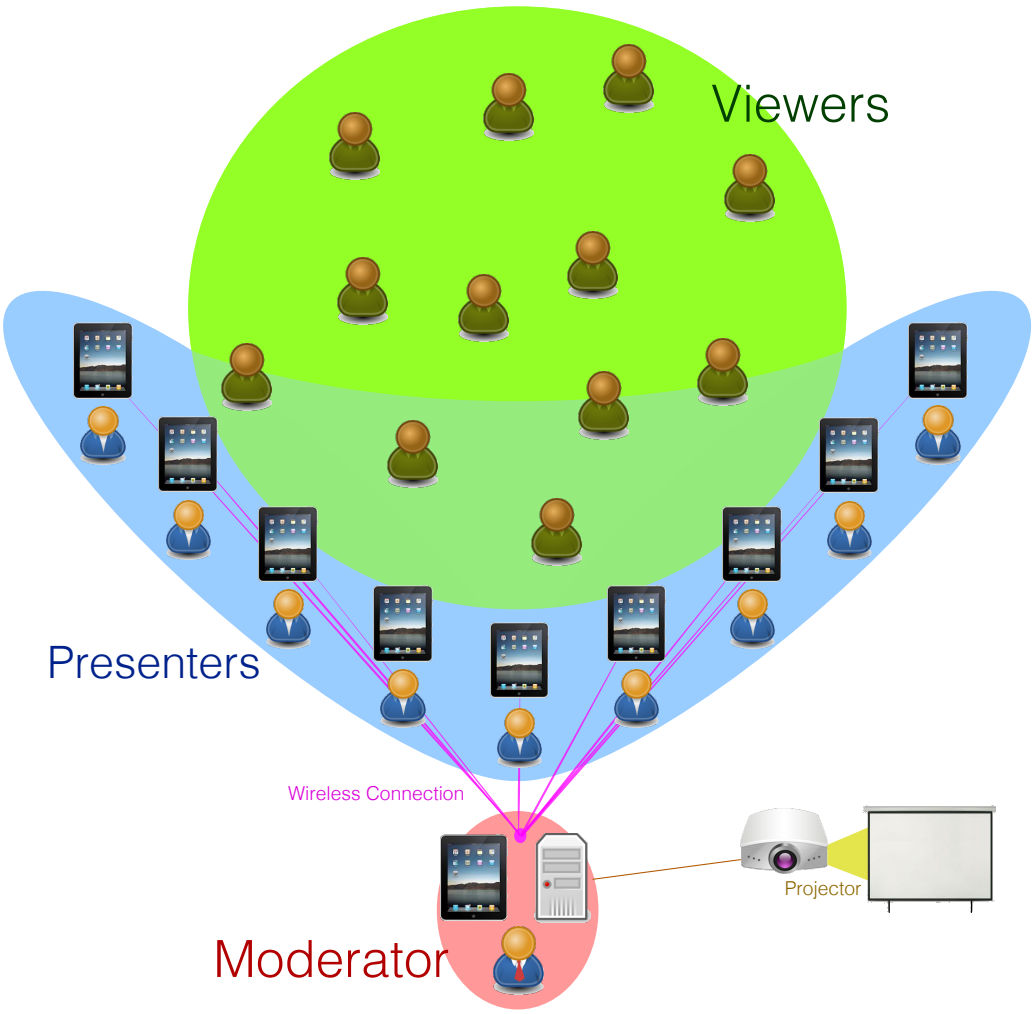


Figure 2.1.: Collaborative Presentation Overview

2.1.1. Roles

Moderator is unique in a presentation. The duty of moderator is to monitor the presentation, as well as coordinating the presenters with other presenters. The device of moderator also act as a server, receiving connections from presenters' device. In addition, moderator is connected to the external display such as monitor or projector, showing presentation content. Moreover, moderator is a subset of presenter, which is also allowed to do presentation.

Presenter is a group of people, of which are going to present their material during the presentation. They will bring their own material to attend the presentation. Presenter who is presenting is considered as active presenter, while the remaining those are inactive presenter.

Viewers are people who attend to the presentation without their own content. The idea of allowing viewers involve more in presentation is shelved in order to focus on presenter at preliminary stage. Therefore, they will only listening to presentation and may only give short comment.

2.1.2. Connections

The connections between presenters and moderator, or among presenters are wireless. Wireless connection is available in most of hand-held devices Including smartphone and tablets. This benefit in allowing presenters to have physical movement during presentation, or having interactive presentation with viewers.

However, the connection between moderator and external display have to be in wired. It is possible that to have streaming to external display using AirPlay or Wi-Fi Miracast, but the performance and fluency may be affected. Moreover, the bandwidth of single moderator would be too large if we use wireless streaming in addition to presenters' connection. The connection might be by 30-pin or lightning to VGA or HDMI for Apple devices.

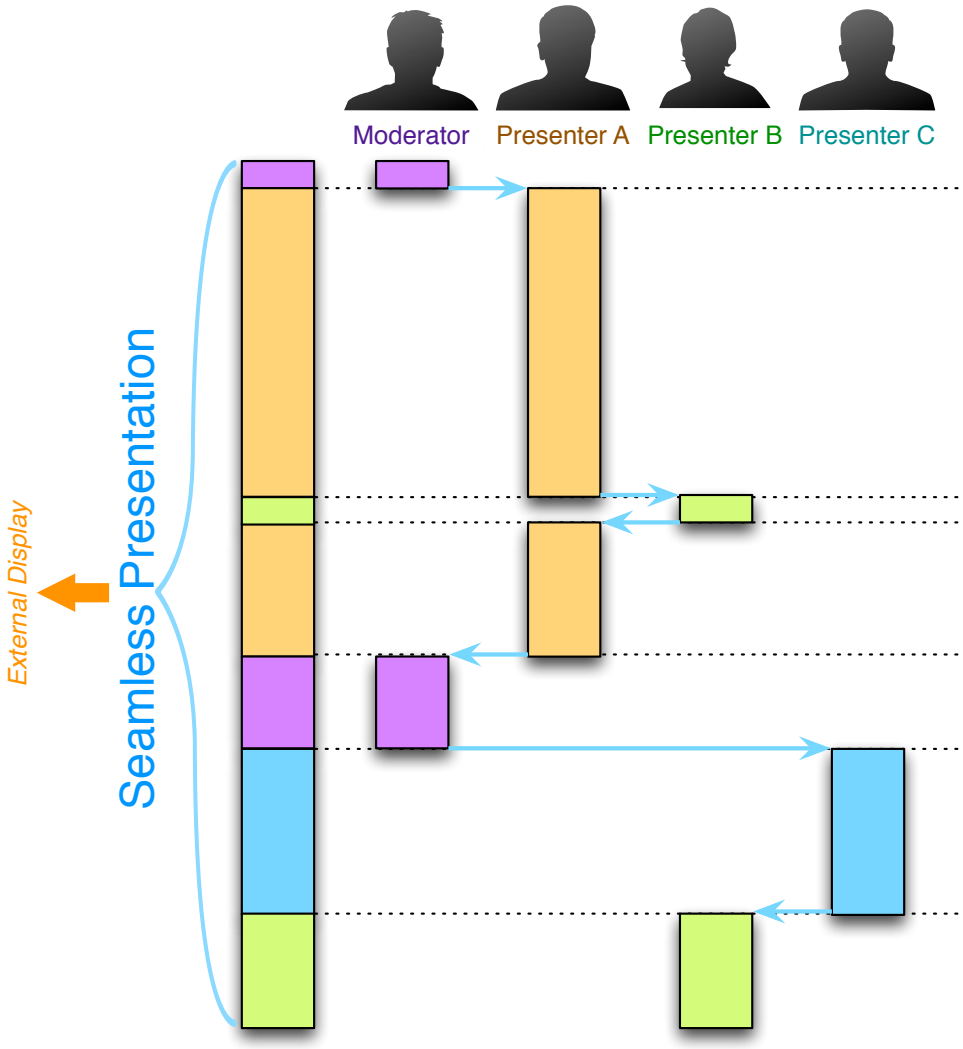


Figure 2.2.: Collaborative Presentation Flow

2.1.3. Presentation Flow

2.2. Features

2.2.1. Seamless Handover of Presentation Control

When there are multiple presenters in a presentation, it is troublesome when a active presenter handover the presentation to another presenter. Regarding the presentation

content are not merged together, presenters may have to reconnect their laptop with the external display, or switching presentation content to another file. Despite the time for those actions are short, it is still affecting fluency of presentation. Technical issues always pop up while the transition action as well. Another solution is only using one computer with single presentation slides file, which this require presenters to gather their materials before the presentation. However, there do not have time for such merging process in many scenario.

With seamless handover of presentation control, presenters could pass the control to another presenter easily without disturbance. When a presenter finish his own section of presentation, he / she only need a click, notifying moderator that the section is finished. Once moderator receive the signal, another presenter is chosen to present the next section. The presentation materials of the next presenter is send to moderator device in real-time via wireless connection, and can start the presentation in a short time.

The seamless handover of presentation control brings the following advantages: do not need wired connection for presenter, do not need any merging and preparation of presentation content needed before presentation, handy handover of presentation control, and improve of presentation fluency when passing presentation control.

2.2.2. Temporary Presentation Control Passing

This feature is a variation of seamless handover of presentation control. With this feature, active presenter could invite another presenter to give a short feedback and opinion on current presentation.

The main difference of temporary presentation control passing from handover of presen-

tation control is that the permission is grant by active presenter instead of moderator. During the presentation, active presenter may receive requests from another presenter, or actively invite another presenter to give feedback. However, this may cause disturbance of current presentation and implementation should provide several methods to avoid this.

A idea is to provide a "do not distrub" function, thus the presenter could focus on presentation and look for any pending requests when the presentation is paused. On the other hand, implementation could limit only one request could be sent from single presenter, or making limited number of request being visible for active presenter.

2.2.3. Presentation Functionalities

Basic presentation functionalities should also be provided apart from collaborative presentation. Following listed few functionalities from current presentation software, and need to be implemented together with collaborative presentation features:

- Showing slides or presentation content
- Support of external display
- Drawing on presentation slides

2.3. Deployment Scenario

We set the deployment scenario focusing to involving and need communication. The following are examples:

2.3.1. Group Meeting

One of the most common scenarios that suitable to deploy collaborative presentation will be a small group meeting that happen every day at offices and campuses.

In this case, we do not fix a size of participants, but only considering a controllable size for moderator. The chairman of meeting, who will have to control the meeting flow and right of speak of participants, need to be the moderator, while other participants are act as presenters. Moreover, the external display may show information, such as agenda, report as well as related materials for discussion.

At the beginning of meeting, moderator connect his/her own device to external display, and connect to an access point wirelessly. Other presenters could connect to the same access point, and therefore could connect and register to moderator's device with local network IP address. Once presenters are registered, moderator can see all connected presenters. Moderator could start the meeting, and pass the control to other participants based on agenda and meeting progress.

During the handover processes, no external files needed to be exchanged between the presenters and moderators device. This saves the overhead when performing handover actions in traditional presentation, either transmitting files between the two devices or reconnecting the external displaying unit. Once the presenters have been granted the control, they can use their own machines and prepared material just have the same experience as their devices have been directly connected to the projectors.

2.3.2. Conference

Another suitable deployment scenario would be council meetings or conference meetings in larger scale.

In a conference, moderator is the president while other attendees are presenter. President have to control and manage the flow of discussion, allowing attendees to speak one by one. With collaborative presentation, it is handy for president to pass the control to another presenter.

Once the presentation control is passed to a presenter, he / she could present materials to all other attendees. If other attendees have opinion or feedback on the presentation content, it could ask for presenter, as well as the moderator for temporary permission.

In this case, the moderator could have a centralized control of the meeting progress, while other attendees could have their chance to present their materials, without exchanging files through wires. Collaborative presentation help in this scenario much.

However, there should be a point need to be mentioned. As the attendee size might be large in this case, the moderator might be overloaded by the large amount of requests, as well as for active presenter. Therefore, limit of request number and “do not disturb” mode should be available for this scenario.

Chapter 3

System Design

3.1. System Overview

Please refer to Figure 3.1

3.1.1. Class Diagram - Moderator

Please refer to Figure 3.2

3.1.2. Class Diagram - Presenter

Please refer to Figure 3.3

3.2. Presentation Content

3.2.1. PDF

The PDF is done with PSPDFKit framework. Functionalities of the framework is rich, but we may not need all of those functions. Following is a table list all features from PSPDFKit, and features we needed in our application:

Funcions	**Features in our application**
Annotation	✓ Drawing on slides
Zooming and Moving	✓ Focus in presentation
Text Search	✗
Multimedia	✗
Links	✗
Encryption	✗
Bookmark	✗
PDF Manipulation	✓ Export single PDF for whole presentation
Internal Web Browser	✗
Multiple Tabs	✗

Table 3.1.: Summary of Functionalities of PSPDFKit

From Table 3.1, we could easily conclude the functions we used from PSPDFKit and adopt to our application.

Annotation: Drawing on Slides

Annotation features is an important concern in our applications. From Fall 2012, we have implemented a drawing pad with arbitrary drawing features, allowing presenter to drop down important notes and emphasize points in the slides. We could make use of the annotation feature to have similar functions of drawing pad, and export the drawing to the same PDF file.

Zooming and Moving: Focus in Presentation

Zoom is helpful for presenters to focus to a region that is presenting. In addition, as PDF support vector graphics, zooming is also useful to view a small elements in PDF.

PDF Manipulation: Export single PDF for whole presentation

With seamless presentation, each presenter could have their own presentation slides. It is easy for each presenter to distribute their own slides in PDF format by their own. However, it is difficult to have a single PDF file that include all slides from each presenter in presentation sequence. Therefore, with PDF manipulation feature, we could export single PDF file for the whole presentation after each presenter finish their part. Moderator or the organizer may easily distribute the whole presentation with a single PDF file.

3.2.2. Video

In order to play presenter content through the moderator, using currently available infrastructures, the video content, either part of the content or the whole video clip, has to be available on the moderator before it can be played on the projector screen

through the moderator device.

The major challenge is how video contents from presenters can be effectively delivered to the moderator device while without affecting much of the presentation flow and the overall performance of the whole system. Also, playback control messages are also required to send from the presenter devices to the moderator, so that the video playing on the moderator side is under the respective presenters control.

Content Delivery

Regrading these requirements, the following video content delivery strategies have been considered.

Direct Transmission of Video Contents

During the presentation is being carried out, the presenter can select a video clip orginally prepared and saved on the presenters device. Then the presenter send the whole video clip to moderator device. After the video clip have been transmitted, the moderator will spawn out the media player and wait for the presenter to start playing the video. The media player disappear after the video is finished and resume back to the static presentation content.

This approach is easy to implement, as the whole video file can be treated as a lump of data and send to the moderator using a simple TCP connection. Once the download completed, the video clip will be saved at the local storage. Since the video is locally stored, video playback quality and smoothness can be ensured. On the other hand, sending the video file is time consuming and resource consuming, especially when the video quality is high or the network condition is unstable. These two drawbacks can

greatly affect the user experience and system performance if the application is poorly implemented.

Video playback by streaming through external video streaming servers

The presenter can first upload their video content to any external video streaming servers which support Apple HTTP Live Streaming (Apple HLS) standard. During the presentation, the presenter can send the HLS-format playlist to the moderator, then the moderator will load this playlist into the media player and start streaming the video.

The network overhead is low for presenters, and the video is likely to be readily available since the video can be started when the enough video buffer is downloaded from the video streaming server. The drawbacks would be the video quality is likely to be downgraded and the video playback may become lag if the connection between moderator and streaming server is poor.

Video playback by streaming through self-contain video streaming server

The moderator works exactly the same as the method using external video streaming servers. The difference is that all the encoding and streaming workflow of producing the video streams will be done on presenter device. The original video is loaded into the device, when the presentation starts, the device encodes the video and start the HTTP service. When the moderator receives the HLS-format playlist, it will get the video data stream from the presenter and start playing when there is enough buffer downloaded.

The main advantage of this approach is that no external streaming server is needed,

and the presenter do not need to upload their content to the media servers first, as the video streams will be prepared by the application on the device. Also, as local network is usually more stable then Internet, the video playing on the moderator side is less likely to be choked. Still, this approach introduce large networking and computational overheads to presenters device, as encoding videos and hosting HTTP services are power-consuming tasks. Also, no current implementations or libraries are available for iOS devices to encoding video or hosting a HTTP server. So theoretically this approach should work, but implementation would be very complex and difficult.

Playback Mechanism

Apart from the delivery of video contents, we have to design a video playback controlling mechanism for presenters to play, pause, stop or seek their videoes. New controlling protocols and user interfaces are needed in order to provide this function.

Also, as the moderator have the right to interrupt any ongoing presentations, there is essential designing another mechanism for the moderator to stop the video and inactivate the current presenter.

3.3. External Display

The moderator would connect to external display, and show the slides on external screen. While the projector showing slide, screen on device should also show the slide so the user can focus on the device screen and audiences, without frequently looking at projection. In addition to slides and presentation content, there should be presentation control and configuration detail shown only in device but not projecting to screen. Therefore, we have to make presentation content synchronizing in device and screen,

as well as having control only showing in the device.

Another concern is the resolution and resizing. Resolution of iPad, iPad 2 and iPad Mini is 1024×768 while resolution of the New iPad is 2048×1536. However, typical resolution of projector or external screen is 1280×1024 and 1920×1080. We separate the problem into two cases: same aspect resolution and different aspect ratio. If the aspect ratio is the same, simply resize the image by multiplying size by a constant ratio. However, if the aspect ratio is different, we need to handle re-calculate the size in order to maintain the aspect ratio. Moreover, center the image and handling of extra space is another concern when making best-fit resizing.

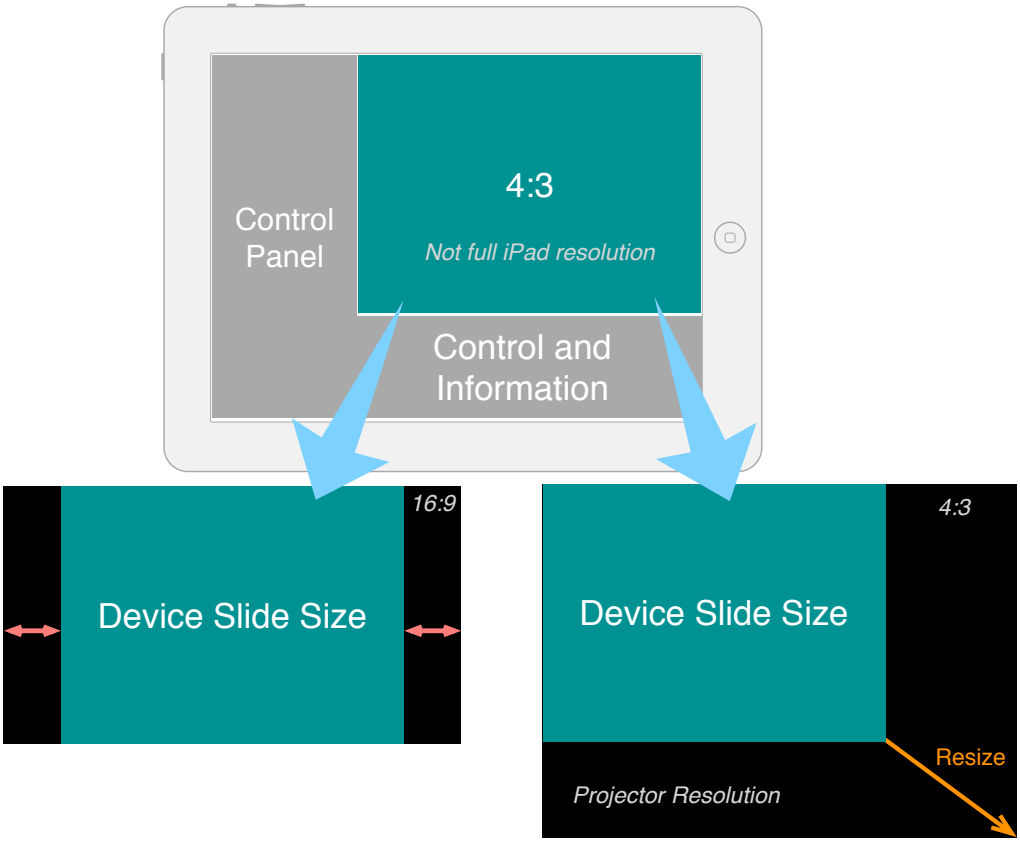


Figure 3.4.: Scaling and Fitting for External Display

3.4. Inter-Devices Communication

As a client-server system, we have to consider about connection between clients and server. Regarding networking and Open Systems Interconnection (OSI) model, there are 7 abstract layers in communications system or network.

Despite it is an abstract concept, it is still worth to make decision about message transmission based on few layers which is more in practical, including data link layer, transport layer and application layer.

Regarding the data link layer, there are much implementation including Wi-Fi and Bluetooth, which are both native supported by iOS devices. We have considered both data link interfaces to implement the inter-device communication, and we decided to use Wi-Fi as our data link layer interface.

There are two main implementation of transport layer, user datagram protocol (UDP) and transmission control protocol (TCP). We use TCP instead of UDP because TCP could ensure that received data is correct. This is important when user send slides from own device to server. In addition, this can also avoid a possibility of error and bug during development process.

Application layer protocol is important for message being understand by both devices. As it is self-defined, the information would be more and we will discuss it in the next session.

3.4.1. Application Layer Protocol

Sending message on network is a costly operation, therefore we have to minimize the size of message by considering the protocol in byte.

There are several type of message being sent. During the design of protocol, we should make the type be easily differentiated and distributed to other object for processing.

Header

We decided to make an application layer header for easy distinguishing command and presenter. The header is total of 8-byte long, including command, presenter ID, checksum and message size.

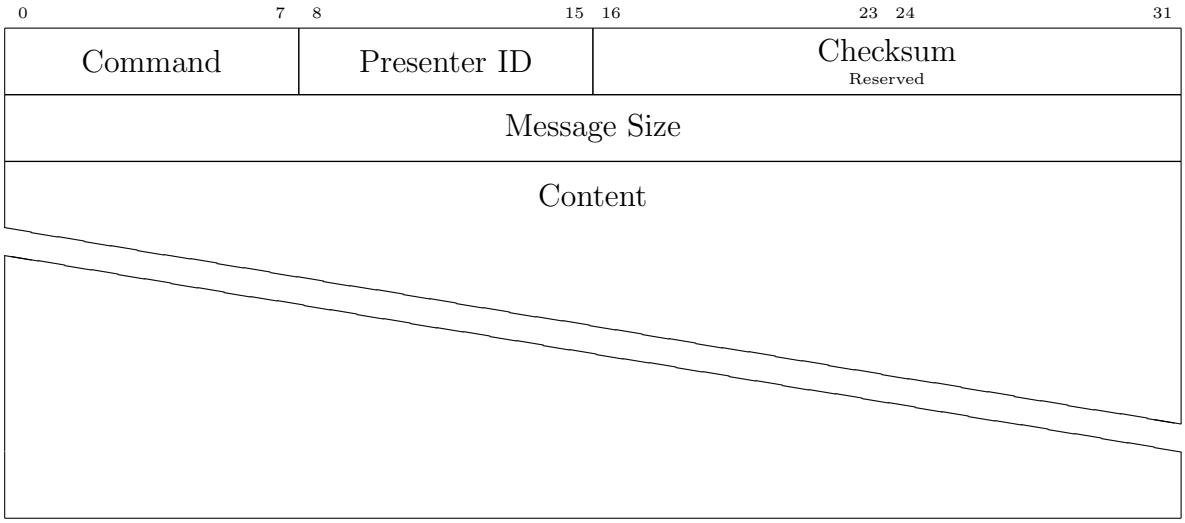


Figure 3.5.: Application layer protocol packet structure

Specifications of Field

Regarding the **presenter ID**, it is a single byte unique value for each presenter. The range of presenter ID is from 1 to 254 inclusively, and 0 is reserved for server, 255 is reserved for unregistered presenter. In usual, presenter ID in a message would not be

0 as the value in message means the client communicate with server. The only case for presenter ID equals 0 is moderator is broadcasting message to all registered presenter.

Checksum is a two-byte field reserved for message integrity. However, it is not used in current stage as TCP guaranteed correctness of received data. This could be useful if some message are sent via UDP.

Message length is a four-byte integer in network byte ordering, representing the length of whole message including the eight-byte header. By referring to this value, we ensure the whole message is received from stream before calling the delegate handling method.

Command is used to distinguish type of message. Adding this field ease the difficulty in partitioning message. Each type of message is associated with a single-byte command value. For detail please refer to Table 3.2.

Type	Action	From	Command
Register	Request	Client	0x01
	Success Response	Server	0x02
	Failure Response	Server	0x03
Unregister	Request	Client	0x04
	Response	Server	0x05
Control Permission	Request	Client	0x06
	Response of Request	Server	0x07
	Grant Permission	Server	0x08
	Withdraw Permission	Server	0x09
Control Signal	Request	Client	0x0C
	Success Respond	Server	0x0D
	Failure Respond	Server	0x0E
Video Playback	Request	Client	0x20
	Success Response	Server	0x21
	Pause Request	Client	0x22
	Pause Success Response	Server	0x23
	Stop Request	Client	0x24
	Stop Success Response	Server	0x25
	Goto Request	Client	0x26
	Goto Success Response	Server	0x27
	Prepare Send Request	Client	0x28
	Prepare Send Response	Server	0x29
	Prepare Send	Client	0x2A
	Prepare Send Complete Response	Client	0x2B
Sending PDF	PDF Data	Client	0x30
	Response	Server	0x31
Annotation	Request	Client	0x34
	Response	Server	0x35

Table 3.2.: Command List Table

Message Sending Chart and Callback

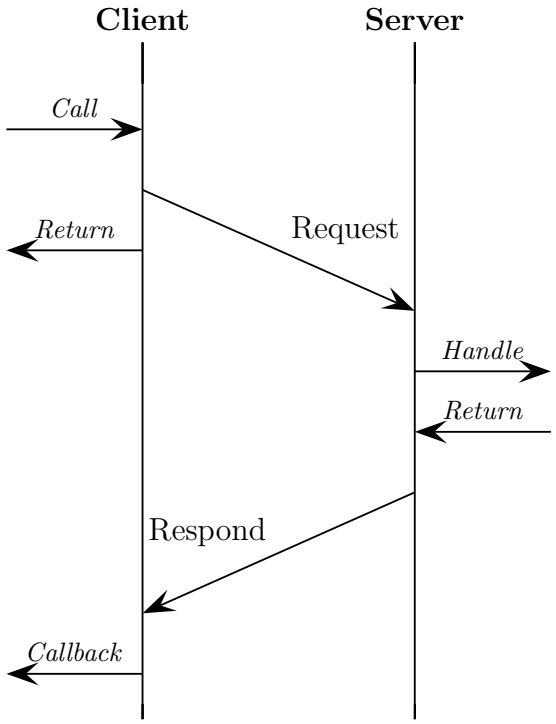


Figure 3.6.: Message Sending Chart for Most Message

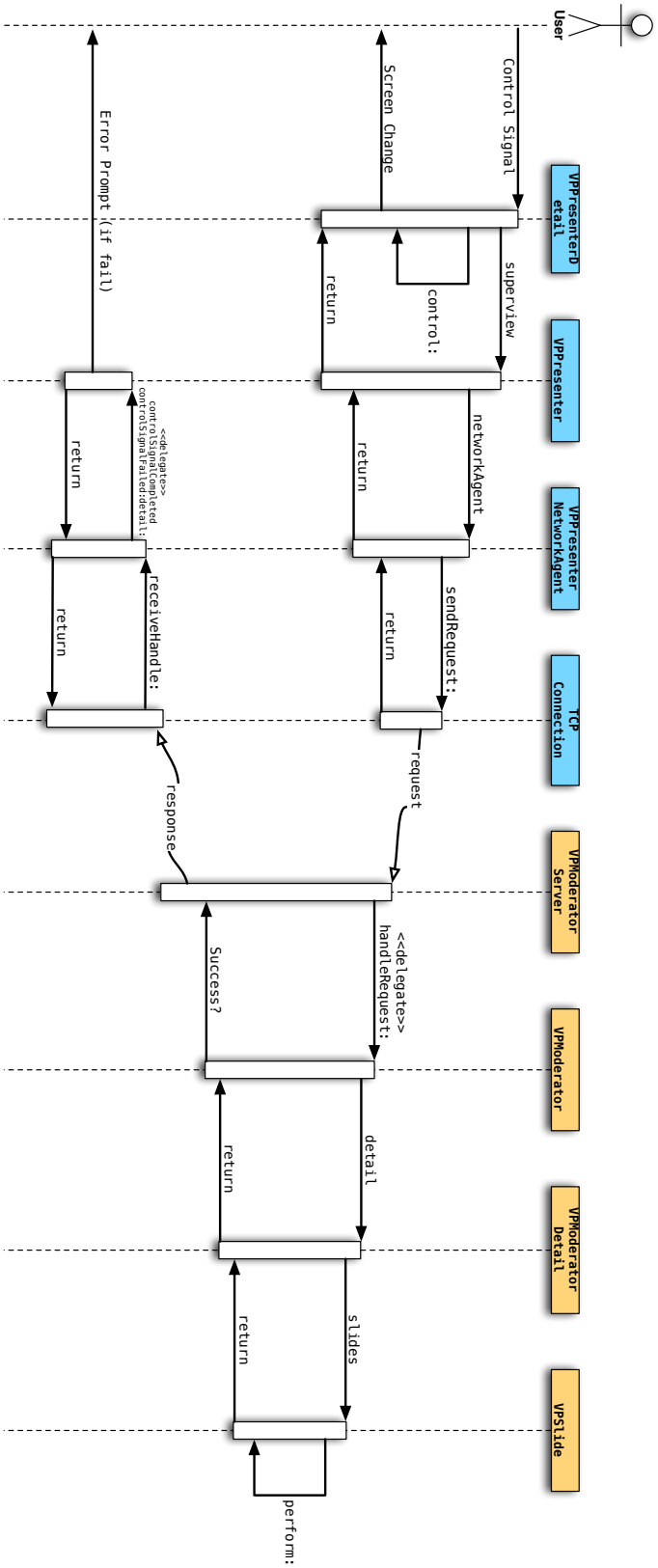


Figure 3.7.: Sequence Diagram of Handling Requests

Message Specification

Following list all message, its message structure and field description. Packet structure diagram could be found in A.

Register

Command	`0x01`
Direction	Client ➔ Server
Argument Count	2
Argument 1	Length of Name *(4 bytes)*
Argument 2	Presenter Name *(Vary length)*
Package Structure Diagram	*Figure A.1*

Table 3.3.: Register Request

Command	`0x02`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.2*

Table 3.4.: Register Success Response

Command	`0x03`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.3*

Table 3.5.: Register Failure Response

Unregister

Command	`0x04`
Direction	Client ➔ Server
Argument Count	0
Package Structure Diagram	*Figure A.4*

Table 3.6.: Unregister Request

Command	`0x05`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.5*

Table 3.7.: Unregister Response

Control Permission

Command	`0x06`
Direction	Client ➔ Server
Argument Count	0
Package Structure Diagram	*Figure A.6*

Table 3.8.: Control Request

Command	`0x07`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.7*

Table 3.9.: Control Response

Command	`0x08`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.8*

Table 3.10.: Grant Control

Command	`0x09`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.9*

Table 3.11.: Withdraw Control

Control Signal

Command	`0x0C`
Direction	Client ➔ Server
Argument Count	2
Argument 1	Control Signal Type *(1 bytes)*
Argument 2	Control Signal Parameter *(3 bytes)*
Package Structure Diagram	*Figure A.10*

Table 3.12.: Register Request

The control signal is specified in Table 3.13

Type	Description	Parameters
0x01	Next Slide	
0x02	Previous Slide	
0x03	Jump to Slide	Slide Number *(1 byte)*
0xF0	Black Screen	As boolean of useWhite *(1 byte)*
0xFD	Handover Control	Presenter ID *(1 byte)*
0xFE	Withdraw Control	Counter *(1 byte)*
0xFF	Return Control	

Table 3.13.: List of Control Signal

Command	`0x0D`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.11*

Table 3.14.: Control Signal Success Response

Command	`0x0E`
Direction	Server ➔ Client
Argument Count	2
Argument 1	Error Code *(1 bytes)*
Argument 2	Error Detail *(3 bytes)*
Package Structure Diagram	*Figure A.12*

Table 3.15.: Control Signal Failure Response

Video Playback

Command	`0x20`
Direction	Client ➔ Server
Argument Count	2
Argument 1	Length of Video Filename *(4 bytes)*
Argument 2	Video Filename *(Vary length)*
Package Structure Diagram	*Figure A.13*

Table 3.16.: Video Play Request

Command	`0x21`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.14*

Table 3.17.: Video Play Success Response

Command	`0x22`
Direction	Client ➔ Server
Argument Count	0
Package Structure Diagram	*Figure A.15*

Table 3.18.: Video Pause Request

Command	`0x23`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.16*

Table 3.19.: Video Pause Success Response

Command	`0x24`
Direction	Client ➔ Server
Argument Count	0
Package Structure Diagram	*Figure A.17*

Table 3.20.: Video Stop Request

Command	`0x25`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.18*

Table 3.21.: Video Stop Success Response

Command	`0x26`
Direction	Client ➔ Server
Argument Count	1
Argument 1	Second *(8 bytes)*
Package Structure Diagram	*Figure A.19*

Table 3.22.: Video Goto Request

Command	`0x27`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.20*

Table 3.23.: Video Goto Success Response

Command	`0x28`
Direction	Client ➔ Server
Argument Count	0
Package Structure Diagram	*Figure A.21*

Table 3.24.: Video Prepare Send Request

Command	`0x29`
Direction	Server ➔ Client
Argument Count	1
Argument 1	Listening Port *(2 bytes)*
Package Structure Diagram	*Figure A.22*

Table 3.25.: Video Prepare Send Response

Command	`0x2A`
Direction	Client ➔ Server
Argument Count	4
Argument 1	Length of Video Filename *(4 bytes)*
Argument 2	Video Filename *(Vary length)*
Argument 3	Length of Video *(4 bytes)*
Argument 4	Video File *(Vary length)*
Package Structure Diagram	*Figure A.23*

Table 3.26.: Video Send

Command	`0x2B`
Direction	Client ➔ Server
Argument Count	0
Package Structure Diagram	*Figure A.24*

Table 3.27.: Video Send Complete Respond

PDF Data

Command	`0x30`
Direction	Client ➔ Server
Argument Count	2
Argument 1	Length of PDF Data *(4 bytes)*
Argument 2	PDF Data *(Vary length)*
Package Structure Diagram	*Figure A.25*

Table 3.28.: PDF Data

Command	`0x31`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.26*

Table 3.29.: PDF Data Acknowledge

Annotation

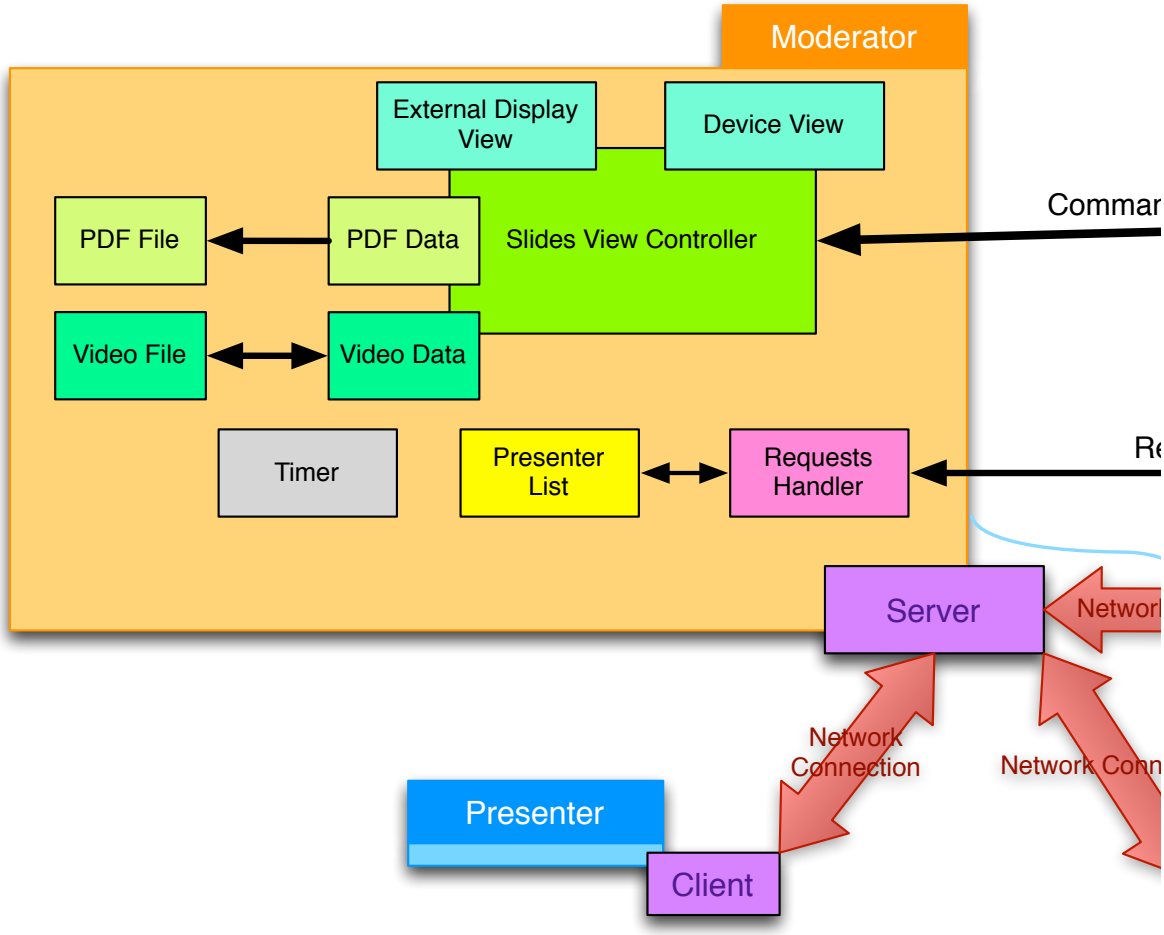
Command	`0x34`
Direction	Client ➔ Server
Argument Count	3
Argument 1	Annotation Type *(2 bytes)*
Argument 2	Data Length *(2 length)*
Argument 3	Data *(Vary length)*
Package Structure Diagram	*Figure A.27*

Table 3.30.: Annotation Request

Command	`0x35`
Direction	Server ➔ Client
Argument Count	0
Package Structure Diagram	*Figure A.28*

Table 3.31.: Annotation Response

Figure 3.1.: Overall Structure



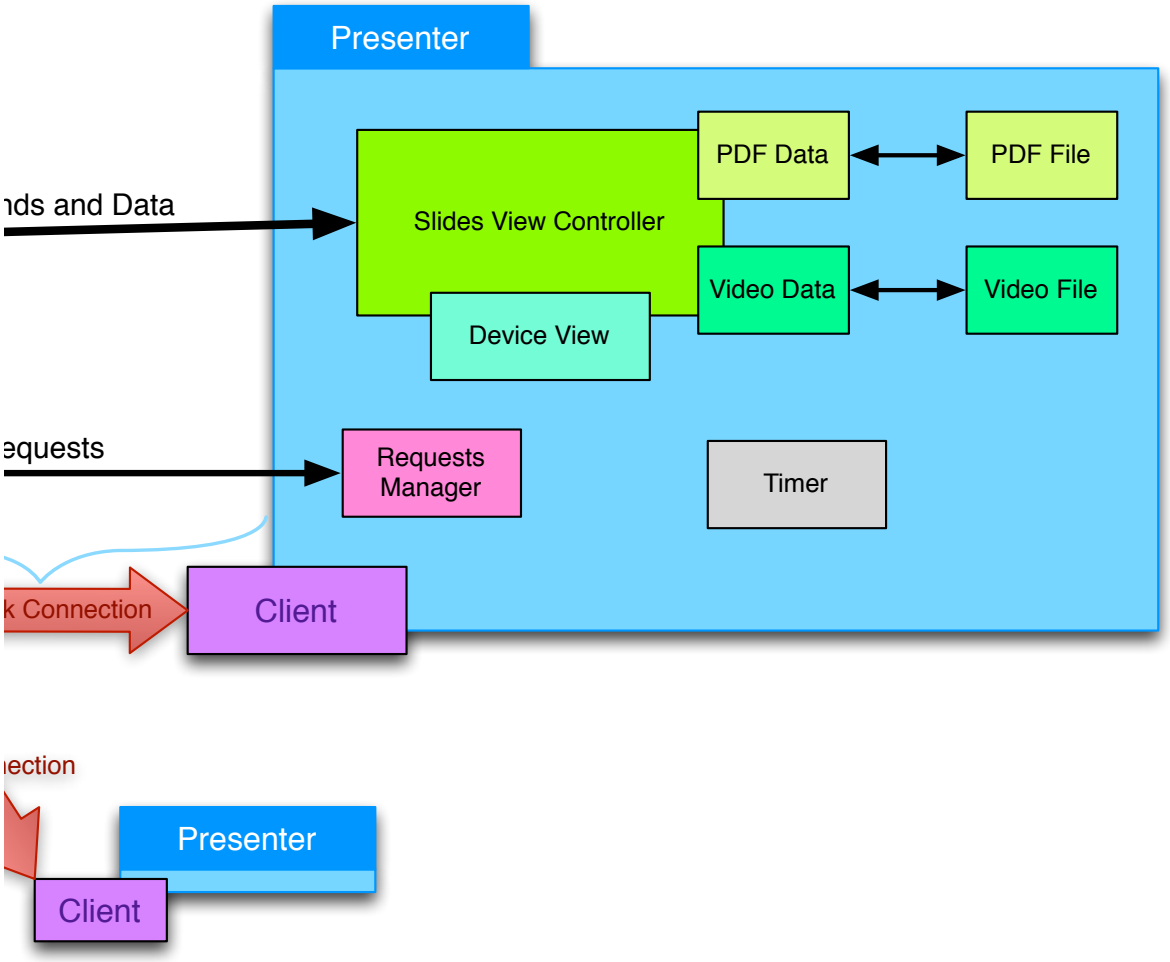
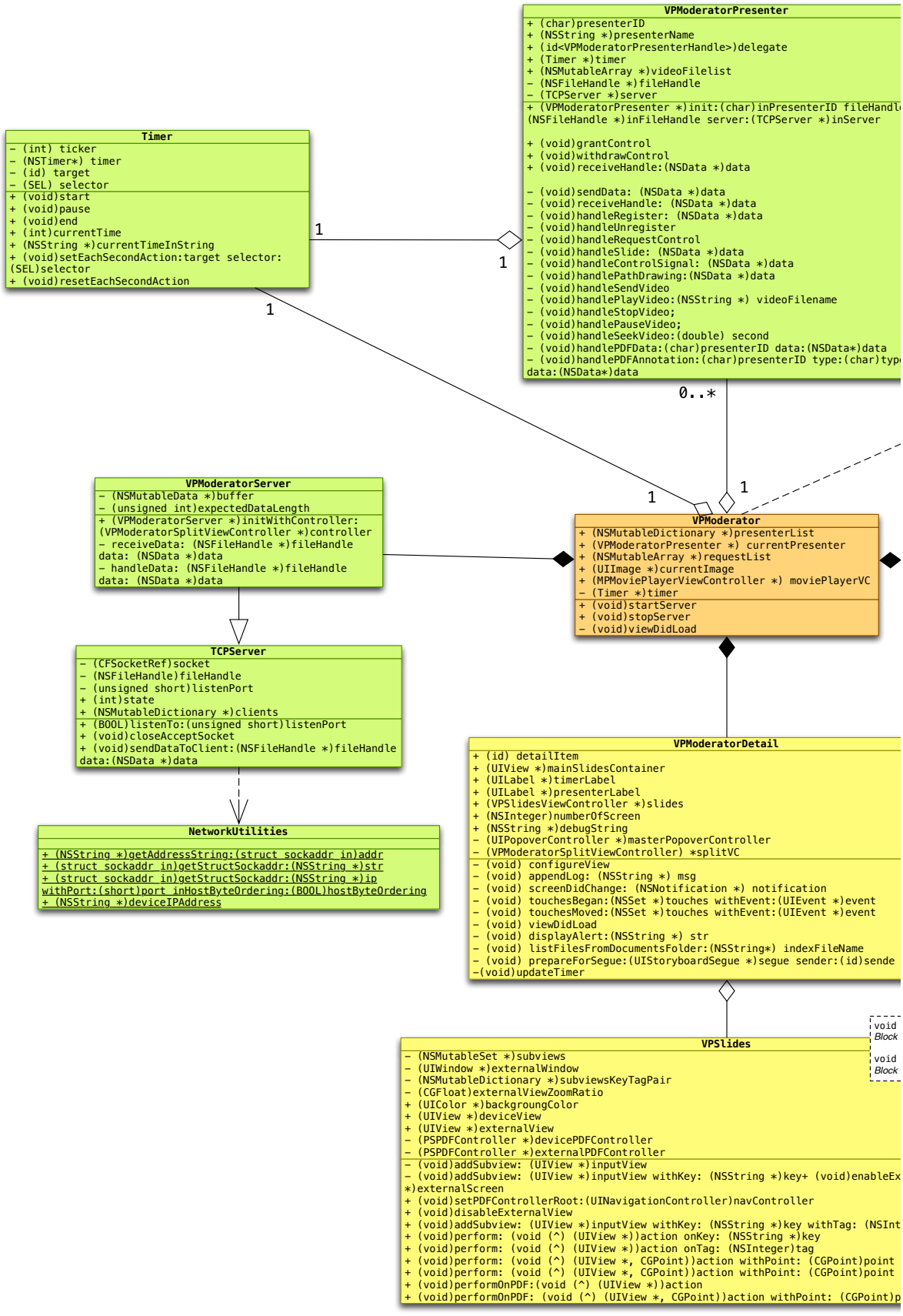

Presenter
PDF Data
PDF File
ıds and Data
Slides View Controller
Video Data
Video File
Device View
equests
Requests Manager
Timer
k Connection
Client
ıection
Presenter
Client

Figure 3.2.: UML Class Diagram of Moderator



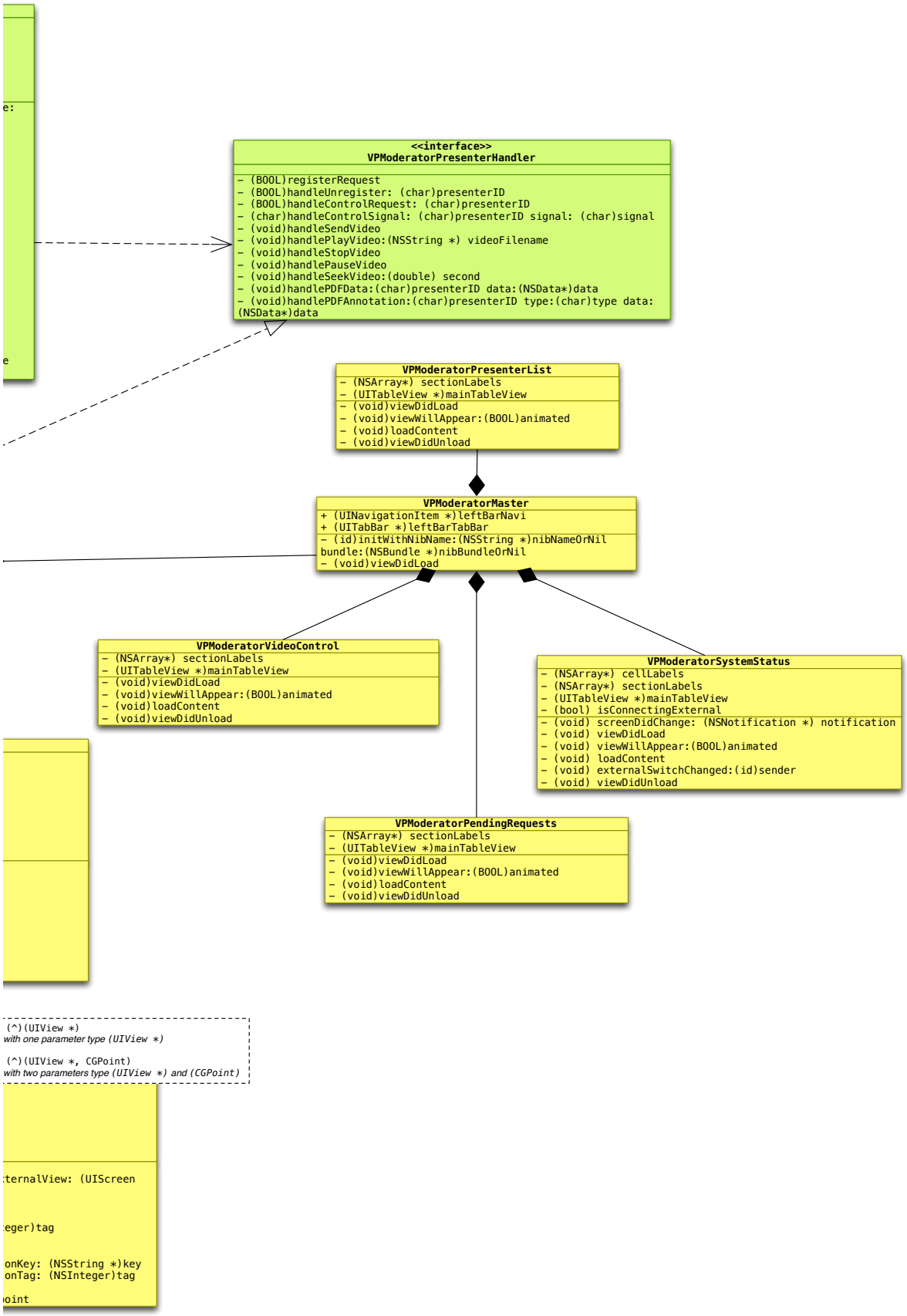
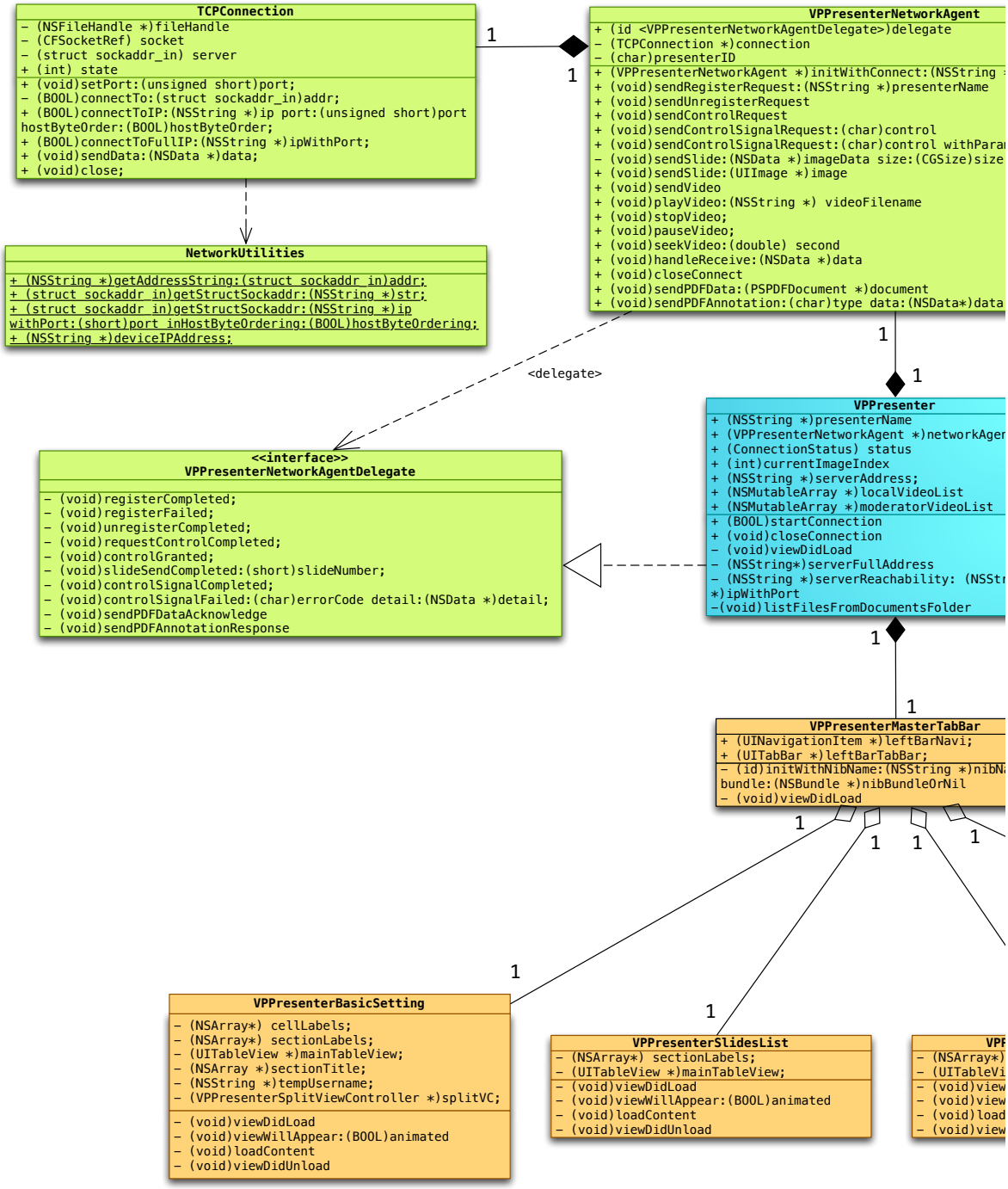
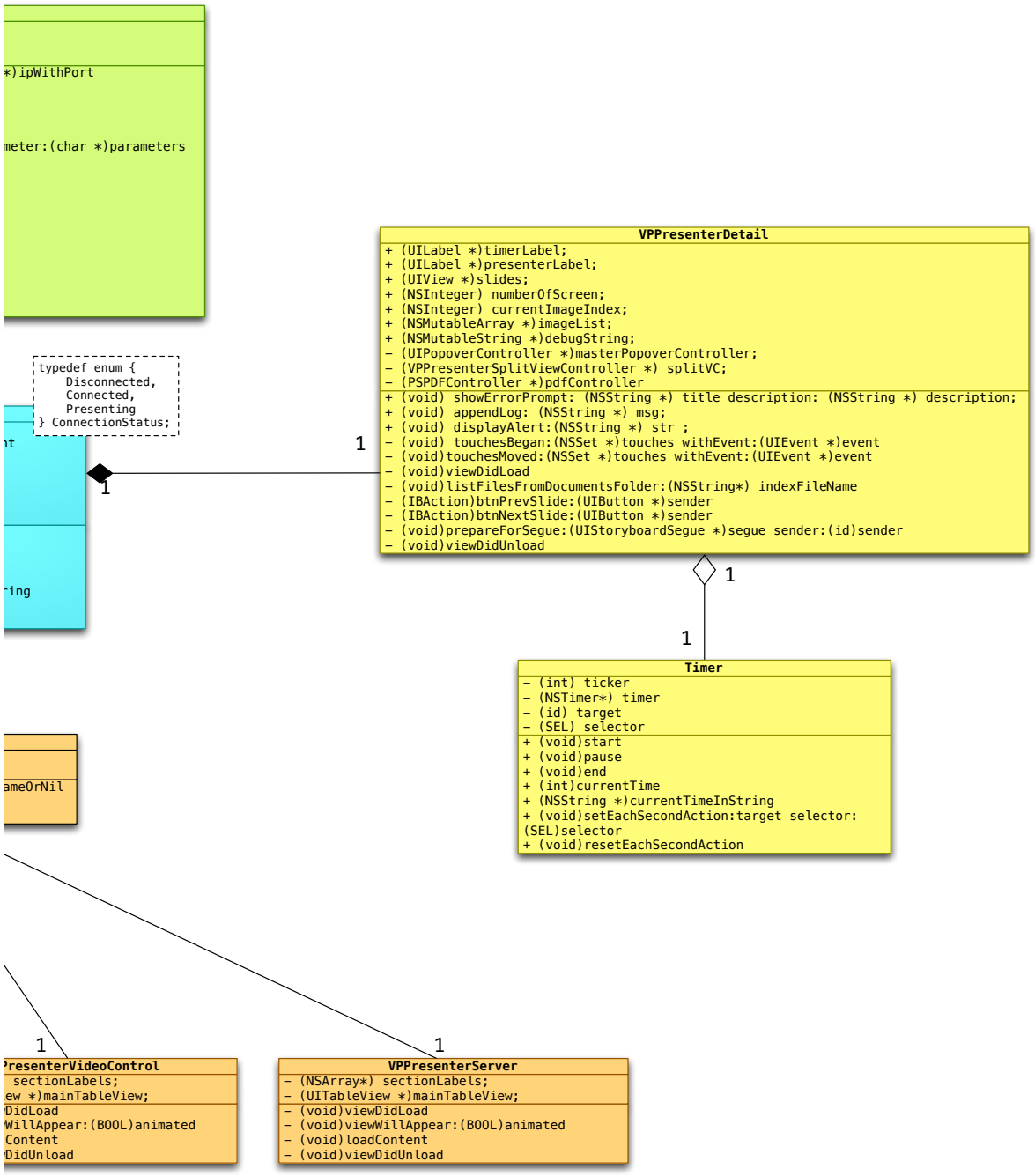

<<interface>>
VPModeratorPresenterHandler
- (BOOL)registerRequest
- (BOOL)handleUnregister: (char)presenterID
- (BOOL)handleControlRequest: (char)presenterID
- (char)handleControlSignal: (char)presenterID signal: (char)signal
- (void)handleSendVideo
- (void)handlePlayVideo:(NSString *) videoFilename
- (void)handleStopVideo
- (void)handlePauseVideo
- (void)handleSeekVideo:(double) second
- (void)handlePDFData:(char)presenterID data:(NSData*)data
- (void)handlePDFAnnotation:(char)presenterID type:(char)type data:
(NSData*)data
VPModeratorPresenterList
- (NSArray*) sectionLabels
- (UITableView *)mainTableView
- (void)viewDidLoad
- (void)viewWillAppear:(BOOL)animated
- (void)loadContent
- (void)viewDidUnload
VPModeratorMaster
+ (UINavigationItem *)leftBarNavi
+ (UITabBar *)leftBarTabBar
- (id)initWithNibName:(NSString *)nibNameOrNil
bundle:(NSBundle *)nibBundleOrNil
- (void)viewDidLoad
VPModeratorVideoControl
- (NSArray*) sectionLabels
- (UITableView *)mainTableView
- (void)viewDidLoad
- (void)viewWillAppear:(BOOL)animated
- (void)loadContent
- (void)viewDidUnload
VPModeratorSystemStatus
- (NSArray*) cellLabels
- (NSArray*) sectionLabels
- (UITableView *)mainTableView
- (bool) isConnectingExternal
- (void) screenDidChange: (NSNotification *) notification
- (void) viewDidLoad
- (void) viewWillAppear:(BOOL)animated
- (void) loadContent
- (void) externalSwitchChanged:(id)sender
- (void) viewDidUnload
VPModeratorPendingRequests
- (NSArray*) sectionLabels
- (UITableView *)mainTableView
- (void)viewDidLoad
- (void)viewWillAppear:(BOOL)animated
- (void)loadContent
- (void)viewDidUnload
(^)(UIView *)
with one parameter type (UIView *)
(^)(UIView *, CGPoint)
with two parameters type (UIView *) and (CGPoint)

Figure 3.3.: UML Class Diagram of Presenter




*)ipWithPort
meter:(char *)parameters
typedef enum {
Disconnected,
Connected,
Presenting
} ConnectionStatus;
VPPresenterDetail
+ (UILabel *)timerLabel;
+ (UILabel *)presenterLabel;
+ (UIView *)slides;
+ (NSInteger) numberOfScreen;
+ (NSInteger) currentImageIndex;
+ (NSMutableArray *)imageList;
+ (NSMutableString *)debugString;
- (UIPopoverController *)masterPopoverController;
- (VPPresenterSplitViewController *) splitVC;
- (PSPDFController *)pdfController
+ (void) showErrorPrompt: (NSString *) title description: (NSString *) description;
+ (void) appendLog: (NSString *) msg;
+ (void) displayAlert:(NSString *) str ;
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
- (void)viewDidLoad
- (void)listFilesFromDocumentsFolder:(NSString*) indexFileName
- (IBAction)btnPrevSlide:(UIButton *)sender
- (IBAction)btnNextSlide:(UIButton *)sender
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
- (void)viewDidUnload
1
1
1
1
Timer
- (int) ticker
- (NSTimer*) timer
- (id) target
- (SEL) selector
+ (void)start
+ (void)pause
+ (void)end
+ (int)currentTime
+ (NSString *)currentTimeInString
+ (void)setEachSecondAction:target selector:
(SEL)selector
+ (void)resetEachSecondAction
1
1
VPPresenterServer
- (NSArray*) sectionLabels;
- (UITableView *)mainTableView;
- (void)viewDidLoad
- (void)viewWillAppear:(BOOL)animated
- (void)loadContent
- (void)viewDidUnload

Chapter 4

User Interface and Experience

4.1. Initial Approach and Proposals

Last semester, we have proposed some initial designs of the user interface layout, the first design is a primitive layout which looks simple but unorganized. Thus we discovered the need of user interface design.

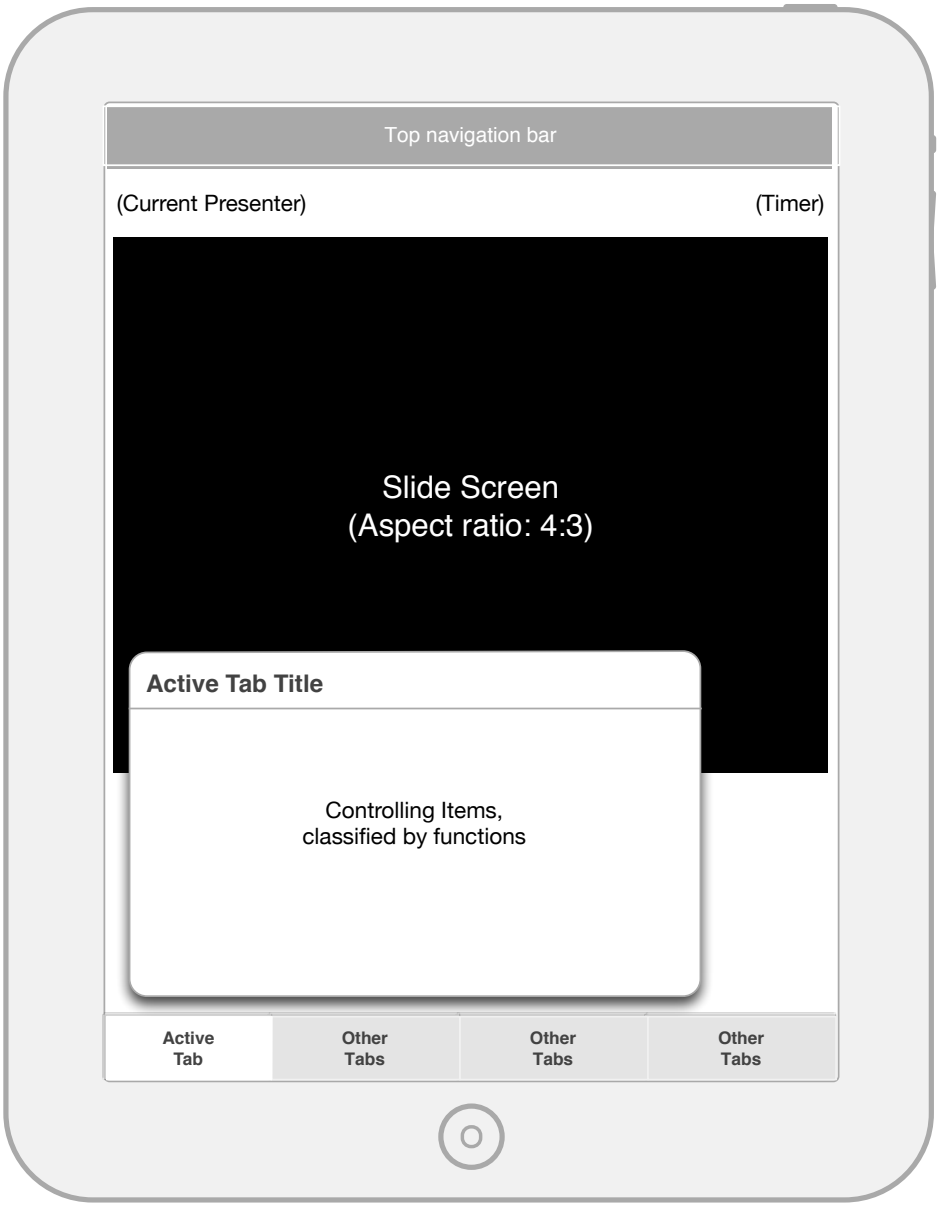


Figure 4.1.: Initial Portrait Interface design

Then we started to try different view elements. Including navigation bars on the top of the screen and tab bars at the bottom of the device screen, and we drafted the second proposal.

The tab bar consists of some tab bar items, each of them was responsible for a category of functions, for example the status and switch for external monitors can group into

the same group named Screen. These items would only be visible when the respective tab bar button was touched, then a pop up dialog box would appear and overlay on the screen. This dialog box would fade out if the user tab outside the window of the screen

Still we discovered a major defect of this interface design, which that the dialog boxes of the active tabs would partially cover the slide screen. For the moderators, iit would greatly affect the effectiveness of the flow controlling ability. For example, if the current presenter displayed some inappropriate contents on the slide screen but the moderator was busy in handling some requests displaying on the dialog box of an active tab, the dialog box may covered the banned contents and the moderator was not noticed. In order to resolve this problem, we finally proposed another interface.

4.2. Final Design

We finally decided to use the split view framework design. This kind of interface is officially supported by iOS and only available on iPad. Based on the this user interface framework, we have extended the view hierarchy as shown the following chart:

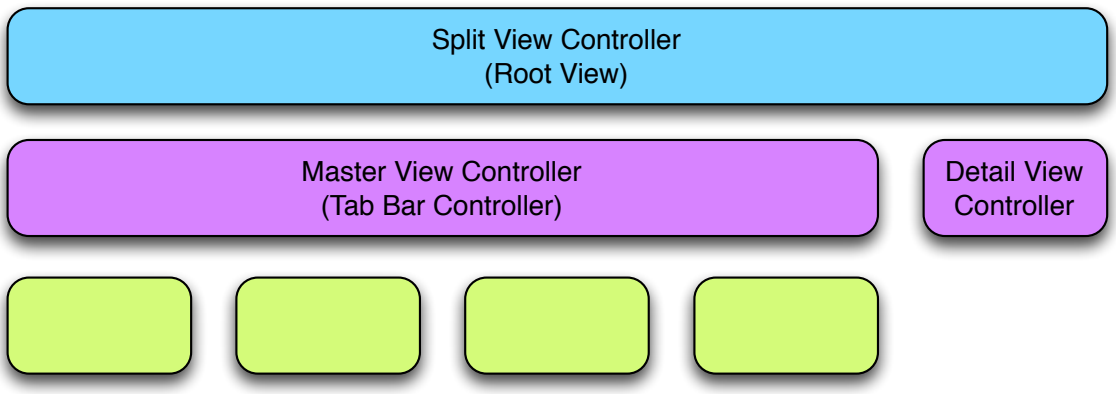


Figure 4.2.: Overall View Hierarchy of the application

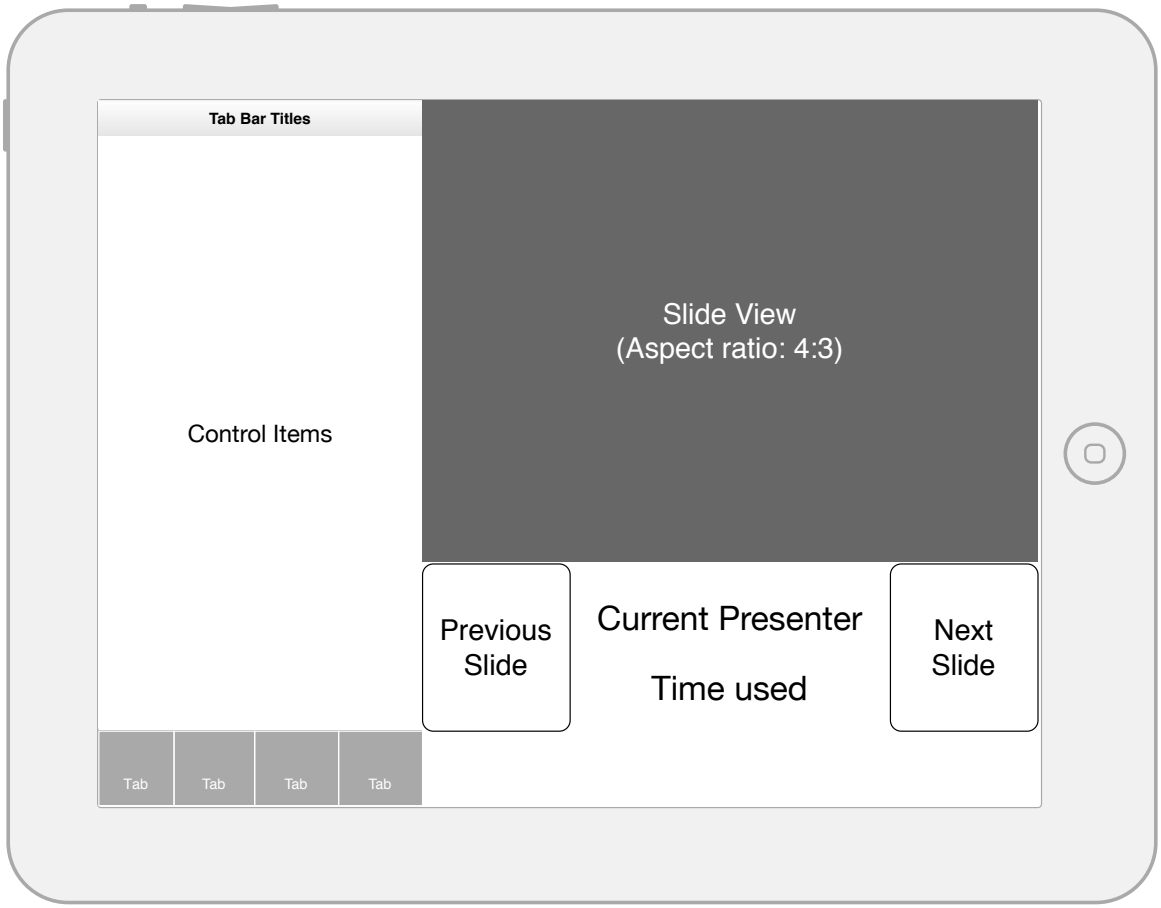


Figure 4.3.: Overview of the user interface of our application

The most remarkable layout of this kind of interface is that the monitor display is divided into two parts. In our design, master views are used to show the controlling options and show the system status. On the other hand, the detail view shows the slide contents and the items related to the presentation flow, such as the current presenter and timer.

4.3. Interface Update

After the development and testing of effectiveness of the user interface, the split view layout is believed to work effectively on our application. Based on this assumption, the

overall layout will not change much, only some items need to be rearranged or changed.

Regarding the previous slide view in the detail view part, which was originally an image view, is replaced by the PDF view controller provided by the PSPDFKit library. The overall layout does not changed much, but all the transitional animations and segue provided by the library make the user experience better. In addition, the button for switching slides on the presenter side can also be removed, as changing pages can be done by simply swipe right or left on the PDF view controller.

Secondly, drawing master view originally on the presenter side of application can be removed. Since the current implementation of the drawing features have been already integrated with the PSPDFKit Library, all drawings will be implemented by the PDF annotations. The original drawing interfaces will be replaced by the annotation editing tools inside the PDF view controller.

In contrast, new controlling items are needed for the video playback feature. On the presenter side, a new master view controller is added to the left hand side of the screen. This view controller display all the local video which have been loaded in by the user through iTunes using the Document folder, and the video already uploaded to the moderator. In addition, all the playback controlling items will be available.

For the moderator side, some previous controlling interface items will be reorganized. Screen Tab and Control Tab are grouped up to be one single tab named Status, and there will be a new tab for handling the video playback function. This view displays all video received from the moderators, with the name of the video file and which presenters they belong to. Also, playback controlling items will be shown in this view.

Chapter 5

Implementation Detail

5.1. PDF

5.1.1. View and Annotation

The implementation of PDF view and annotation is mainly based on the `PSPDFController`. The `PSPDFController` handled all elements, animation and detail for us. It should be mentioned that the `PSPDFController` have to be placed inside a `UINavigationController`, and the annotation bar could only be visible when it is placed to a appropriate controller.

There is no much to discuss about PDF viewing and annotation itself. Instead, the technique used for synchronization is more valuable to discuss. For those topics, refer to section 5.3.

5.1.2. Manipulating PDF

The manipulation of PDF file is done in the moderator device with the `PSPDFProcessor` class. Once moderator received presenter's PDF document, it store in a temporary

space, and import it into `PSPDFController` for showing on screen. By implementing the delegate methods, the `SlidesController` could keep monitoring changing page of the PDF document. With these detail, we could reconstruct a PDF file containing all slides.

To extract particular pages from a PDF document, we use `generatePDFFromDocument:pageRange:outputFileURL:options:progressBlock:error:` from `PSPDFProcessor` , and create a list of PDF file. After the presentation is finished, it is combined together and available to retrieve via iTunes' file sharing.

5.2. Video Playback

5.2.1. Network Communication and Data Transmissions

Extension of network protocol and handling

Sending and Receiving Video

Presenters are allowed to send videos before the moment that is actually needed. Since sending a video file takes significant amount of time even when the network condition is good, if the video transmission process is still rely on the user triggering event originated from the Cocoa user interface framework (i.e.: Performing the whole sending process on the main thread), the application will appeared to be frozen after the touch event triggered, which comes out a bad user experience.

In order to resolve this problem for providing better user experience and enhance the stability of the system during the transmission of the video files, a new strategy is developed. Figure 5.1 shows the process sequence concerning the video transmission

process.

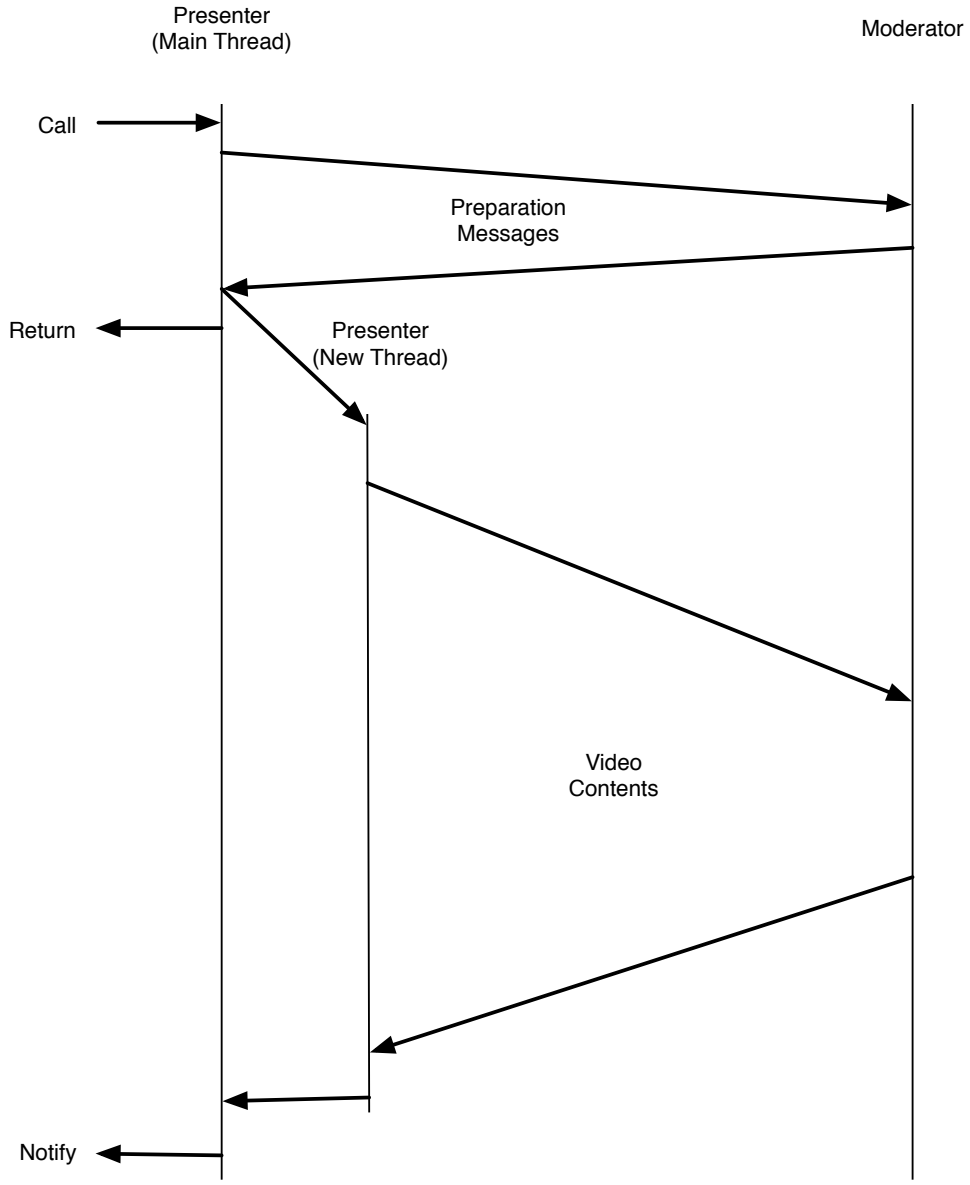


Figure 5.1.: Calling sequence of video transmission process

Firstly, the presenter will first send a video sending request to the moderator. Just like other controlling protocols, this will be done on the main thread. Once the moderator device received this request, it will immediately start a temporary TCP server socket and start accepting new connection. After that, the moderator respond the respective

presenter with the port number of the newly setup server socket.

After the temporary port number is received by the respective presenter, the presenter side application will create a new thread to establish the connection with the moderator and send the video content through the network. When the transmission is finished, the video file will be stored in the local storage of moderator. The moderator will once again notify the respective present that the transmission is completed. Then the temporary connection will be closed and the connection thread on the presenter side will also be terminated.

Each presenter will have a list of uploaded video, which is updated after each video transmission. Then the presenter can choose to play the uploaded video anytime.

Playback Control

Given that a list of uploaded video stored at the moderator side, presenters can send request to play and control the video playback processes though the network.

Play When the presenter send the play control request to the moderator, it embed with the video file name, which ensure the moderator to play the same file as the presenter want.

Pause, Stop and Seek Similar to the playing request, except seeking will also send out the seeking position in seconds, to the moderator.

Moderator Control

As moderator have the power to interrupt any ongoing presentation, the moderator have the right to stop any control from the presenters. Moderator also have the right of

controlling the progress of the video, and the right to blank the screen if the presentation is judged to be inappropriate.

`NSThread` and `pthread`

Regrading the thread programming of iOS applications, Objective-C language support both Cocoa Thread Library and the traditional-styled POSIX-C Thread Library. Although the official iOS development documents stated that both thread libraries are fully supported by iOS and they can be mixed up to use, we decided to use the Cocoa Thread Library calls when implementing the video transmission sequence.

It is because Cocoa Thread Library consists of a series of thread-safe library call. Also, enjoying the benefits from the well documented library calls and class references of the Objective-C iOS libraries, there are documentations labelled which classes within the Cocoa framework are thread-safe or thread-unsafe. Moreover, the function calls from the `NSThread` Class and for its instances are useful and easy to use. For example, the main thread instance can be found at anytime through the static class function call `[NSThread mainThread]` . Another useful function call is to send direct messages between threads using `performSelector:onThread:withObject:waitUntilDone:` .

These simplify much of the codes when comparing to implement the application using POSIX functions. As we need to declare a lot more global variables for inter-threading communications, or using global scope pthread pointers to retain the main thread.

5.2.2. Playback

The `MPMoviePlayerController` Class is provided by the Media Player framework natively available in the iOS Objective-C Library. We have chosen this class to be our video player as it natively support all video or audio formats and codecs provided by iOS.

Typical movie files with extension `.mov`, `.mp4`, `.mpv`, `.3gp` with the video contents compressed with MPEG-4 or H.264 Baseline Profile can be played by this movie player. Moreover, this movie player also support streamed video content from server provide HLS Standard video streaming services. Thus this class make the whole application flexible and extensible as the whole video playback component do not need to be reimplemented when providing the video streaming feature.

In our current implementation, we use this class to play local videos. This class have a lot of useful properties which can be accessed to obtain useful information, including the durations, resolution and media type of the movie. Also, the style of the controlling items of the movie player can also be modified to the correct behaviour of displaying the video on the screen.

5.3. External Display

When implementing the external display, we have two concerns: detection of external display and drawing on external display.

5.3.1. Detection

To detect the external display, we, again, use the `NSNotificationCenter` for detecting broadcast. Registering `handleScreenDidConnectNotification` and `handleScreenDidDisconnectNotification` , we can handle the external screen immediately when it is connected or disconnected.

After the connection of external monitor, we have to manage the content of external monitor. The setup of external monitor involve two classes, `UIScreen` and `UIWindow` . `UIScreen` contain settings of the connected external monitor including resolution and brightness. In addition, `UIScreen` contains available resolution setting of connected monitor. Based on the resolution available, of which preferred resolution is used in prototype, we create an instance of `UIWindow` . Adding root view to the created `UIWindow` object, we are able to add subview on it, showing custom content to external screen.

5.3.2. Synchronization

The next step is to make content to show in external screen, synchronizing with device view. To manage those actions, we have create a new view controller, handling operations regarding external screen called `VPSlidesViewController` . Moreover, we are trying to hiding that there are two view hierarchies in the controller. Thus classes using this controller may only consider it as normal view controller.

Exiting Views - Cloning

Copying a view, we found a view may not be able to make deep copy and sometime may need custom copying in order to have a deep copy of view. To handle different class

in `VPSlidesViewController` for copying, we have use **introspection** for determining class of an object, including `isKindOfClass:` and `isMemberOfClass:`, then handle each class differently.

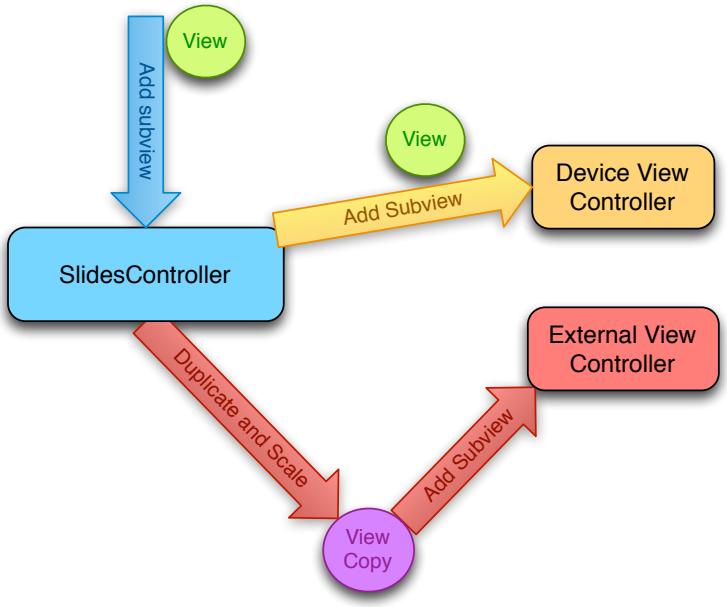


Figure 5.2.: Cloning an object

Another issue is to duplicate a view controller instead of view. The solution is native: we duplicate the view controller with similar technique. However, the scaling and frame adjustment is done based on the `view` property, as well as the `addSubview:` method.

Handling Changes - Duplicating Message

In order to duplicate a message, we first need to make a message, or a piece of code, become easy to handle in program. A good solution is to use block as parameter, abstracting programming logic.

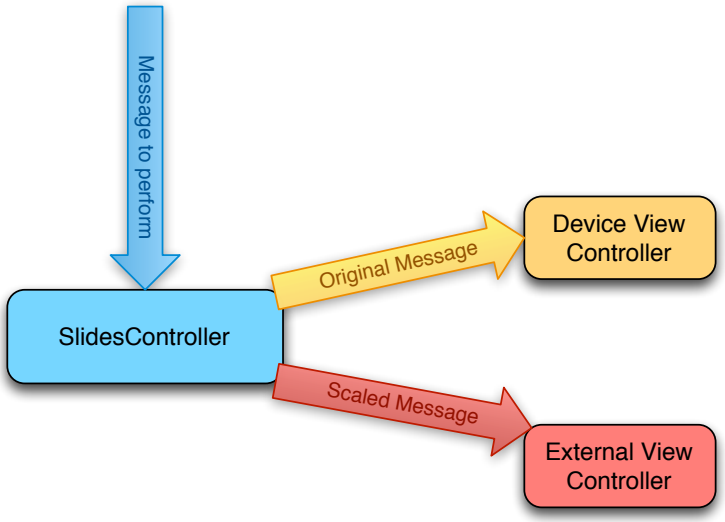


Figure 5.3.: Duplicating a message

The block basically having one parameter, the view, and the block content assume valid view is passed into it and able to perform operation on the block. Following is implementation of `perform:onTag:` without additional parameter:

```
1 - (void) perform:(void (^) (UIView *))action onTag:(NS
        Integer) tag {
2    action([self.deviceView viewWithTag: tag]);
3    action([self.externalView viewWithTag: tag]);
4 }
```

However, block without parameter could not handle program logic which include detail of position within the view, which would need to be scaled during synchronization. Therefore, another set of `perform:onTag:withPoint:` is implemented, by sending block with scaled point as parameter.

```
1 - (void) perform:(void (^) (UIView *, CGPoint)) action
        onTag:(NSInteger) tag withPoint:(CGPoint) point {
2    action([self.deviceView viewWithTag: tag], point);
3    action([self.externalView viewWithTag: tag],
           CGPointMakeScale(point, 1, self.
```

```
            externalViewZoomRatio));
4   }
```

Handling Changes - Delegate Response

We use delegate for synchronization of third party framework, i.e. the PSPDFKit. Delegate is a key concept in iOS and Objective-C programming for communicating between objects, which idea is similar to listener in Java or a callback function in other language. With delegate, an object could ask another object, namely delegate object, to perform some action. Syntactically, the delegate object have to compromise a protocol, while the delegator has a reference pointing to the delegate object. Delegator do not concern about the actual class of the delegate object, as long as it compromise the protocol of delegate.

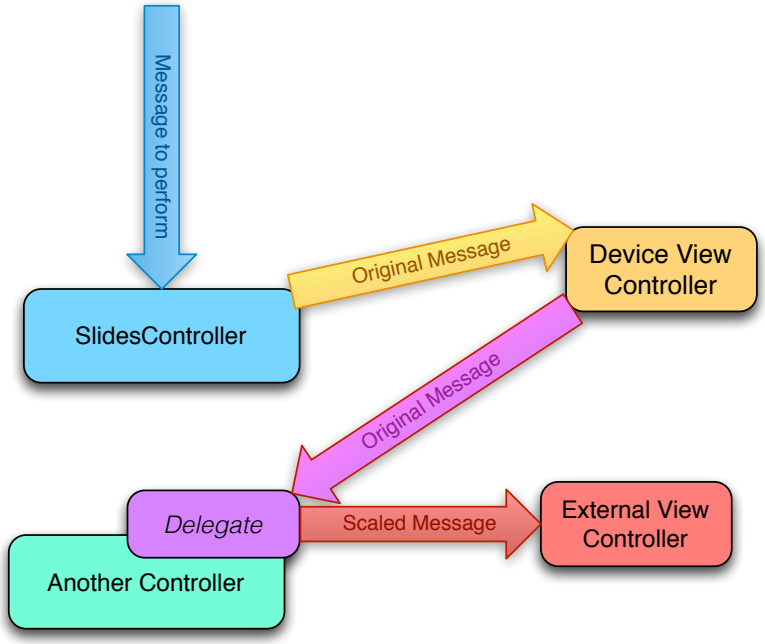


Figure 5.4.: Message duplicating with delegate

As the PSPDFKit's `PSPDFViewController` has abstracted and handled most actions including go to another page, or editing annotation for us. In simple word, the

view controller has catch the message and handle it for us, and we could not intercept the message. Therefore, we could not use the method of duplicating message at this case. Fortunately, `PSPDFViewController` has a property of delegate, with the protocol `PSPDFViewControllerDelegate` , and we can do synchronization with aid of delegate object.

The protocol includes method such as `pdfViewController:didLoadPageView:` , `pdfViewController:didShowPageView:` . Once we implement those methods with the slide controller, we could get notify when the PDF view controller in the device display changes its displaying page, and we could then send message to view controller in the external display to perform similar action.

In the implementation, instead of implementing a separated class for delegate, we use the `SlidesViewController` to implement the `PSPDFViewControllerDelegate` protocol and handle the delegate method. The `SlidesViewController` duplicate the message to external display's view controller, with proper resizing and mapping.

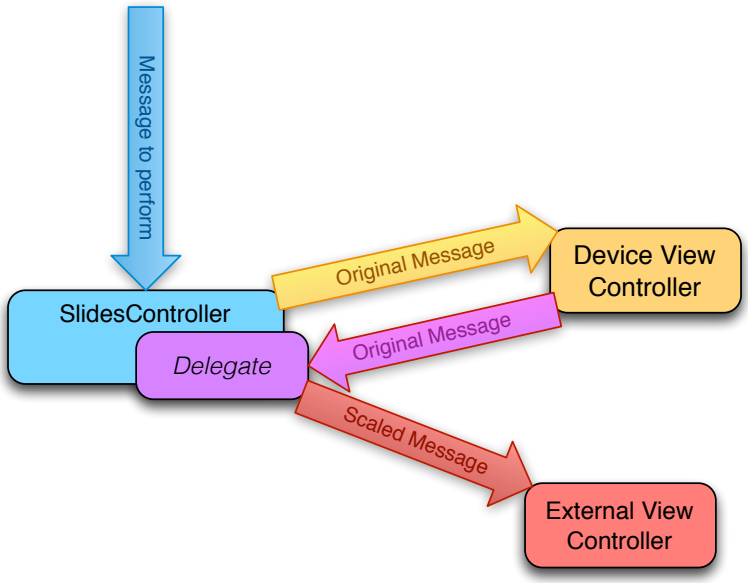


Figure 5.5.: Message duplicating with delegate of same object

Adjustment for External Display

Despite we need to synchronize device view and external display, some of those do not need, or should not need to be synchronized. For example, the `PSPDFController` should shows UI elements in the device view, allowing user to jump to a page or start editing an annotation. Those UI elements are not necessary for external display.

5.4. Network Connection

As Objective-C is superset of C, and iOS has implemented interface on Berkeley sockets, as known as BSD sockets, we can use the recompile codes written in C and developed based on Linux. However, we tried to avoid some low level referencing and try using Core Framework calls and `NSFileHandle` .

Core Framework of iOS provides family of `CFSocket*` methods for some socket operations. We have used including `CFSocketCreate()` , `CFSocketSetAddress()` , `CFSocketInvalidate()` and `CFRelease()` for replacing system calls such as `socket()` , `connect()` , `listen()` etc. In addition, we have wrap the socket file descriptor to `NSFileHandle` , providing more utilities for operations. Therefore we could avoid storing variables in low level, making use of object oriented paradigm and API provided in Cocoa foundation. Wrapping the file descriptor to `NSFileHandle` , we can use more API of Coca foundation and avoid complicated codes. For instance, Linux system call `read()` is a blocking call and require thread programming and inter-thread communication for preventing race condition. With `NSFileHandle` , we could make use of `NSNotificationCenter` with broadcasting for notifying data availability without blocking or implementing threads, and handling received data easily.

We have write a class of `TCPServer` and `TCPConnection`. Both class only making method such as `connectTo:`, `listenTo:` public to call. Classes use `TCPServer` or `TCPConnection` do not need to consider the internal implementation and simply register callback when receiving data.

5.5. File System

Comparing to other mobile device OS, iOS adopts the "Sandbox" file storage approaches for each application, which is more secure because the file imported for one application will not affect other application installed on the same iOS machine. In contrast, this file storage approaches limits the flexibility of file handling.

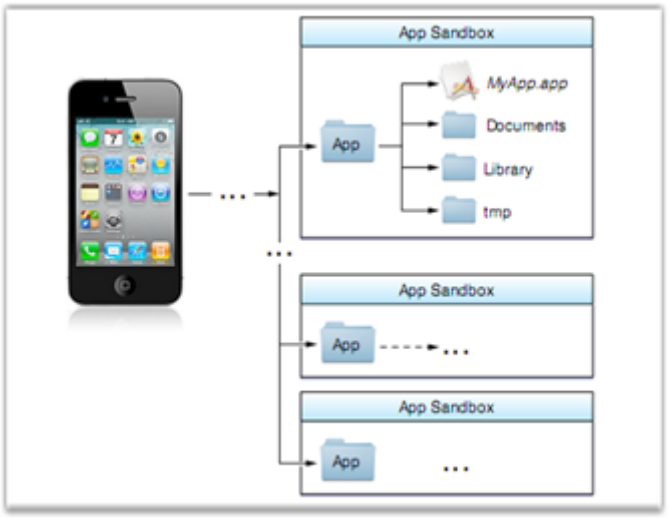


Figure 5.6.: An illustration of the iOS Sandbox storage approaches

After some modification of the applications, it allows users to add files into the document folder through iTunes. But neither the users nor the developers can add, create or access any sub-folders in this file sharing folder; neither through iTunes, nor through program coding.

But iOS file system provide a workaround for developers to store the data related to their applications. This special folder is named Library in iOS 5.1.1, which is not visible to the users, since they will not be able to see the contents of these folders through iTunes. But it can be fully controlled by the application developers through program codes. This folder comes into handy when handling the incoming PDF files and video files of the moderator side applications, since there is a cache folder inside this Library folder and developers have full access to this folder.

We make use of the file sharing ability provided by iOS and iTunes, to allow the presenters to import their PDF files and video files. At the moment of transferring files from the presenters to the moderator, incoming files can be stored into the cache folder of the moderator.

5.6. User Interface

5.6.1. Split View

Figure 5.7.: Split View Representation display in Xcode interface builder

Split view controller is one of the famous user interface design framework, widely using by the application on iPad. This framework itself have already support and pre-implemented lots of animations and segue actions concerning the two main subviews of it, namely Master View and Detail View, from left to right respectively.

Similarly, subviews will also holds a property to the reference of the nearest split view controller ancestor in the view hierarchy. That means the subview controllers can easily access the property of the split view controller. This is essential for different view controllers to communicate and passing values and variable as we often use the controlling items in the master view to control the interface items possessed by another view controller.

5.6.2. Tab Bar

Figure 5.8.: Tab Bar View display in Xcode interface builder

Tab bar is used to divide the same view space into different pages. It can be achieved by linking the respective tab bar items with respective view controllers. Similar to the split view controllers, the tab bar controller can contains its subviews in an `NSArray`, with object index from 0 to $(n-1)$ when there is n tabs defined.

Again, the subview controllers will also hold a property of object reference of their nearest tab bar controller ancestor. This useful is to change the title view of the top navigation bar when switching between tabs. Also, it is common that there are

different layouts of navigation bar view, and it can be changed easily by directly change the navigation bar behaviors.

5.6.3. Table View and Cells

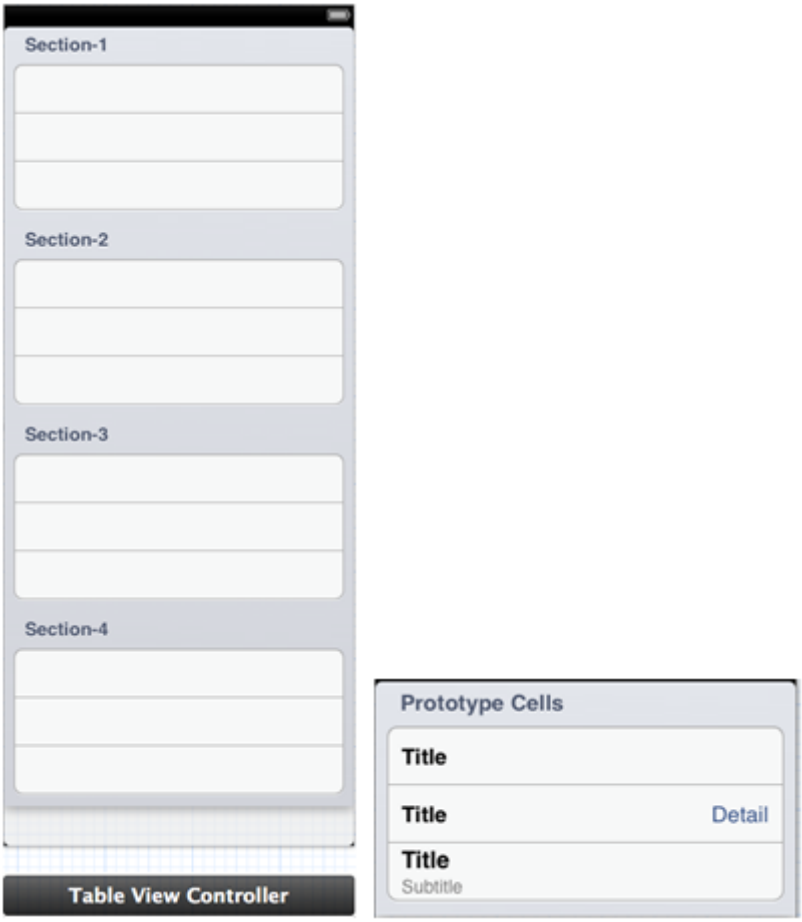


Figure 5.9.: Table View and Prototype Cells display in Xcode interface builder

In our implementation, the table view controllers under each tab bar item are stored in the tab bar controller as mentioned above. The table needed to be configured into dynamic type table view, which that all the contents, number of rows and sections are defined by program code. Although the program code can define all the cell layout properties such as fonts and colors, the interface builder also support to build prototype

cells to configure the cell layout.

Every prototype cell have to assign a unique “reuse identifier” for the table view controller to maintain a “pool” of reusable cells. Cells that use the same identifier will be generated based on the same prototype cells.

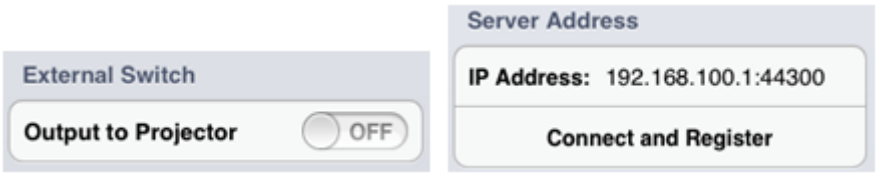


Figure 5.10.: A controlling item added as a subview of a grouped cell.

Also, controlling items can be separately created and added into table cells .This kind of cell is common on the set up screens and scenarios. This modification is not possible to be defined on the interface builder. So the control items have to be defined and added to the cell by program code.

Chapter 6

Progress and Evaluation

6.1. Source Control

Source and revision control is an important concern for any project, especially for project composed of more than one people. In last semester, we have been using Dropbox as source control. Dropbox provides features such as synchronization of files, and stored old revision of files. However, the synchronization of Dropbox is not fast enough when both of us are developing in the same time. In addition, Dropbox do not resolve any conflict but only creating a backup file. Dropbox is not suitable for source control for an active project. Therefore, we explore for another source control solution.

We decided to use git as source control. Git is a popular and common source control software, supporting multiple branch and non-linear development, as well as version control. Moreover, Xcode, the integrated development environment for iOS application, also support git.

The next issue is to find a host for repository of our project. There are free web service available for hosting a repository of git, such as github and bitbucket. However, we found some network problem when connecting those web services with department

machines. Therefore, we built our own server to host the repository. The server is a ubuntu linux while the package is called gitolite. gitolite is using SSH key for managing access control to repository, and distinguish user.

Once the server is stable, we move all existing source code into the server and create repository for our project. Afterward, we stick closely to the git server by creating branches for new features and testing. The git server allow us develop simultaneously without disturbing progress of each other. Branches created could be merge with git, and only need few manual operation during merging.

6.2. Schedule

Jan 2013	Code clean up and refinement
	Set up of git server
Late Jan - Mid Feb 2013	Research of Third Party Library
Mid Feb - Early Mar 2013	Investigate iOS video plackback
	Testing on PSPDFKit with external display
Mar 2013	Testing of remote video playback
	Testing of PSPDFKit with remote moderator
Late Mar - Early Apr 2013	Integration of libraries towards main application
Apr 2013	Testing and Debugging
	Documentation

Table 6.1.: Progress Report of Spring 2013

Chapter 7

Contribution and Reflection

7.1. Contributaion

7.1.1. Summer 2012

The project kickoff in Summer 2012, and I have been working on some preliminary preparation of project. I first work on marketing research and testing on existing presentation softwares, including Opencast Matterhorn, TechSmith Camtasia Relay. This give me an whole picture of functionalities that user may want. With these experiences, I start drafting a specification of our project.

Meanwhile, I am also a newbie of iOS programming. Therefore, I started studying iOS programming based on a course of Standford University, which videos and notes are available in iTunes U, together with Apple's developer site and manual. Writing some simple applications, I start get the feeling of iOS programming.

7.1.2. Fall 2012

Starting of regular semester, I have been continuing the draft of specification together with Jack. We have been thinking for a new direction of presentation together, and finalized the idea of collaborative presentation until October.

On the other hand, I have written some simple applications on testing some functionalities of iPad and APIs. Aspects including external display and synchronization, network connection as server-client model on iOS, as well as a drawing pad on top of images. During developing those testing applications, I write with core classes which could be reused later, and a simple UI on top of the core classes. With core classes, I could combine those apps and with Jack's work in short time.

In short, I focus on the **model** part of Model-View-Controller, as well as the core of synchronization with external display.

7.1.3. Spring 2013

At the beginning of Spring 2013, we have evaluated the work done in Fall 2012, and consider refinement of collaborative presentation concept. Meanwhile, I start discovering OpenCV on iOS, as well as setup the git server for hosting the repository of vPresent. Despite OpenCV could not contribute in this project, it is still a good experience on iOS programming.

After the git server settle down, I focus on PDF parsing and viewing. I have tried to parse a PDF with iOS official APIs, as well as exploring some third party APIs. Comparing iOS APIs with few third part framework and considering integration with collaborative presentation, I chose the PSPDFKit as the third party framework for

parsing PDF, and start working on integration with existing applications. Integration consist of two main parts, handling synchronization with external display and network communication for data transfer and control signal.

7.2. Reflection

From this project, I have learned a lot of iOS programming. I started us a newbie, and now could finish a prototype of an iOS apps with my partner. Despite I have not work on the user interface, I have get used to the core of iOS programming, including MVC model, delegate and protocol, as well as block as encapsulation of programming logic.

Besides the technical knowledge, I have also learnt how to come up with an idea, and tried to make the idea becomes real product by implementing a prototype. This is very common in the early stage of developing a product. This project would be a precious and valuable experience for my future.

Chapter 8

Conclusion

In the last semester, we have defined collaborative presentation, a new concept of presentation style. We have implemented two prototypes for demonstrating collaborative presentation, for moderator and presenter respectively. The prototype can demonstrated a subset of functions stated in our design, we are able to demonstrate the idea of collaborative presentation, as well as proof the feasibility of implementing collaborative presentation application on iOS.

In this semester, we have done further researches and studies on some third party library for supporting at one type of common presentation content files and trying to expend collaborative presentation to support rich media elements. And we finally integrated the PSPDFKit Library into our application to support PDF files, and using native Apple Development Library to support the playback of videos.

We continued to sharpen our skills of developing iOS application and the development environment. And we are now able to integrate 3rd party library into our application and continue to expend our network application protocols. We are also able to use NSThread APIs to create multithreading application on iOS to improve the performance and. Our learning ability has been improved, as well as the problem-solving

skills.

Finally, we tried to extend our applications from the progress of last semester. As application development involve many aspect including software design, user interface and experience designing, implementation of application, testing, debugging and also profiling of application, aiming at optimization, we have once again experience of whole software engineering cycle. This is a valuable experience for us.

Chapter 9

Acknowledgement

Firstly, we would like to thank our supervisor Prof. Michael R. Lyu. He provided us valuable comments and guidelines throughout the whole project.

Secondly, we would like to thank the researchers from VIEWLab, Mr. Edward Yau Hon Hei and Mr. Un Tze Lung provided great amount of hardware supports and remarkably ideas to make our project become more fruitful and interesting.

Last but not least, we would like to thank our lab technicians to provide the accessibility of using the developing tools of Mac OS machines in our department computer laboratory, as well as providing disk quota and virtual machines for server hosting.

Chapter 10

Reference

[1] Apple Inc. iOS Developer Library [Online]. Available: https://developer.apple.com/library/ios/navigation/index.html

[2] Object Management Group. UML Specification [Online]. Available: http://www.omg.org/spec/UML/

[3] Stack Overflow [Online]. Available: http://stackoverflow.com/

[4] Wikipedia [Online]. Available: http://www.wikipedia.org/

[5] PSPDFKit Reference [Online]. Available: http://pspdfkit.com/documentation/

Appendix A

Network Message Specification

A.1. Register

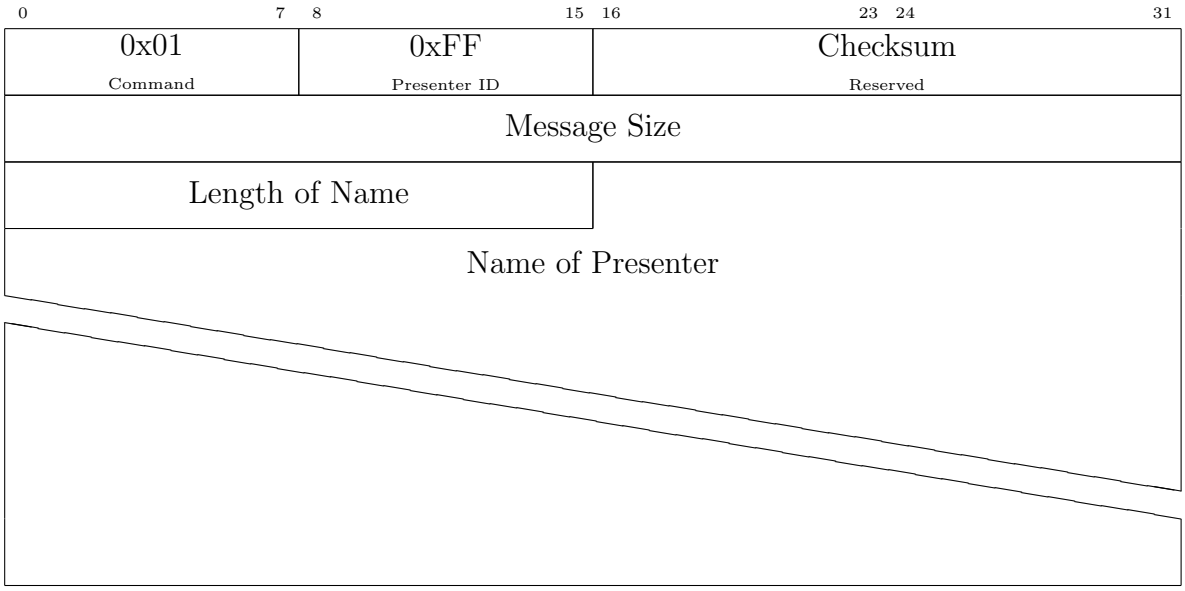


Figure A.1.: Register Request

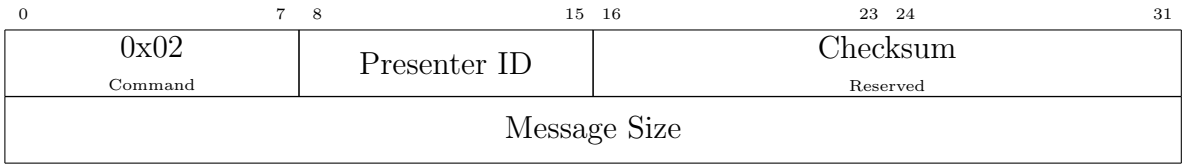


Figure A.2.: Register Success Response

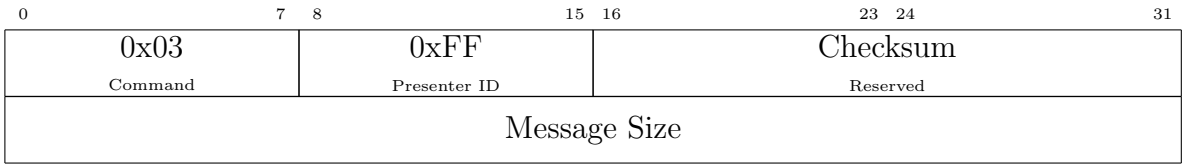


Figure A.3.: Register Failure Response

A.2. Unregister

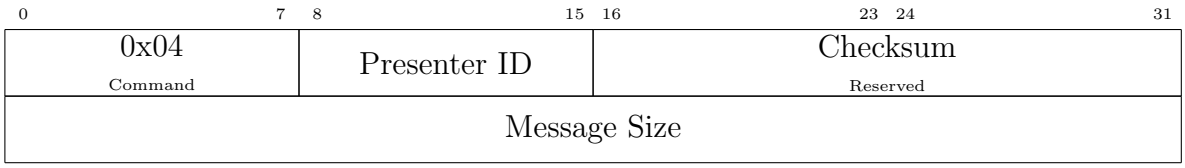


Figure A.4.: Unregister Request

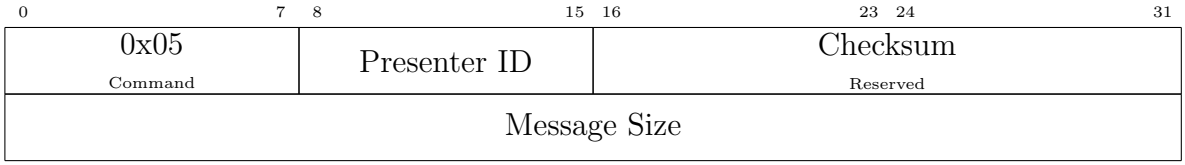


Figure A.5.: Unregister Response

A.3. Control Permission

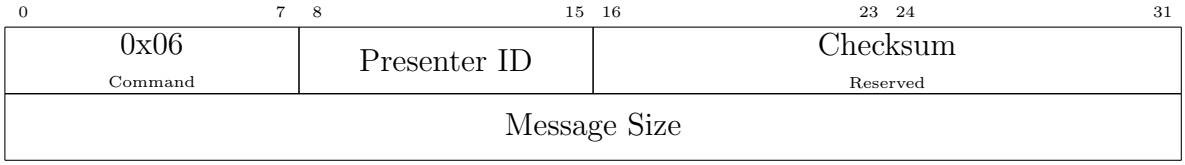


Figure A.6.: Control Request

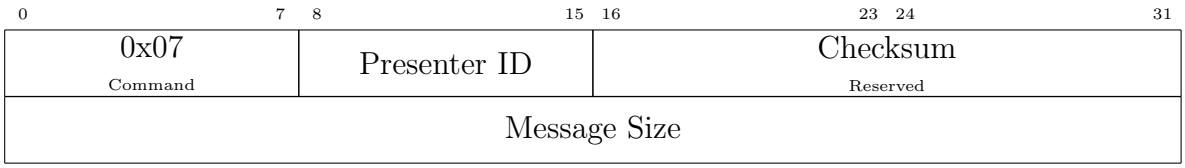


Figure A.7.: Control Response

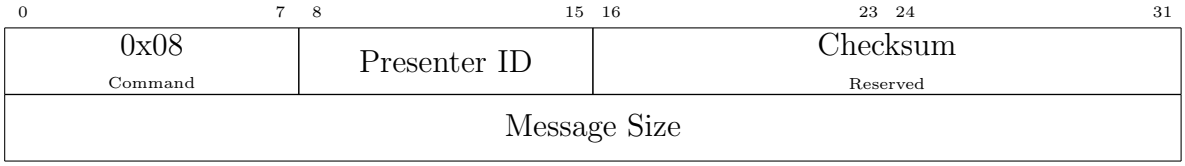


Figure A.8.: Grant Control

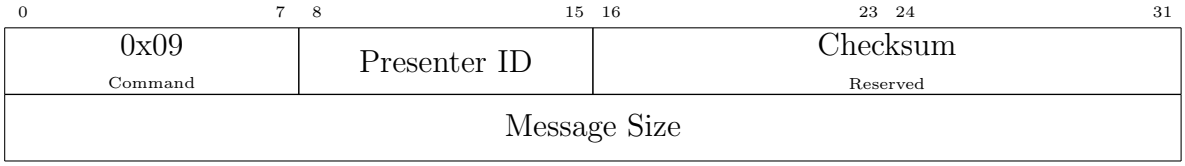


Figure A.9.: Withdraw Control

A.4. Control Signal

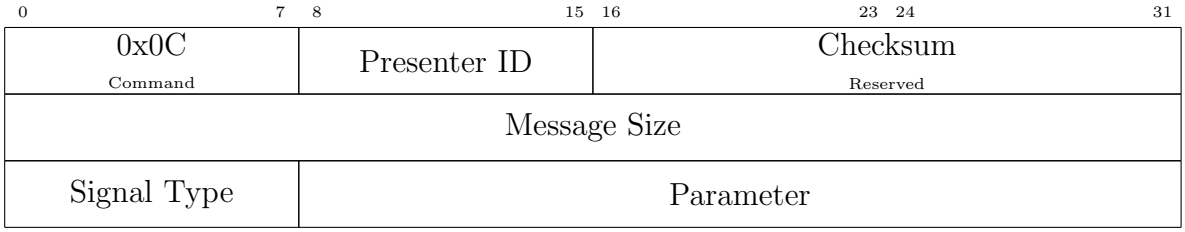


Figure A.10.: Control Signal Request

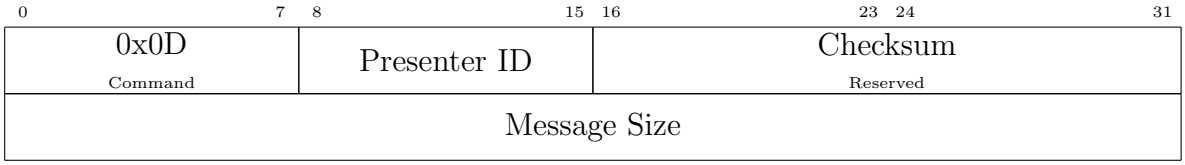


Figure A.11.: Control Signal Success Response

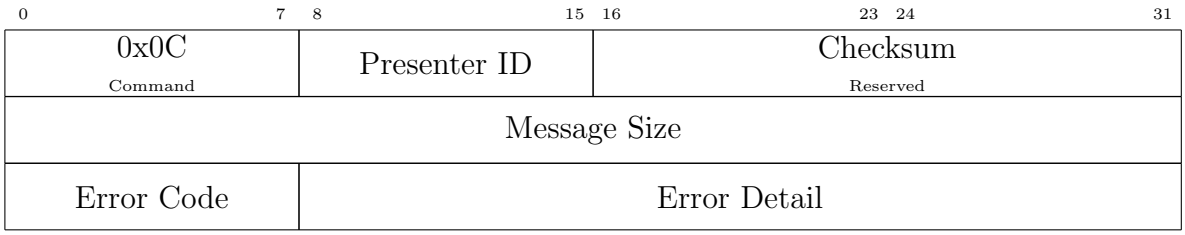


Figure A.12.: Control Signal Request

A.5. Video Playback

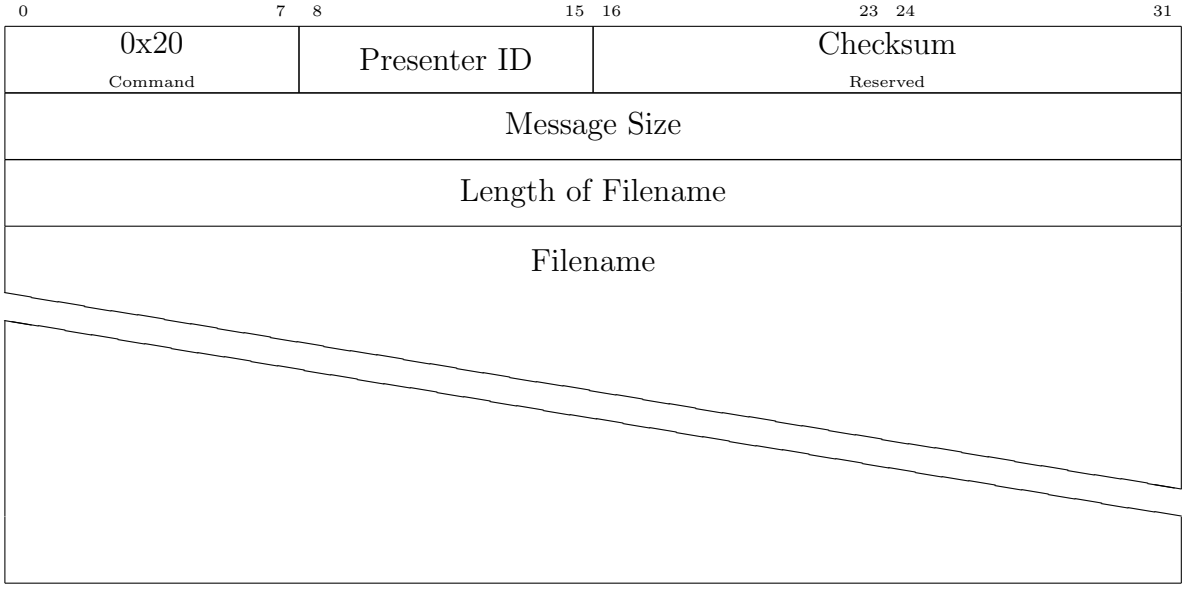


Figure A.13.: Video Play Request

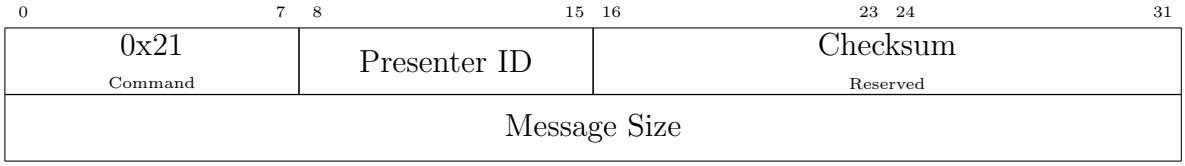


Figure A.14.: Video Play Success Response

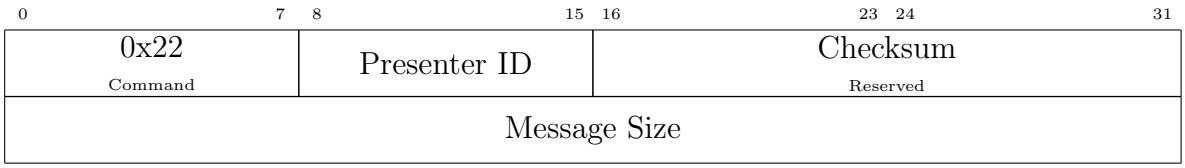


Figure A.15.: Video Pause Request

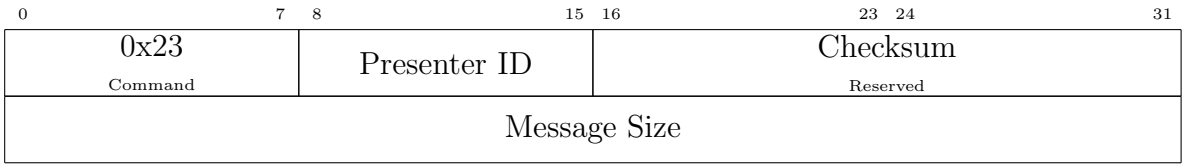


Figure A.16.: Video Pause Success Response

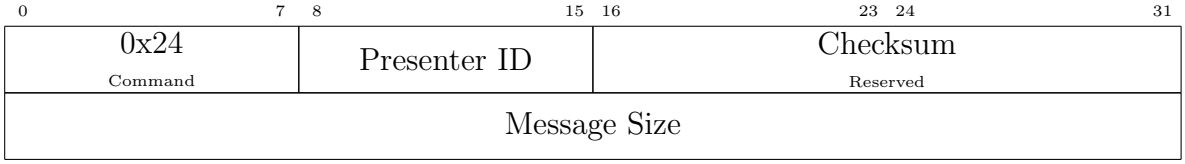


Figure A.17.: Video Stop Request

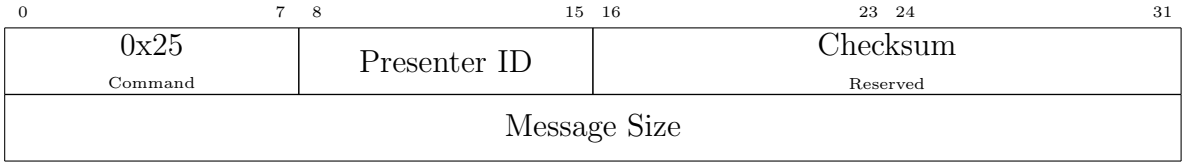


Figure A.18.: Video Stop Success Response

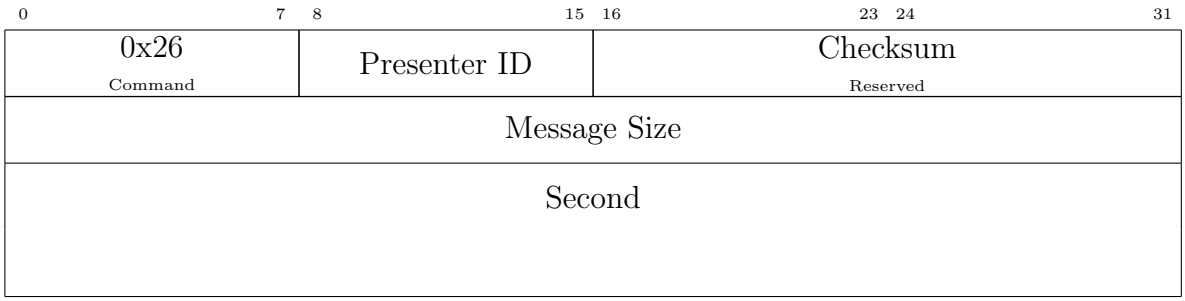


Figure A.19.: Video Goto Request

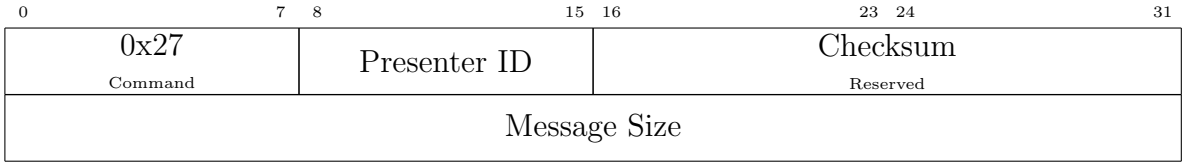


Figure A.20.: Video Goto Success Response

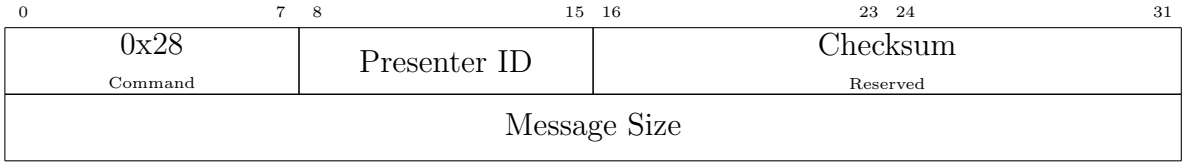


Figure A.21.: Video Prepare Send Request

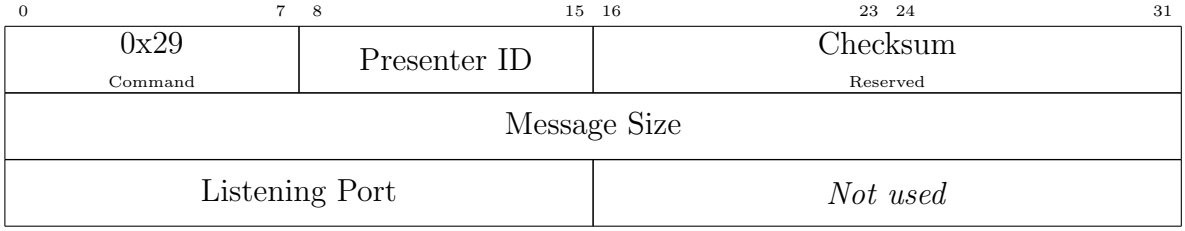


Figure A.22.: Video Prepare Send Response

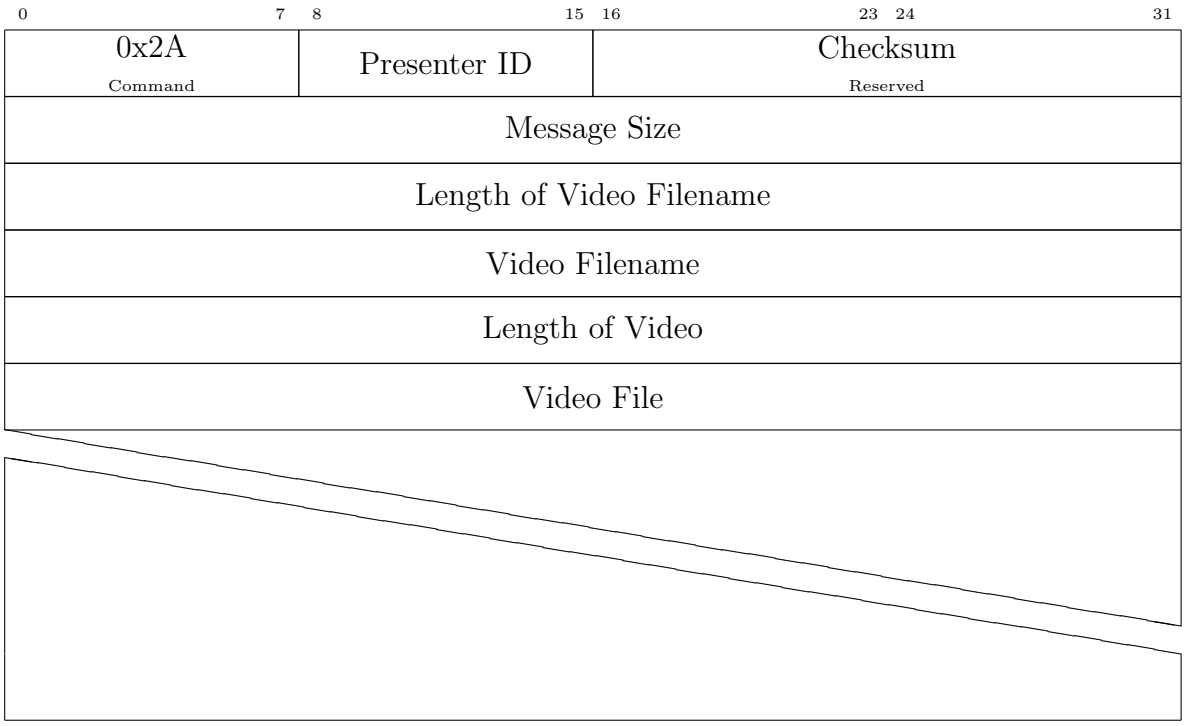


Figure A.23.: Video Send

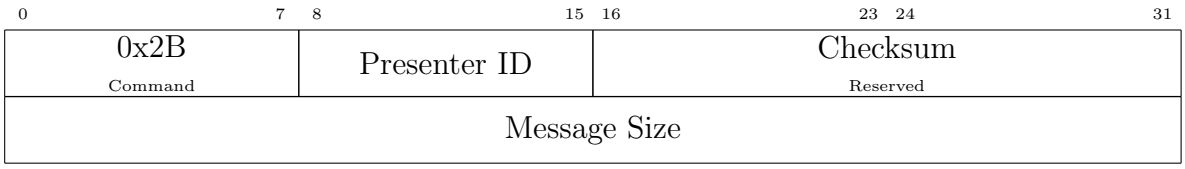


Figure A.24.: Video Send Complete Respond

A.6. PDF Data

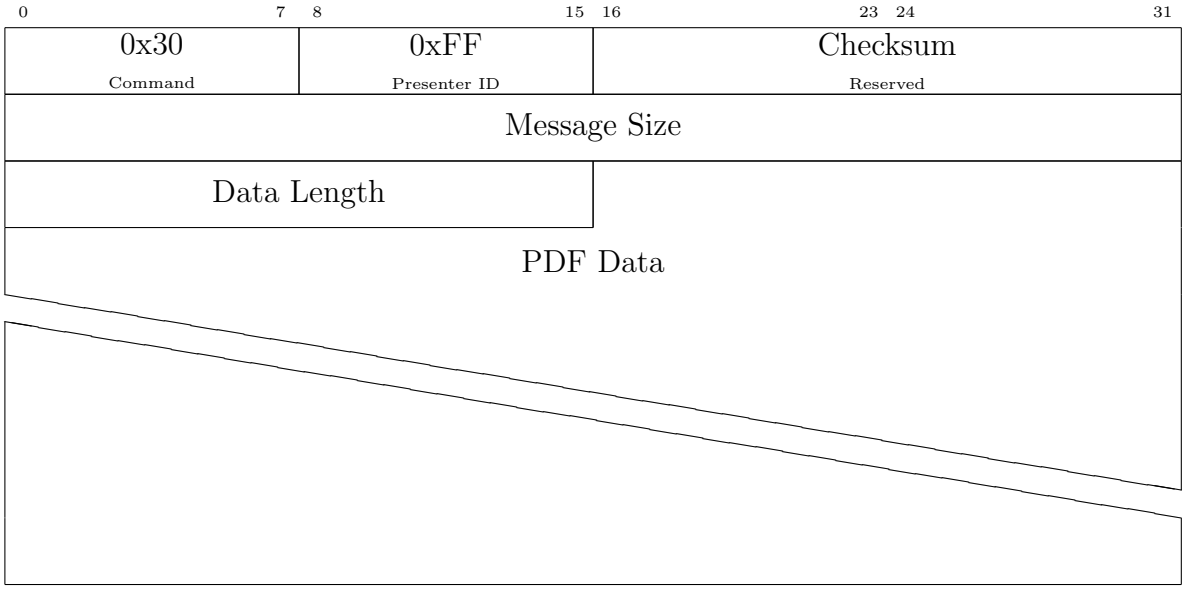


Figure A.25.: PDF Data

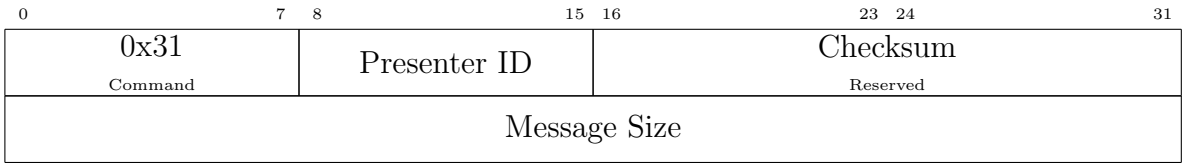


Figure A.26.: PDF Data Acknowledge

A.7. Annotation

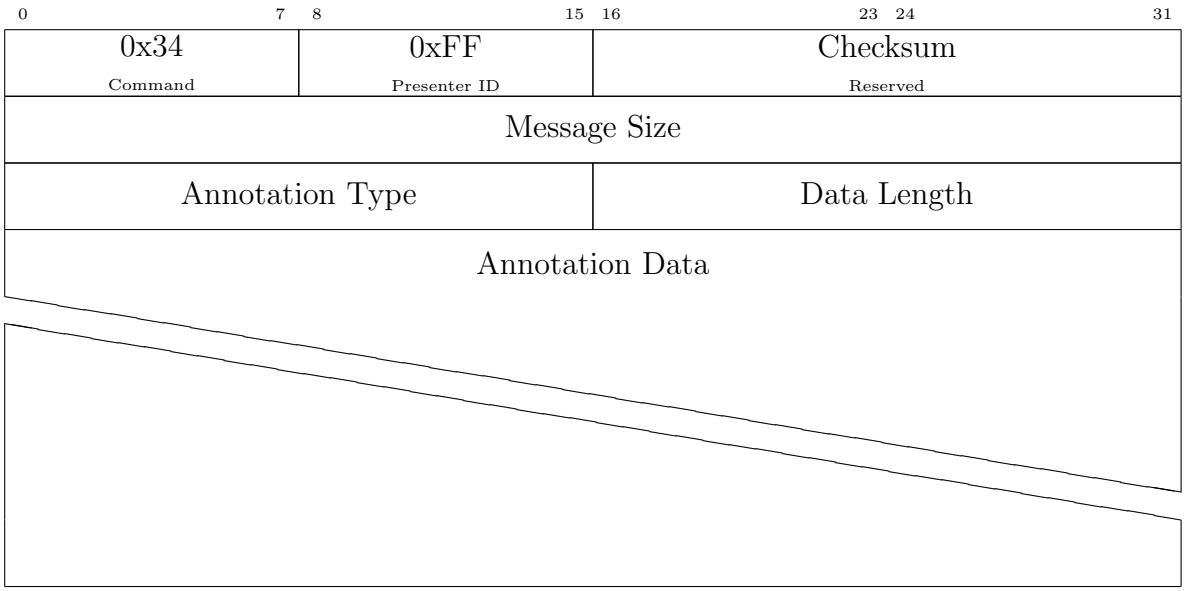


Figure A.27.: Register Request

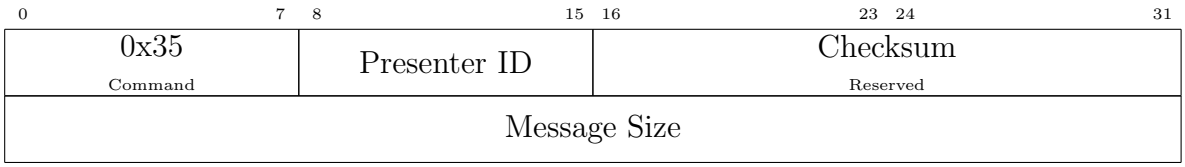


Figure A.28.: PDF Data Acknowledge