

© Copyright 2023

Lalin Wachirawutthichai

The Wide-Field Ethnography Navigator:  
A browser-based multi-stream, multi-modal dataset viewer  
that supports the navigational work of Wide-Field Ethnography research

Lalin Wachirawutthichai

A master's project paper  
submitted in partial fulfillment of the  
requirements for the degree of

Master of Science

University of Washington

2023

Reading Committee:

David Socha, Chair

Clark Olson

David LeBlanc

Program Authorized to Offer Degree:  
Computer Science & Software Engineering

University of Washington

**Abstract**

The Wide-Field Ethnography Navigator:  
A browser-based multi-stream, multi-modal dataset viewer  
that supports the navigational work of Wide-Field Ethnography research

Lalin Wachirawutthichai

Chair of the Supervisory Committee:  
Associate Professor David Socha  
Computer Science & Software Engineering

Wide-Field Ethnography (WFE) refers to an approach of gathering and analyzing large datasets of videos, audio files, screen capture videos, photos, and other digital artifacts related to recording the intricate intermingling of human subjects with computer systems and the social relationship and collaborations among these entities. WFE datasets are high in volume, containing multiple types of data and multiple data sources capturing the same events or moments of interest from different perspectives. For instance, the BeamCoffer datasets has 6 terabytes of video and audio recordings of software developers at work, videos of their computer screens, and thousands of photographs. It covers 11 working days with 380 hours of video footage and 292 hours of screen capture recordings. The sheer volume of data gathered and its modal diversity make it hard to navigate the dataset to find the moments that are meaningful to the research question, especially if one wants to simultaneously play more than one video or audio file to concurrently see and hear different perspectives of the action unfolding at a particular moment of time. There are currently no tools that offer a reasonable way to navigate a WFE dataset. This project describes the WFE Navigator, a software system built to help researchers navigate through such datasets.

# Table of Contents

List of Figures .....	6
List of Tables .....	8
1 Introduction .....	9
2 Background .....	10
2.1 The dataset: BeamCoffer .....	10
2.2 WFE Ecosystems.....	11
2.3 Previous work in navigational tools .....	12
3 What we built .....	17
3.1 Timeline .....	19
3.2 Playback Controls.....	21
3.3 Stream View .....	21
3.4 Query Controller .....	22
4 Similar systems .....	24
5 System Design.....	26
5.1 Overview of Application Components.....	26
5.2 How Data Flows through Components .....	31
5.3 How Playback Works.....	32
5.4 Domain specific models .....	39
5.5 Custom Classes and Methods .....	41
5.6 Management of Audio Branches.....	42
5.7 User Interface Considerations.....	44
5.7.1 What the Stream Manager Shows .....	45
5.7.2 Usability .....	47
6 Evolution of Concepts .....	48
6.1 Definition of Stream .....	48
7 Conclusion.....	51
7.1 Constraints and Limitations .....	51
7.2 Future Work.....	52
8 Bibliography.....	54

# List of Figures

Figure 1. Sample of equipment setup: GoPro cameras (blue), regular video cameras (yellow), and Zoom audio recorders (green) (Socha, 2015). .....	10
Figure 2. Sample directory structure. ....	11
Figure 3. Previous prototype of calendar view (Ogo & Socha, 2019). ....	13
Figure 4. Previous prototype of day view (Ogo & Socha, 2019). ....	14
Figure 5. Concept sketch (A) of day view (Ogo & Socha, 2019). ....	16
Figure 6. Concept sketch (B) of day view (Ogo & Socha, 2019). ....	17
Figure 7. WFE Navigator showing 4 streams. ....	18
Figure 8. General mapping of screen real estate. ....	19
Figure 9. Timeline area elements annotated. ....	20
Figure 10. Additional Timeline area elements annotated. ....	21
Figure 11. Playback Controller area. ....	21
Figure 12. Query Controller area. ....	22
Figure 13. Add (B) or remove (C) streams from query. ....	23
Figure 14. Application Domain Model showing relationships to the React framework. ....	27
Figure 15. Mapping of screen real estate to components. ....	28
Figure 16. Timeline: Timeline Slider component. ....	29
Figure 17. Timeline: Stream Timelines component. ....	29
Figure 18. Timeline: Stream Controller component. ....	29
Figure 19. Playback Controls component. ....	30
Figure 20. Query Controller component. ....	30
Figure 21. Stream View component. ....	31
Figure 22. Data access by component. ....	32
Figure 23. Playback Sequence Diagram. ....	34
Figure 24. Playback Sequence: (1) Start Playback. ....	35
Figure 25. Player UI when media content is available. ....	36
Figure 26. Player UI when there is no Media at time t. ....	36
Figure 27. Player UI when Media is hidden by user. ....	36
Figure 28. Player UI when Media source is invalid. ....	36
Figure 29. Playback Sequence: (2) Ongoing Playback. ....	37
Figure 30. Playback Sequence: (3) Stop Playback. ....	38
Figure 31. Domain model of a WFE Stream. ....	39
Figure 33. Domain model of a WFE Media. ....	40
Figure 34. Properties and methods of Stream and Media objects. ....	42
Figure 35. Timeline snippet for audio reference. ....	44
Figure 36. How audio goes through gain and panner controls before outputting to the output speaker, matching the state of Figure 34. ....	44
Figure 37. Audio Controller in enabled state. ....	46
Figure 38. Audio Controller in disabled state. ....	46
Figure 39. Playback Controller in inactive state. ....	47
Figure 40. Playback Controller during ongoing playback. ....	47
Figure 41. Query Controller when there are matched queries. ....	48

Figure 42. Query Controller when there is no matched query..... 48  
Figure 43. Streams tagged with PS A and Video file type over 3 separate days..... 50

## List of Tables

Table 1. Comparison table for existing tools based on criteria. ....	25
Table 2. All possible Media Handler Interfaces. ....	45



# 1 Introduction

Ethnography is the study of people, and ethnographical research data is traditionally gathered using methods such as participant observation, interviews, and artefact analysis. Such data collecting techniques commonly consume a lot of time and produce somewhat subjective observations based on the interpretation of the observer involved.

Wide-Field Ethnography (WFE) is a subset of ethnography coined by Socha to refer to “an approach of gathering and collaboratively analyzing large, multi-modal, multi-stream datasets of [physical-cyber-social systems] in action” (Socha, Adams, et al., 2016). Wide-Field Ethnography embodies ‘wideness’ in various ways. WFE utilizes a *wide* variety of data collecting tools to gather data from a *wide* range of data fields (space, time, modalities) across a *wide* expanse for collaborative use by a *wide* set of disciplinary fields.

Datasets gathered for WFE studies are high in volume, multi-modal, and multi-stream. A multi-modal dataset is a dataset that contains multiple types of data, such as a dataset with videos, voice recordings, photos, and health data. A multi-stream dataset is a dataset that contains temporally parallel data, such as day-long video recordings of a building entrance captured from 3 angles, producing 3 separate recordings output. Physical-cyber-social systems refers to an intermingling of physical phenomena and digital phenomena in social settings. This physical phenomenon refers to physical actions such as people’s speech and body language, and this digital phenomenon is the product that results from people interacting with machines such as updating code repositories. An example of physical-cyber-social systems (PCSSs) is pair programming (Socha & Sutanto, n.d.), where software engineers physically interact with machines to produce digital products in a socially collaborative environment. WFE is the study of the intricacies and interplay of entities in such a phenomenon.

Extensive resources are inevitably necessary to gather WFE-like datasets. Researchers would need various types of equipment to optimally collect different modes of data, such as a video camera for audio-visual footage, high-fidelity microphones for audio-only recordings, and screen capture software for screen capture recordings. Multiple units of each equipment are needed to make parallel recordings. Fortunately, the declining costs of hardware enables researchers to gather and store high quality, large, and comprehensive data sets with more ease than before. However, the volume of data gathered still presents limitations in viewing, indexing, analyzing, and even just navigating the dataset alone. There are currently tools available in the market to satisfy some requirements but do not yet offer a reasonable process that resolves the unique challenge of navigating a WFE dataset. The goal of the WFE Navigator is to provide a navigation tool that helps researchers navigate through large multi-stream, multi-modal datasets such as the BeamCoffer dataset (Socha, 2015) effectively and efficiently.

## 2 Background

### 2.1 The dataset: BeamCoffer

BeamCoffer is a pseudonym for a Seattle area software development organization whose engineers, activities and office settings are the subject of Socha’s PCSS observation. The dataset gathered from BeamCoffer in January to March 2014 captures material surrounding “software engineers using AGILE practices and collaborating in naturalistic environments” (Socha, 2015). This WFE-like dataset will be referred to as the *BeamCoffer dataset* from this point hence forth.

The BeamCoffer dataset is a large dataset (by 2014’s standards) that contains multi-modal, multi stream datasets. The dataset contains approximately 6 terabytes of data, including videos (380 hours of footage, 981 files in total), audio, photos (1000s), screen capture (292 hours), engineer field notes, interviews, and transcripts. The dataset spans 11 days of concurrent recordings during the organization’s operating hours. A total of 9 GoPro cameras, 6 high-quality Zoom audio recorders, screen capture software, and a handheld camera were used.

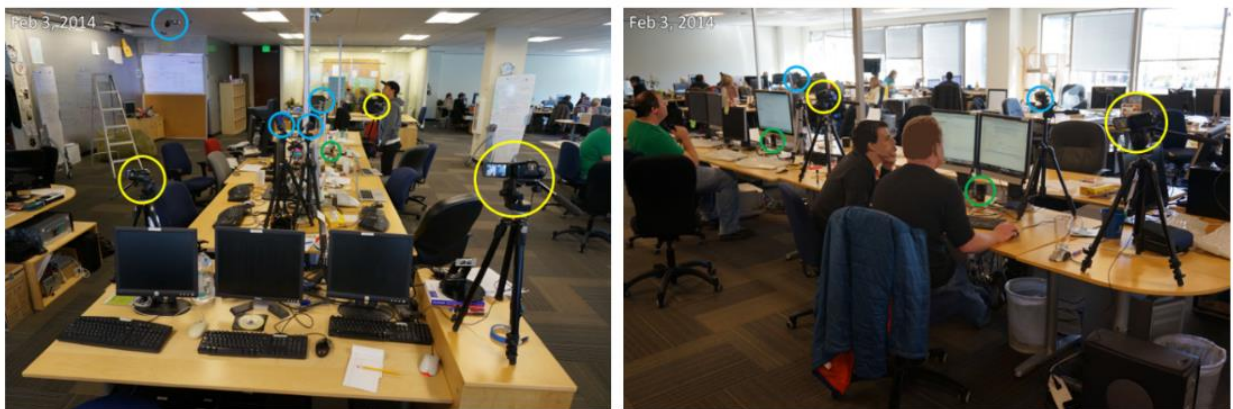


Figure 1. Sample of equipment setup: GoPro cameras (blue), regular video cameras (yellow), and Zoom audio recorders (green) (Socha, 2015).

In WFE, a stream is defined as a “time-based sequence of data from one particular data source” (Socha, Jornet, et al., 2016). The content of a stream may be a sequence of events (moments in time), a sequence of segments (contiguous portions of time), or a combination of both. An example of an event is a photograph or a still-frame from a video. An example of a segment of data is a singular video or audio recording file.

The BeamCoffer dataset’s data of a particular day from a particular location are typically made up of a sequence of multiple contiguous video recordings. We refer to the sequence as a singular unit of stream. The reason behind non-continuous segments of data is limitations in storage. The equipment used in data gathering of the BeamCoffer dataset was only connected to local storage, and the local storage has limited capacity. Once the equipment has reached its capacity, the ongoing recording was stopped and transferred to a storage drive to free up the equipment memory to continue recording the next segment. A stream may contain a sequence of events or segments that are either contiguous or isolated.

Therefore, a stream is an abstraction of a continuous sequence of data, regardless of how the content is assembled.

## 2.2 WFE Ecosystems

The **WFE Navigator** is a tool designed to simplify navigation over large multi-stream, multi-modal datasets. Nonetheless, the WFE Navigator exists as part of an ecosystem and can assist in navigational work only after some pre-processing steps have been completed.

Once the data has been collected, each file is organized into a directory structure determined by the researcher. For the BeamCoffer dataset, Socha grouped the dataset by data collection date, then location, and then equipment. The resulting directory structure forms a stream at each leaf folder directory, like that in Figure 2.

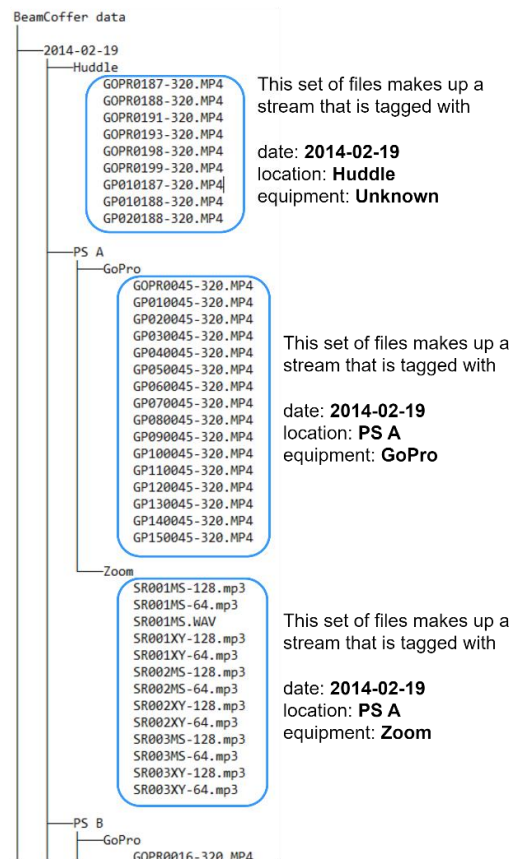


Figure 2. Sample directory structure.

This directory structure is then crawled and tagged using a python script to generate the BeamCoffer metadata database. Each data file is added as a database entry with information related to itself and its source, such as capture start time, capture end time, duration, file name, file type, capture location, capture mode, capture equipment, capture mode, and file type. This database will be referred to as the

BeamCoffer metadata. The python script that crawls the directory structure was developed by Wei Wang in 2016 for his MSCSSE capstone (Wang et al., 2016). Wang's python script creates a SQLite database with the dataset's metadata, where each entry consists of data extracted from each file. The WFE Navigator then uses the BeamCoffer metadata database to determine which portions of the dataset to visualize and play.

## 2.3 Previous work in navigational tools

The WFE Navigator is not the first ideation or implementation of navigational tools for Wide-Field Ethnography datasets. Here, we discuss prior works that led to the current iteration of the WFE Navigator.

Figure 3 shows a view from a developed prototype of a navigational tool made by Soto Ogo in 2014 for his undergraduate CSSE capstone and described in a later work-in-progress paper (Ogo & Socha, 2019). This page is the first page a user sees when viewing the BeamCoffer set dataset using this prototype. Users see a monthly calendar view from January to March of 2014. This range of dates was the same range of dates where this particular BeamCoffer set dataset was recorded. We can see that some dates have an image in their space, while some are empty. The days where there are images are days where there is data. In this specific dataset, there are 13 dates with data: 1 day in January, 11 days in February, and 1 day in March. This was a landing page of the prototype which is conceptually equivalent to a main menu.

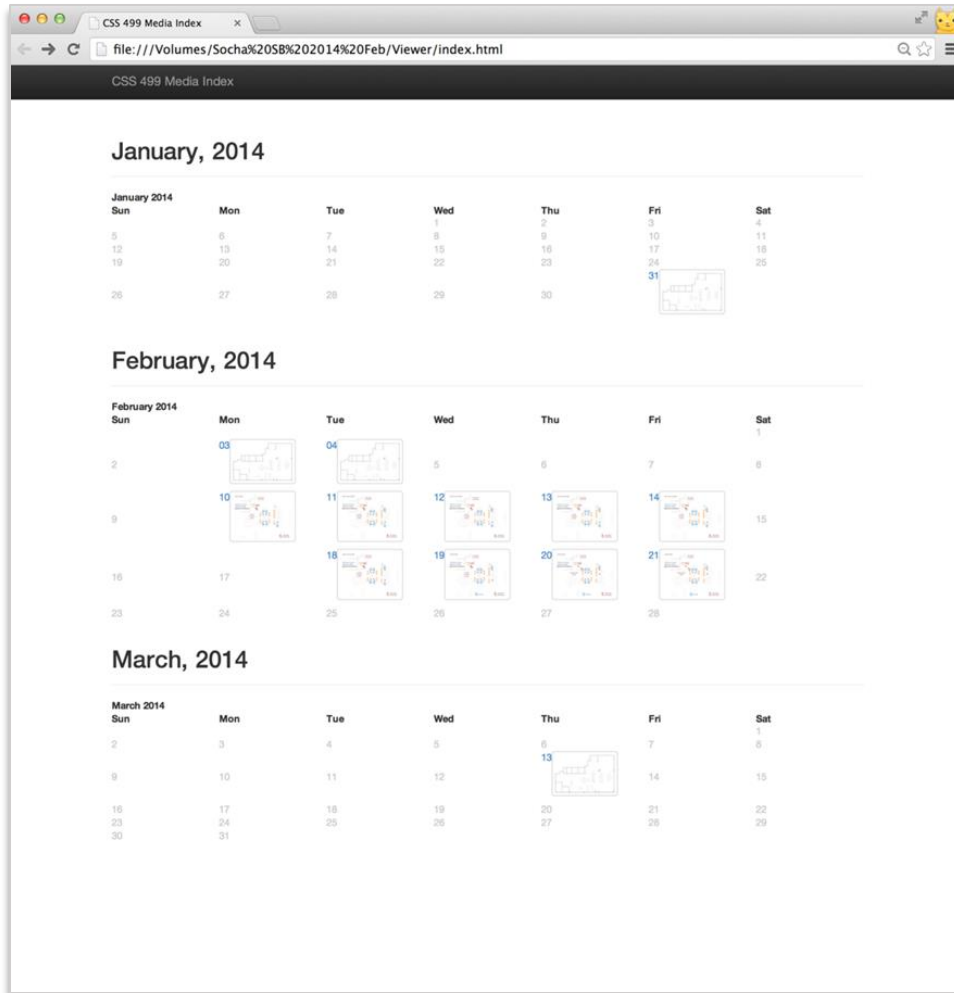
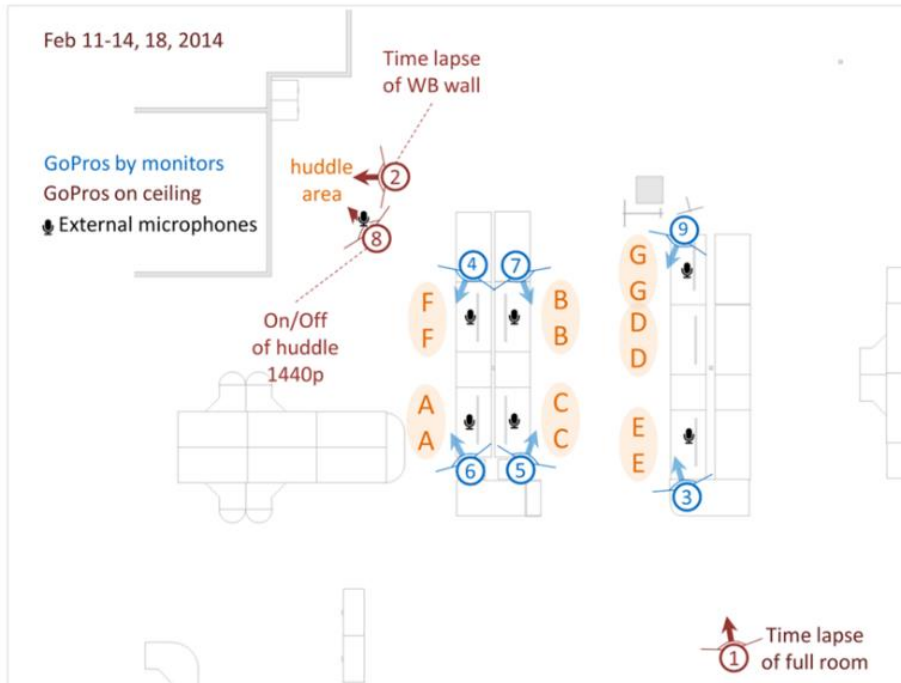


Figure 3. Previous prototype of calendar view (Ogo & Socha, 2019).

When a user clicks on a date that has data, they are directed to a more detailed view for that day as seen in Figure 4. On this page, the user sees a date indicator on the top left, followed by an image of the floor plan specific to the data collected on that day. These floor plan images for each day are the same as the images shown in Figure 3, the monthly view. Each floor plan shows the layout of the location the data was collected at, including information about the equipment used, their location, and setup details.

2014-02-12



Huddle	PS A	PS B	PS C	PS D	PS E	PS F
GoPro	GoPro	GoPro	GoPro	GoPro	GoPro	Zoom
00084-320.MP4	GPFR0040-320.MP4	GPFR0011-320.MP4	GPFR0009-320.MP4	GPFR0006-320.MP4	GPFR0023-320.MP4	SR001MS-64.mp3
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	5 columns
00085-320.MP4	GP010040-320.MP4	GP010011-320.MP4	GP010009-320.MP4	GP010006-320.MP4	GP010023-320.MP4	SR002MS-64.mp3
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	5 columns
00086-320.MP4	GP020040-320.MP4	GP020011-320.MP4	GP020009-320.MP4	GP020006-320.MP4	GP020023-320.MP4	SR002XY-64.mp3
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	5 columns
00087-320.MP4	GP030040-320.MP4	GP030011-320.MP4	GP030009-320.MP4	GP030006-320.MP4	GP030023-320.MP4	SR003MS-64.mp3
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	5 columns
00088-320.MP4	GP040040-320.MP4	GP040011-320.MP4	GP040009-320.MP4	GP040006-320.MP4	GP040023-320.MP4	SR003XY-64.mp3
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	5 columns
00089-320.MP4	GP050040-320.MP4	GP050011-320.MP4	GP050009-320.MP4	GP050006-320.MP4	GP050023-320.MP4	
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	
00090-320.MP4	GP060040-320.MP4	GP060011-320.MP4	GP060009-320.MP4	GP060006-320.MP4	GP060023-320.MP4	
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	
00091-320.MP4	GP070040-320.MP4	GP070011-320.MP4	GP070009-320.MP4	GP070006-320.MP4	GP070023-320.MP4	
16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	16 X 9	
4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	4 X 3	
5 columns	5 columns	5 columns	5 columns	5 columns	5 columns	

Figure 4. Previous prototype of day view (Ogo & Socha, 2019).

Each pair of labels A through G represents a working location, referred to as a Pairing Station, or PS. Each pairing station is a workstation where 2 or more engineers physically gather to work together on a single digital task. One GoPro camera and one Zoom H2n audio recorder are placed by the monitor at some of the pairing stations. The numbers 1 through 9 represent the recording devices, which are further classified by type using colors. Numbers in blue are locations of GoPro cameras placed by the monitors, numbers in red are that of GoPro cameras placed on the ceiling, and the microphone icons are

for external Zoom H2n audio recorders. These audio recorders were used to collect higher quality audio recordings than the audio from the GoPro cameras. Each camera location marked on the floor plan also shows an arc and an arrow, indicating the angle of capture (arrow) and approximate field of view (arc).

Users can browse the data collected at each location using the links listed below the floor plan image. The links are grouped by recording location and equipment. Within each column, the files are sorted chronologically. In Figure 4, the day shown has collected data from 7 locations: the *Huddle*, *PS A*, *PS B*, *PS C*, *PS D*, *PS E*, and *PS F*. Some links from *PS A* to *E* are not shown on this page due to limitations in capturing the screen. Under the links to MP4 video files, there are more links to a collection of still-frames extracted from every 5 seconds of that specific video, available in 3 formats: 16 x 9, 4 x 3, and arranged into 5 columns. These image collections are extracted frames rendered into PDF pages of images in the aspect ratio of 16:9, PDF pages of images in the aspect ratio of 3:4, and a single image file with images arranged into 5 columns (to whatever height is necessary to contain all images). These collections of frames help users to approximately browse the video it corresponds to without opening the file before a navigational tool was developed.

The prototype in Figure 3 and Figure 4 provides a more intuitive way than using the system directory to browse the dataset. The monthly view provides a way to determine availability of data on a daily scale. After the user has selected a date with data to view, the day view provides information about that day's equipment setup, recorded location, and shows all the links to data recorded in that day together in one location. Ogo's prototype largely improves the macro-level data navigation. While this prototype is a fantastic start to navigating WFE datasets, there are many things we cannot accomplish with it. Here are some major shortcomings what we would like to address in this latest iteration:

- **Handling sequences of videos:** Imagine a scenario where a user wants to view the day's recording of pairing station A. The user must navigate to column *PS A* and select a link to open the media file. This action will play that media file on the same page below the lists (not shown). After the file's playback has reached the end, the user will need to return to the top of the day view page to select and open the next video in the sequence. This presentation of data emphasizes the discontinuous nature of the individual files in the dataset, undermining the abstraction of streams.
- **Susceptible to errors:** Typically, a full day of GoPro camera recordings placed at a pairing station contains a sequence of about 17 videos. Hence, the user would need to repeatedly select the correct files to view a day's worth of data of one location. The manual navigation makes the process prone to human errors. A user can easily select the wrong file in the sequence after navigating around the page each time, causing the user to skip or repeat one or more files in the sequence. This error is also not easy to detect once made due to the nonintuitive file names.
- **Finding clusters of data:** There is no indicator of capture time and duration associated with each media file shown on the day view page. This lack of data prevents the user from determining at a glance when each media or stream starts, when each media or stream ends, whether each set of recordings is a continuous stream, whether there is data available at a specific time, and how a subset of data temporally relates to other subsets of data.
- **Finding where to play in video:** If a user wants to view a recording at a specific capture time, the process requires more effort. In this case, the user will need to reference the metadata of each

recording to determine the capture start time and capture end time. Then, the user must calculate the time difference to find the recording elapsed time and browse to it.

- **Finding offsets:** In a scenario where the user wants to navigate around multiple streams concurrently, the user needs to open each file in separate players and navigate each file to the matching recorded time before viewing. Another added complexity to this already complicated routine is that each media file does not usually have the same capture start time, so elapsed time is not useful in traversing content across multiples streams.
- **Handling two or more concurrent videos:** After the user has navigated to the desired time for each file, the user then needs to start playing all the players while making sure that they continue to be in-sync with each other. The more streams the user wants to view concurrently, the more complicated this process becomes. Realistically, a user is not able to control multiple players at the same time with no delays. A workaround for this issue is to over or under correct the elapsed time to match with the timing difference of when each file can start playing or swapping between viewing different players.

After the implementation of the prototype, Ogo envisioned a design concept that resolves some of the issues he encountered. Figure 5 shows a concept sketch of what a better day view could look like. The page would have video or audio players of every location grouped into time segments. This design would make navigating the data set less prone to error. This design sketch also provides some time context, which is an improvement to the lack of micro-overview. However, this design represents time as discrete value ranges, such as “10:00 - 12:00”, but the recordings from each equipment source do not have the exact same capture start and capture end time. Hence, representing the available data in this manner can be misleading. Other potential issues are increasing visual clutter as the dataset grows.

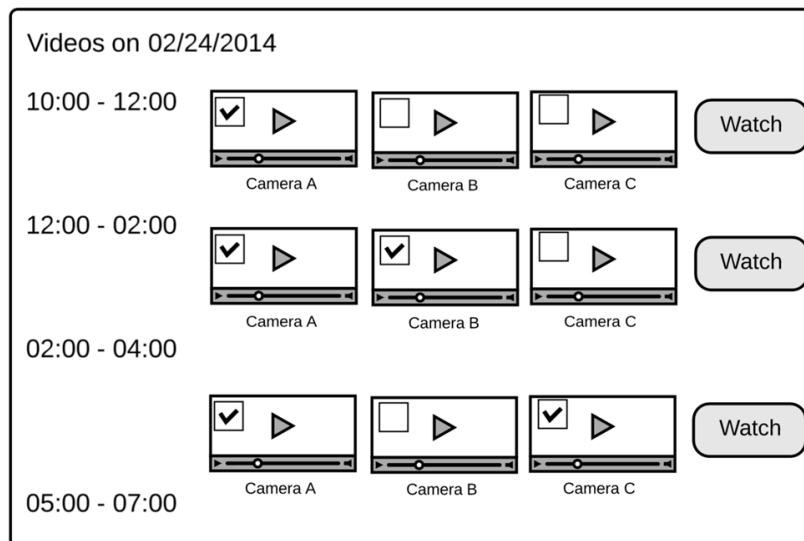


Figure 5. Concept sketch (A) of day view (Ogo & Socha, 2019).

Figure 6 is another concept sketch of a day view. This design concept is a possible vision of a better navigational tool. The visual clutter in Figure 5 is managed by showing only one time segment on the currently viewed page. If the user wants to navigate to a different time segment, the user can click the



“Next” link button to do so. Figure 6 provides a possible solution to the unsynchronized controls issue with a dedicated central control. However, this design does not address how to handle multiple streams that do not have the same start and end time.

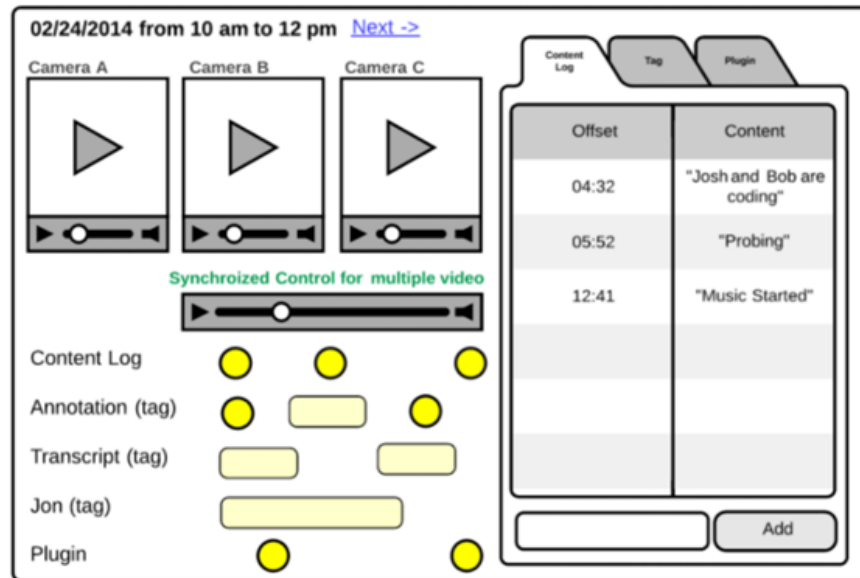


Figure 6. Concept sketch (B) of day view (Ogo & Socha, 2019).

There are existing solutions to address the inconsistent start and end times. One of the workarounds is to preprocess all the media files and slice the files to fit into fixed time segments. However, this process can take a significant amount of time, especially for large datasets like the BeamCoffer dataset. This solution is not flexible to changes such as adjustments to time segments, where one would have to re-process all the files over again.

The major challenge that all these previous navigational work tries to overcome is to present a stream as a cohesive unit while hiding the implementation details. The concept of ‘files’ gets in the way of a user viewing a stream, impeding the user’s ability to view a stream as a continuous and cohesive source.

### 3 What we built

Considering previous design concepts’ strengths and limitations, I created the current navigational tool. Figure 7 is what users should expect to see when viewing multiple concurrent streams on the WFE Navigator, with the separate parts of the user interface identified in Figure 8.

The screenshot displays the WFE Navigator interface with the following components:

- Timeline:** A horizontal timeline at the top shows the playback period from 10:53:20 to 17:33:20 on February 19, 2014. Key markers include:
  - slider starts at...:** February 19, 2014 (Wed) 10:43:29 GMT-0800
  - Current playback time:** February 19, 2014 (Wed) 14:32:28 GMT-0800
  - slider ends at...:** February 19, 2014 (Wed) 17:51:34 GMT-0800
- Stream List:** A list of four streams is shown on the left:
  - PS A, gopro
  - PS G, gopro
  - Unknown, Unknown
  - Huddle, Unknown
- Playback Controls:** A 'Playback speed multiplier' is set to 1. There are 'START PLAYBACK' and 'STOP PLAYBACK' buttons. Below this, it indicates '4 stream(s) in view'.
- Stream View Area:** Four video thumbnails are displayed:
  - PS A:** Shows a live video feed of a conference room. Below it are volume sliders for Left (L) and Right (R) channels, both at 50%, and a 'Volume' slider at 100%.
  - PS G:** Shows a black screen with the text 'file not found' in red. Below it are volume sliders for L and R channels at 50%, and a 'Volume' slider at 100%.
  - Unknown:** A dashed box containing the text 'no media to display'.
  - Huddle:** Shows a live video feed of a group of people in a huddle. Below it are volume sliders for L and R channels at 50%, and a 'Volume' slider at 100%.
- Query Controller:** A search bar at the bottom contains the query:
 

```
AND [Media Type] = [Video] AND [Date] = [19]
```

 Below the search bar is a 'QUERY DATABASE' button. A message states '7 stream(s) match your query'. At the bottom of the query controller are two buttons: 'ADD MATCHED STREAMS TO VIEW' and 'REMOVE MATCHED STREAMS FROM VIEW'.

Figure 7. WFE Navigator showing 4 streams.

Figure 8 shows a the WFE Navigator with 4 video streams in view. From top to bottom, users see the *Timeline* area, the *Playback Controls* area, the *Stream View* area, and the *Query Controller* area. The details of each area are described below.

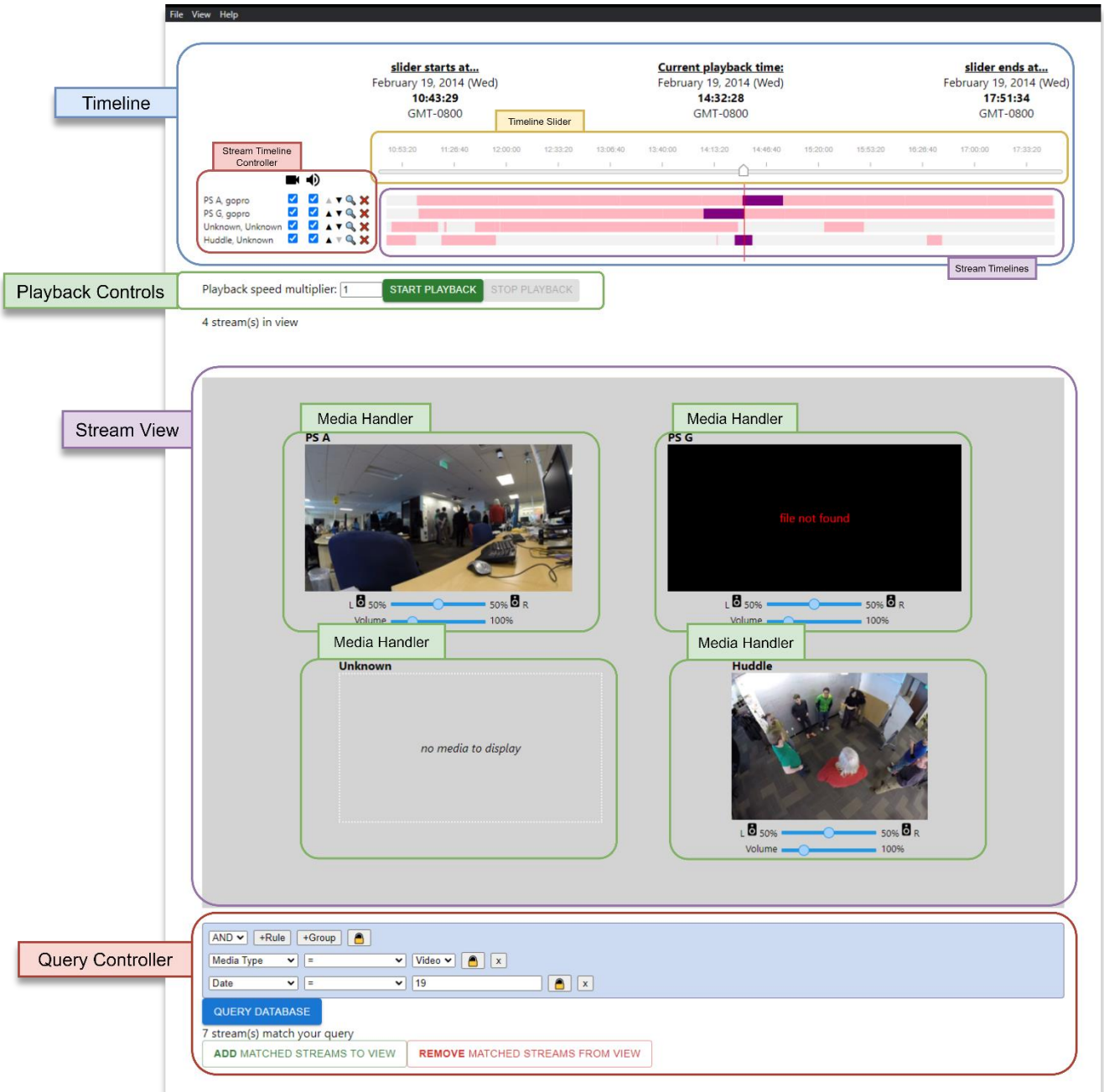


Figure 8. General mapping of screen real estate.

### 3.1 Timeline

Figure 9 shows the *Timeline* portion of the user interface. The *Timeline Slider* is the central point of the *Timeline area* where the user can determine the viewing state of the WFE Navigator.

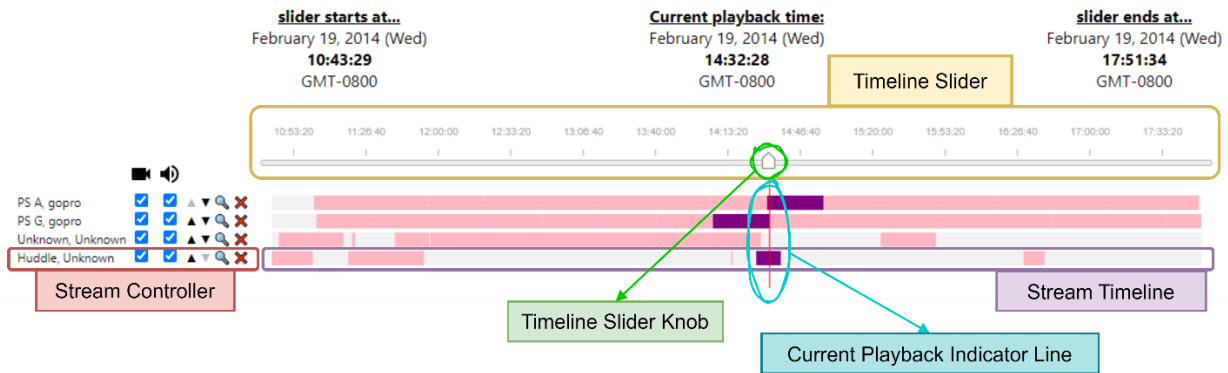


Figure 9. Timeline area elements annotated.

The *Timeline Slider* is a value slider whose value represents the moment in time of all streams being viewed, or time  $t$ . The indicator for time  $t$  is a red vertical line that extends below the *Timeline Slider* over all *Stream Timelines*. This red line signifies what time on the *Stream Timelines* is time  $t$ .

A *Stream Timeline* shows the presence of data in a particular stream. Any period where there is data in that stream, that area on the *Stream Timeline* is shaded pink. Grey areas are the period where there is no data. Segments that overlap with the red line are shaded purple to identify the specific file of the stream that has data of time  $t$ . This purple shading also highlights the slightly different start and end times of the segments across different streams. Users can hover each shaded area to view the file name that particular shaded area is depicting.

Above the *Timeline Slider*, users can see 3 labels, which are the slider start time, current playback time  $t$ , and slider end time. The slider start and end times are dynamically determined based upon the extent of the time range in the selected streams. All 3 labels provide a detailed time format with human-readable date, time up to the seconds, and indicates the time zone of the capture time. The slider start time and end time give users visual context to the scaling of the slider width. The middle label always shows current playback time  $t$ , so users can determine the time state of the streams being viewed in the *Stream View* area to the nearest second. The *Timeline Slider* also shows small slider tick labels directly above the timeline to help browse the timeline more easily.

A *Stream Controller* for each stream is located to the left of each *Stream Timeline*. Figure 9 shows the parts of the controller parts. The controller contains a simplified label for each stream and controls that show/hide the stream, mute/unmute the stream, move the stream controller and timeline up or down in the display, and removing the stream from view. The looking glass icon was intended to provide a larger view of the selected stream, but that functionality has not been implemented.

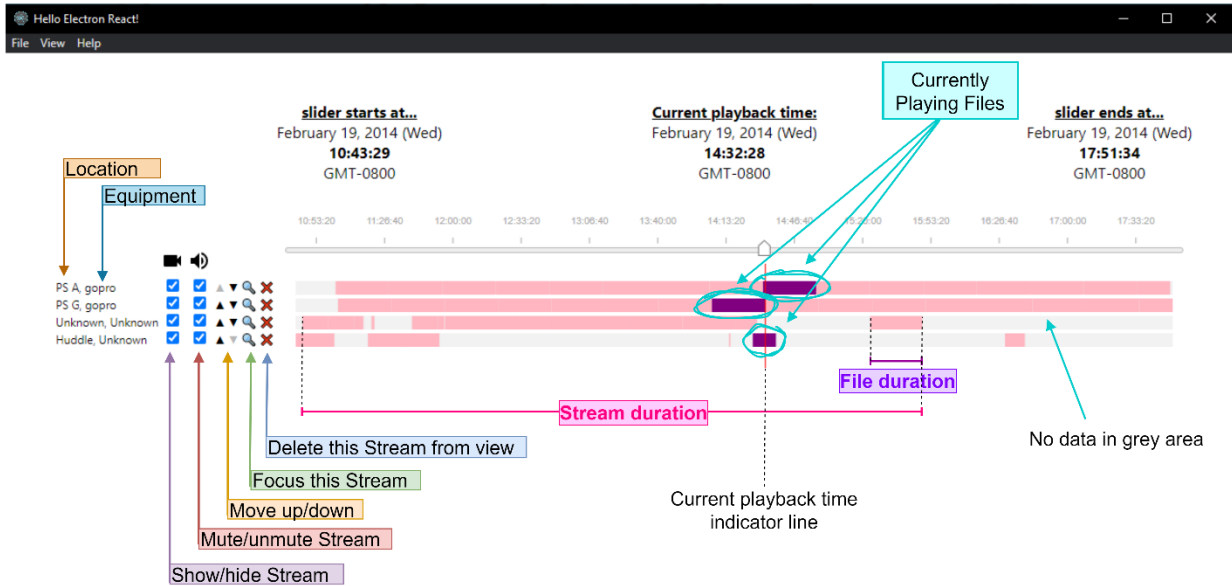


Figure 10. Additional Timeline area elements annotated.

### 3.2 Playback Controls

Figure 11 shows the *Playback Controls*, which are located below the *Timeline* area.



Figure 11. Playback Controller area.

Users can interact with the *Playback Controls* in 2 ways, by changing the playback speed multiplier and to start/stop time  $t$ . Users can input any positive number (decimals accepted) in the text box and click the 'START PLAYBACK' button to start playing all streams in view from time  $t$ . Once the WFE Navigator playback is ongoing, the playback speed multiplier input box and the start button are disabled, and the 'STOP PLAYBACK' button is enabled. When the user stops the playback, the buttons and input box revert to their prior state.

### 3.3 Stream View

The *Stream View* area is where users can see the content of the selected streams. The Media Handler handles each stream based on the type of the source file. In Figure 8, all 4 streams in view are video streams, however, only 3 streams have data to display at the current playback time. The Media Handler labeled 'Unknown' corresponds with the third *Stream Timeline*, where there is no data overlapping with

time  $t$  (red line overlaps with grey area). Media Handlers that have audio data will instantiate with audio controls below the video image. These controls are the panner control and volume control, and each audio control is independent of other media players. Users can choose to pan the audio output of *PS A* stream to their left and the audio output of *Huddle* stream to their right if their speakers support stereo output. A use case like this helps isolate the audio source when playing multiple audio recordings concurrently.

### 3.4 Query Controller

The Query Controller allows users to filter streams based on matches between the input query and the properties of media files within streams. If the user inputs a query like shown in Figure 12, the returned matches will include every stream that has one or more media files of type video and was recorded on the 19<sup>th</sup>. This filtering was a necessary part of the WFE Navigator as there are many streams and choose from in such a large dataset, and finding streams of certain criteria while manually referencing the dataset metadata is tedious. Hence, the *Query Controller* connects the WFE Navigator to the BeamCoffer metadata database. The query controller helps users to systematically build a query to filter the BeamCoffer dataset.

Users can add or remove streams to the WFE Navigator view in batches, and the WFE Navigator will maintain no duplicate streams in view. Below the 'QUERY DATABASE' button, the count of matched streams from user query is stated. Users can choose to add or remove streams to/from view with reference to the matched streams from their query and can only do so if their query returns a match.

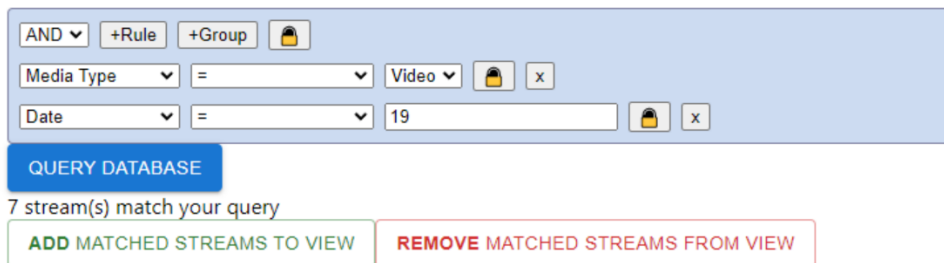


Figure 12. Query Controller area.

For instance, imagine a scenario where the WFE Navigator currently has stream A and B in view, as shown in Figure 13A. Their query results match with a set of streams containing stream B, C, and D. The current state of the streams in view and matched streams is as shown in the yellow circle. If the user adds the matched streams to view, the WFE Navigator adds the difference of sets between the current view and matched streams (streams C and D), resulting in the current view containing the union of both sets (Figure 13B). If the user removes the matched streams from view, the WFE Navigator removes all intersecting streams (only stream B) from the current view (Figure 13C).

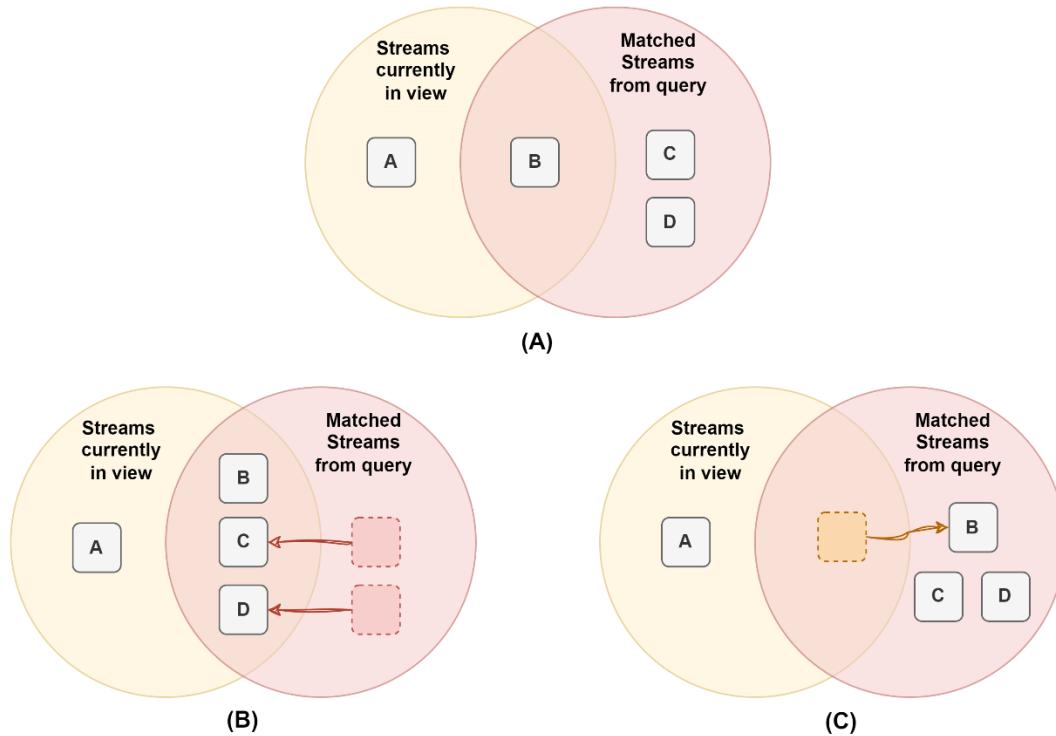


Figure 13. Add (B) or remove (C) streams from query.

The WFE Navigator resolves, among others, the issues with synchronizing player controls across streams of media files. The WFE Navigator uses the BeamCoffer metadata database to inform the arrangement of streams and link its media files for viewing. This means that the WFE Navigator is always showing data in reference to time  $t$ . The *Stream Timeline* shows information about the duration of the stream, which file in the sequence is being played, duration of the file being played, elapsed time of file being played, and content of that stream at time  $t$ . The *Timeline Slider*, *Stream Timelines*, and *Media Handlers* all represent time  $t$  but in differing ways. *Timeline Slider* and *Stream Timelines* show time  $t$  with a temporal context of surrounding data, while the *Media Handlers* show the actual data at time  $t$ . The *Timeline Slider* is the central control to browse streams users have added into the navigator view. When the user moves the slider to a specific time, all elements on the WFE Navigator page will update to reflect data relevant to the selected time.

A substantial change in presentation is representing time in visual format. This visual cue gives users various temporal contexts of the data being viewed. With a timeline, users are able to determine if and when there is data available from which stream, whether data across streams are concurrent, if there are significant gaps in a stream, approximate stream or file duration, and the stream's start and end time in relation to other streams etc. These changes not only provide more information to users but also automate the handling of sequence of files and finding offsets.

This iteration has improved playback by enabling multiple streams to be played at the same time. The WFE Navigator handles hand-offs across files in sequence with no interruptions. Individual volume and

panner controls are added to playback, allowing for more flexibility in amplifying specific streams and control audio output destination in a 2-channel stereo setup. The panner controls open the possibilities such as viewing one stream with audio sent to the left speaker output and another stream with audio sent to the right speaker output concurrently.

## 4 Similar systems

There are various tools available in the space for navigating datasets like the BeamCoffer dataset. However, no single tool supports all the significant features needed for a WFE dataset.

Table 1 is a comparison table that builds upon the analysis discussed in “A System to Support Qualitative Research on Large Multimedia Data Sets” (Ogo & Socha, 2019). The criteria included are features identified to be the highest priority features by Socha, as he was manually navigating the BeamCoffer dataset and realized the need for a system like the WFE Navigator. We can see that none of the tools featured in Table 1 offers all the desired features, except the WFE Navigator.

The criterion in Table 1 are defined as follows:

- **Video/Audio Combining Tool:** The tool has a feature to combine a visual source and audio source from separate files into one singular viewing. (i.e., watching muted video A with audio from video B or voice recording C)
- **Multiple Video View:** The tool can display multiple videos at the same time in the same viewing space, such as within one continuous page. There must be no explicit navigation needed to view each video.
- **Central Controls:** The tool has one control that applies to every media being viewed.
- **Logical Video View:** The tool has a logical order of viewing videos or media. This order can be in the form of playlists or sequencing that is not arbitrary and can be adjusted by the user.
- **Stitched Video Creator/Viewer:** The tool can combine a series of multiple videos into one newly rendered video file, or the tool can view a series of videos as if they were a single file.
- **Search Media by Property:** Users can search media collections by various properties or tags.
- **Local Storage:** The tool can access files in local storage and does not need a remote connection to function.
- **Large Dataset Support:** The tool can support the viewing, searching, and controlling of up to thousands of data files or more.
- **Platform:** The tool supports the listed platform. WIN refers to Windows operating systems. OSX refers to Mac operating systems. L refers to Linux operating systems. Web refers to web-based applications launched in browsers.



Table 1. Comparison table for existing tools based on criteria.

Criteria / Tool	Video/Audio Combining Tool	Multiple Video View	Central Controls	Logical Video View (playlist, sequence)	Stitched Video Creator/Viewer	Search Media by Property	Local Storage	Large Dataset Support	Platform
Panopto		Yes	Yes	Yes	Yes	Yes	Yes		Web
Transana		Yes	Yes	Yes	Yes		Yes	Yes	WIN, OSX
Atlas.ti				Yes			Yes		WIN, OSX, Web
ELAN		Yes					Yes		WIN, OSX, L
AVPlayer		Yes	Yes					Yes	WIN, OSX
VALT		Yes	Yes				Yes		
Ogo's Prototype		Yes		Yes		Yes	Yes	Yes	WIN, OSX
WFE Navigator	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	WIN, OSX

Panopto is a web-based video platform for creating and sharing audio and video content. Panopto is useful in cases such as a university professor sharing a lecture session with two camera feed of his or her face and a screen capture of his or her presentation slides. Panopto offers the capability to view multiple files concurrently, however, it's unlikely to be able to support a dataset as large as a WFE dataset with possible more than 10 concurrent streams. Panopto does not provide the navigation to the next media in the sequence directly from the previous media.

Transana is a well-known and notable tool in the field of video research work. It has the capability to support video and audio file transcriptions, tag moments of interest, import and arrange recordings in fit the project's needs, synchronously plays multiple video and audio files as well as the transcripts, and provides robust support for collaborative annotation. Nonetheless, Transana is limited to a few media files at a time and has no support for streams of media. Transana's strength is in its annotation capabilities. The preparation work that comes before annotation and analysis can take place is not well supported nor automated using Transana. Users must first import the files they want to look into, so Transana requires the user to already know which part of the data will be analyzed.

Both Panopto and Transana are tools that offer the most features Socha was looking for based on previous analysis. Panopto, Transana, Atlas.ti, and VALT are all proprietary software, which would limit custom changes we might want to add. ELAN and AVPlayer are open-source and free of charge but offer even less features than the other systems. None of the systems we found is contained and comprehensive (not using multiple tools to achieve one complete course of the required action) enough for WFE datasets. Most importantly, none of the systems that exist today are able to present the abstraction of a stream from a collection of discrete source files.

## 5 System Design

### 5.1 Overview of Application Components

The WFE Navigator is developed using the React framework. Per React's best practices, the application is made up of multiple smaller modular components. Figure 14 is a breakdown of the React components of the WFE Navigator. There are two groups of components in this diagram. The ones arranged above the *App* are native React objects, which are components that exist by default in the React framework. The remaining ones are domain specific entities created for the WFE Navigator. The application has one access point: *App*, which is a native React component. All other components can be accessed by going through the main *App* component. Each component represented in Figure 14 is a custom React component unless annotated otherwise.

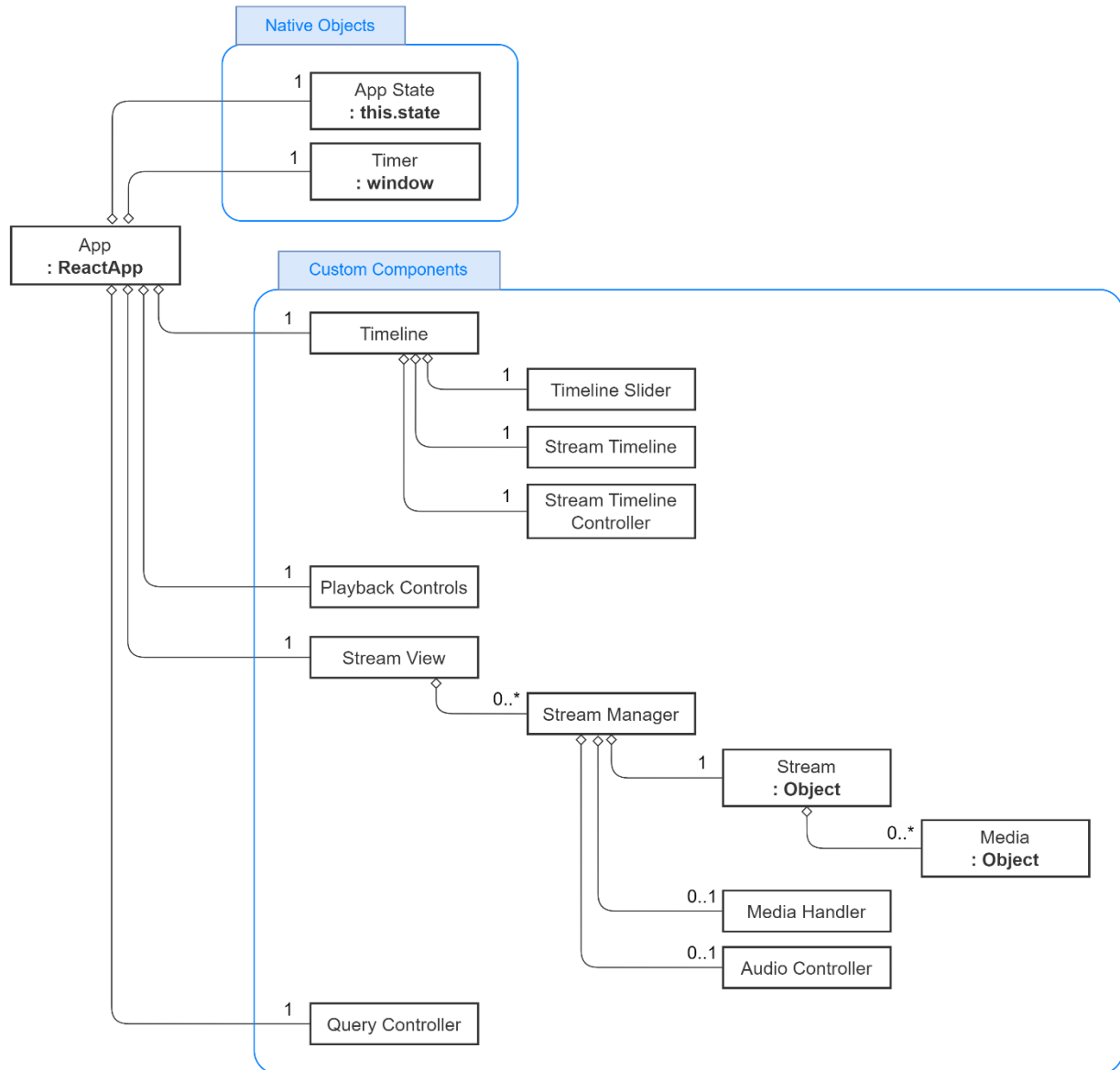


Figure 14. Application Domain Model showing relationships to the React framework.

Figure 14's custom components consist of 4 main functional components: *Timeline*, *Playback Controls*, *Stream View*, and *Query Controller*. These 4 main functional components are located on the leftmost side of the enclosed custom components. *Timeline* and *Stream View* are React container components. A container component is a component whose main or sole responsibility is to contain smaller components. Each component is focused on doing a single thing. This is in alignment with the Single Responsibility aspect in the SOLID principles (Martin, 2003).

The annotated figure below is the overall mapping of screen real estate to the components mentioned. Components that are encapsulated within another component are contained by that component. For

instance, the boundaries of the *Timeline Slider*, *Stream Controller*, and *Stream Timeline* are fully covered by the boundary that indicates the *Timeline* component.

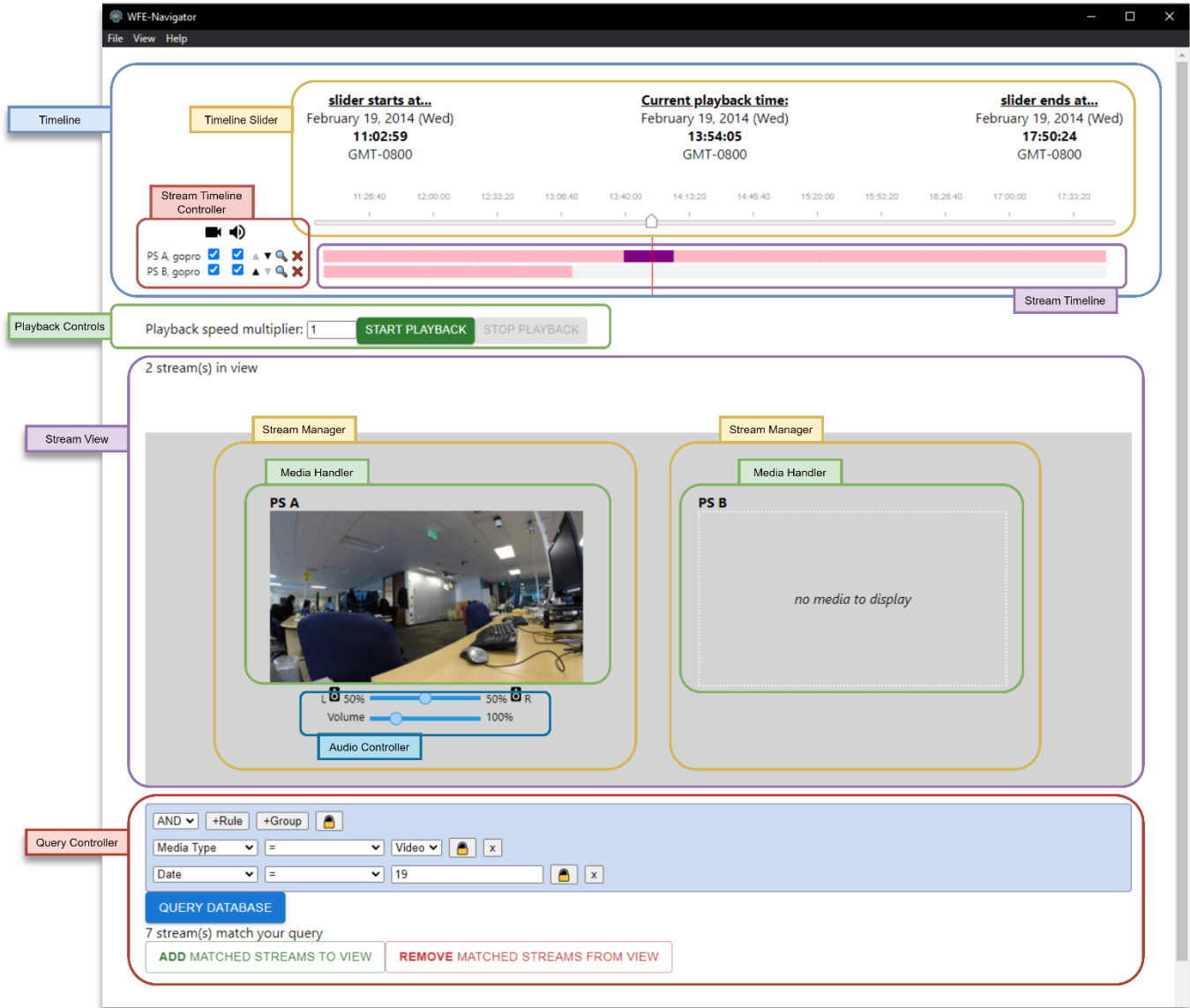


Figure 15. Mapping of screen real estate to components.

The *Timeline* container is responsible for presentation of timeline-related components. The *Timeline* contains exactly three components: one *Timeline Slider*, one *Stream Timeline* component, and one *Stream Controller*. The *Timeline Slider* includes the value slider (Figure 16) that indicates time  $t$  and the start, current and end labels above it. The slider knob represents time  $t$ , the specific state the WFE Navigator is presenting. The user can move the slider knob left or right to change time  $t$ , which will

propagate changes to all the other components. Similarly, during ongoing playback, the Timer updates time  $t$ , and the slider knob will also move along accordingly.

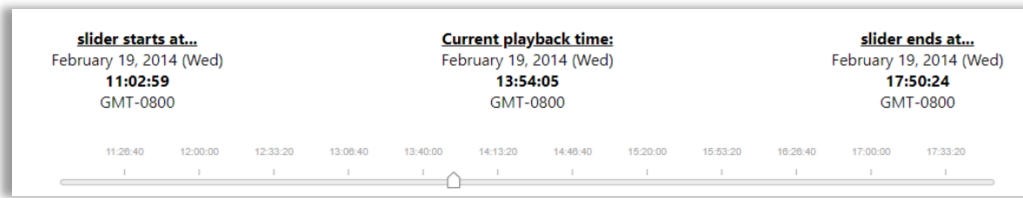


Figure 16. Timeline: Timeline Slider component

*Stream Timelines* includes the visualization of viewed streams as adjacent timelines. Each Stream is represented by rows. Each Media is represented by color-coded bars within each row. When time  $t$  overlaps with a Media, that Media bar will be represented with a different color. The *Stream Timeline* also features a vertical red line, which is the value indicator that corresponds with the slider knob in *Timeline Slider*.

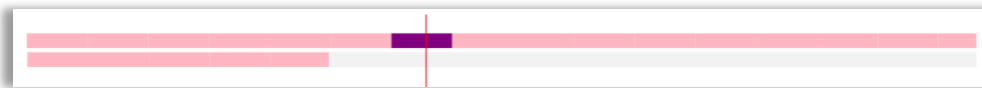


Figure 17. Timeline: Stream Timelines component

*Stream Controller* renders the options to show media, mute media, re-order media, focus, and delete media from view for streams in current view. The options are grouped by rows and pairs with each timeline row of a stream. Both the *Stream Timelines* and *Stream Controller* display a set of stream data instead of one per component.

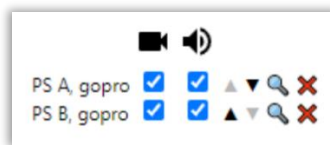


Figure 18. Timeline: Stream Controller component

The *Playback Controls* is where the user can change the playing state and the playback speed of the WFE Navigator. The user can input a desired playback speed before starting playback. The playback speed must be a non-negative value greater than zero. When the user triggers the start button, the playback speed will be locked and cannot be changed until the user has stopped the current playback. The start button is enabled when the application is in a stopped state and disabled when the application is in a playing state. The opposite applies to the stop button.



Figure 19. Playback Controls component

The *Query Controller* allows users to add or remove streams from the current view. The *Query Controller* sends queries to the database and captures the returned data into the *App State*. The message below the 'Query Database' button indicates how many streams are stored in *App State* memory. Only when there is queried data saved in memory will both the add and remove buttons be enabled. The user can then add the streams from *App State* memory that are not currently in view to the current view. The removal option removes all streams in current view that are present in the queried data.

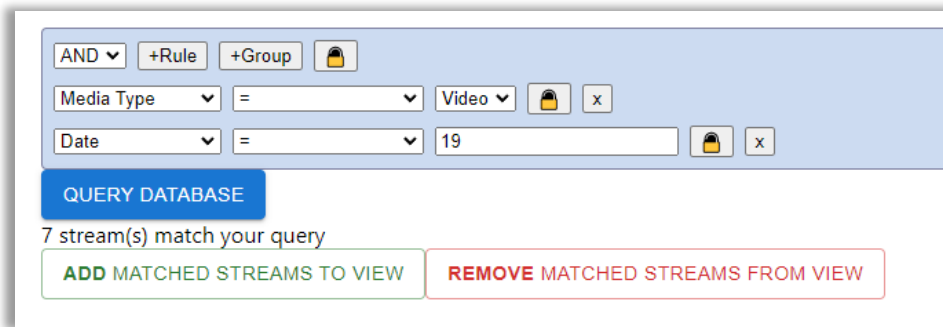


Figure 20. Query Controller component

Unlike *Timeline*, the *Stream View* may contain zero, one or many *Stream Managers*, depending on whether the user has added any Streams into view. *Stream View* is a container component for *Stream Managers*. A *Stream Manager* is tied to a Stream object upon instantiation. Each *Stream Manager* contains one *Media Handler* and one *Audio Controller*. However, *Media Handler* and *Audio Controller* are only instantiated when its corresponding Stream has media content at the *App State* time  $t$ . Based on the viewing options specified by the user in the *Stream Controller*, the *Media Handler* and *Audio Controller* will display different results. *Media Handlers* and *Audio Controllers* are displayed independently of each other. An example of some cases is shown in Figure 21. Stream View component For detailed breakdowns of Stream and Media object, see Figure 31 and Figure 32.

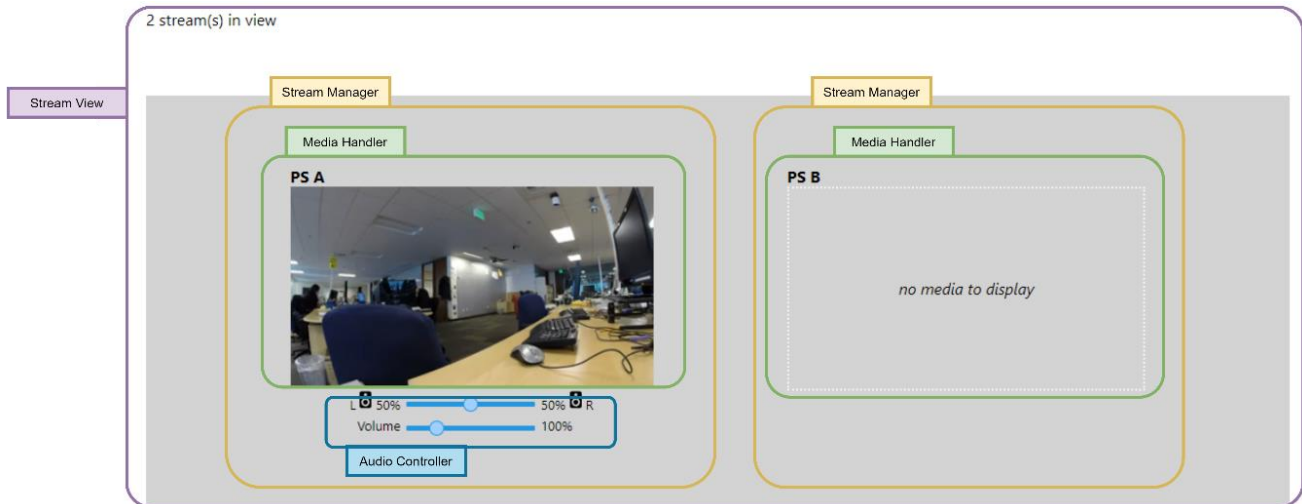


Figure 21. Stream View component

## 5.2 How Data Flows through Components

Data in React applications flows from higher-level components to lower-level components. We define higher-level components as containers of lower-level components. Figure 22. Data access by component shows an illustration of React component hierarchy of the WFE Navigator. The items inside each component area are displays of data used by that component. For instance, *Timeline* shares its top border with *App*, so *Timeline* is a child component of *App*. Similarly, *Timeline* shares its bottom border with three components: *Timeline Slider*, *Stream Timeline*, and *Stream Controller*, making them child components of *Timeline*. *Timeline Slider* and *Stream Timeline* have an overlap usage of some data that is inside *Timeline* or *App*; however, *Timeline Slider* cannot see the value of *sliderRange* and *masterTime* that *Stream Timeline* is displaying and vice versa. Note that data is passed one level at a time. While *Timeline Slider* is a child component of *App* by proxy, *Timeline Slider* cannot access any data that *App* has not made available to *Timeline*.

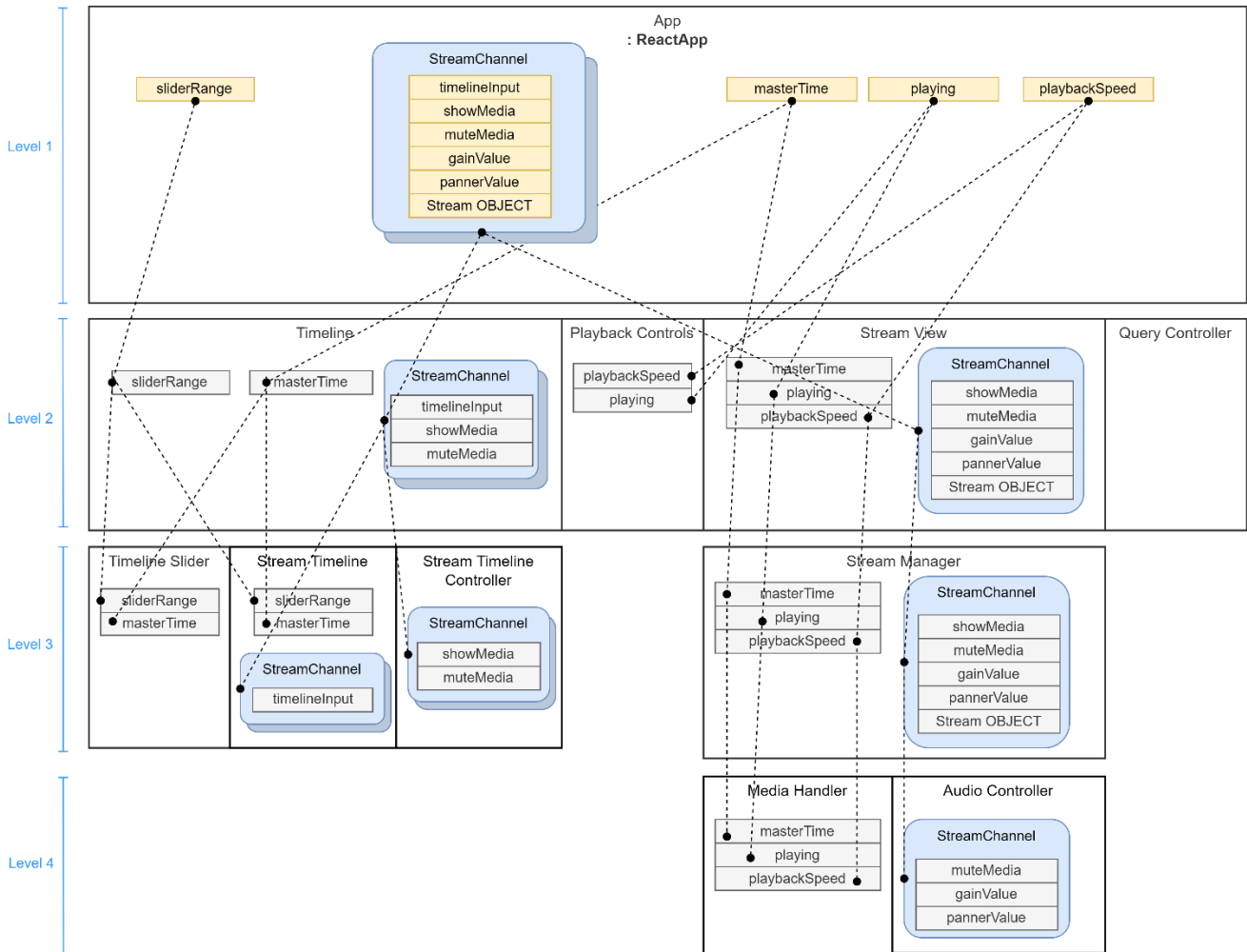


Figure 22. Data access by component.

Only one component exists at the first level: `App` (where `ReactApp` indicates that this is the root render component). The `App` component is the access point to all other components in levels below it. In React, data can be passed down from higher-level components to any lower-level components it contains. This means that every component has access to all the data available to their immediate parent component. The color-coded items indicate that there is only one copy of each value, and this data is stored in the first level. Recurring data values seen in Figure 22. Data access by component, such as `masterTime`, indicates read access to the original data in `App`.

### 5.3 How Playback Works

The central interaction of the WFE Navigator application is the high-level navigation and playback process. The high-level navigation is conveyed using a color-coded timeline in the `Timeline` area, and



*Media Handlers* in the *Stream View* area. The player could be empty or showing media. The *Timeline* shows where there is media to play. At a higher level, the playback process mimics the result of stitching a sequence of multiple media files together into a larger singular source for playback. However, the WFE Navigator is solely about viewing and navigating data. It does not currently make any changes to the data being viewed. It does not create any new media files nor make changes to any source files. Instead, the WFE Navigator dynamically switches to the correct media file during playback.

Figure 23 is a system sequence diagram of the WFE Navigator playback process depicting the starting, continuing, and stopping of the playback process. The entities represented in Figure 23 are React components unless stated otherwise in the figure.

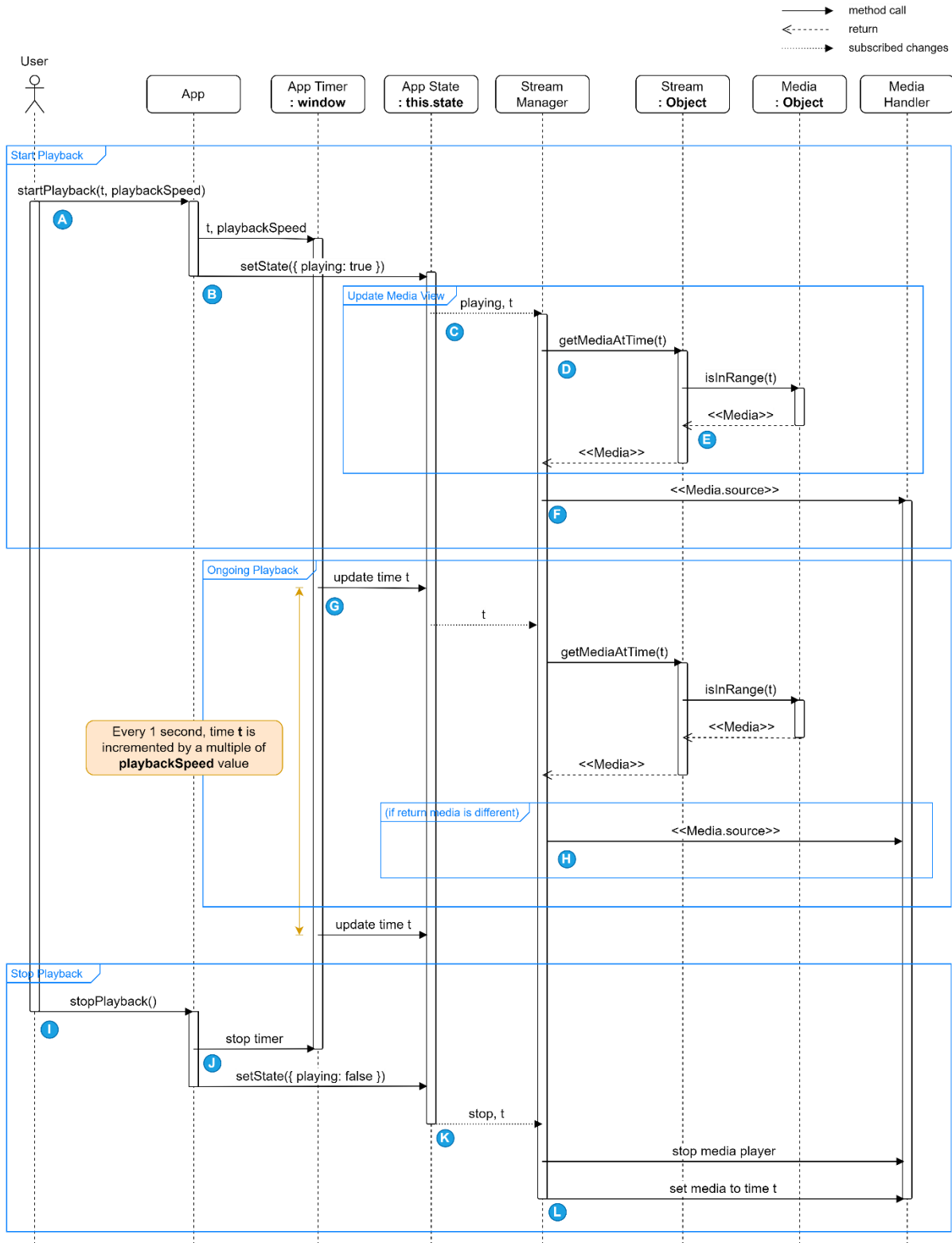


Figure 23. Playback Sequence Diagram.

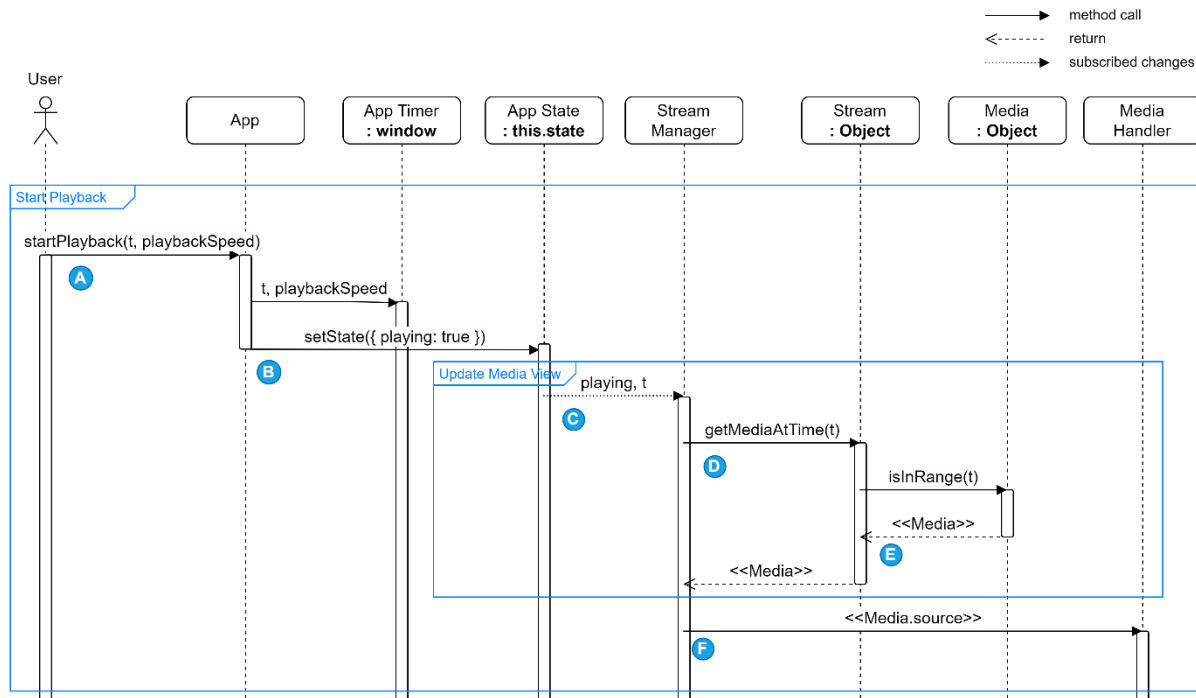


Figure 24. Playback Sequence: (1) Start Playback

The first interaction shown in Figure 23 is the starting of the playback process from a stopped state. Note that since the WFE Navigator follows the ReactJS framework, the *Stream Manager* subscribes to changes in the application state instead of the application triggering method calls to it.

The playback start proceeds as follow (refer to Figure 24, which shows the top portion of Figure 23):

- (A) The user triggers a playback button on the main application.
- (B) The application sends the selected start time *t* and playback speed to the JavaScript Timer. This starts a repeating call that updates *t* every second and sets the application playing state to *true*.
- (C) The *Stream Manager* listens to changes in application playing state and start time *t*.
- (D) When the *Stream Manager* notices a change in playing state or *t*, the *Stream Manager* checks its associated stream object for any media content present at time *t*, which is done by checking all *Media* objects within the *Stream* object for any content present at time *t*. If a *Media* is in range of time *t*, it implies that time *t* is within the start time and end time of that particular *Media* object.
- (E) If there exists a media in range of time *t*, that *Media* object is returned to the *Stream Manager* through the *Stream* object.
- (F) The *Stream Manager* dispatches the media source to the *Media Handler* for playback.

In an ideal case where there is Media at time  $t$  and the source is valid, the player will produce a player interface like in Figure 25.

There are several other possible values that can be sent to the media handler at this step. Each of these scenarios has a separate user interface feedback to indicate the user settings and communicate specific errors. If there is no media content present at time  $t$ , the *Stream Manager* will not receive any returned media, and the *Media Handler* will not be called. This interaction will result in the player displaying a message like in Figure 26. If there is a corresponding Media, but the user has selected to hide the content, the *Media Handler* would also not be called. The user will see a message like shown in Figure 27 and not see any player or audio controls. In the case where there is Media at time  $t$ , but the source is invalid, the *Media Handler* will be called and will return an error. The *Media Handler* will handle this error by hiding the player and showing feedback like in Figure 28.

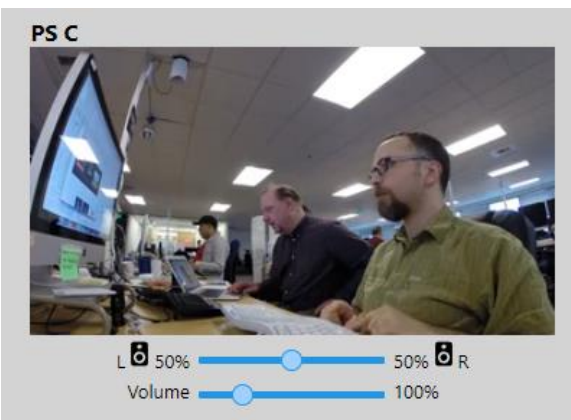


Figure 25. Player UI when media content is available.



Figure 26. Player UI when there is no Media at time  $t$ .



Figure 27. Player UI when Media is hidden by user.

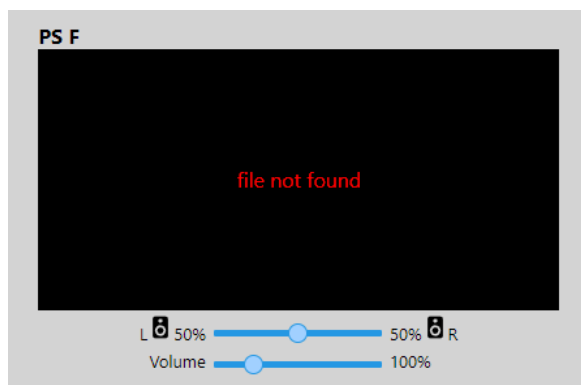


Figure 28. Player UI when Media source is invalid.

In the following sections, we will use the term *Update Media View* to refer to the sequence covered by (C) through (E) in Figure 24.

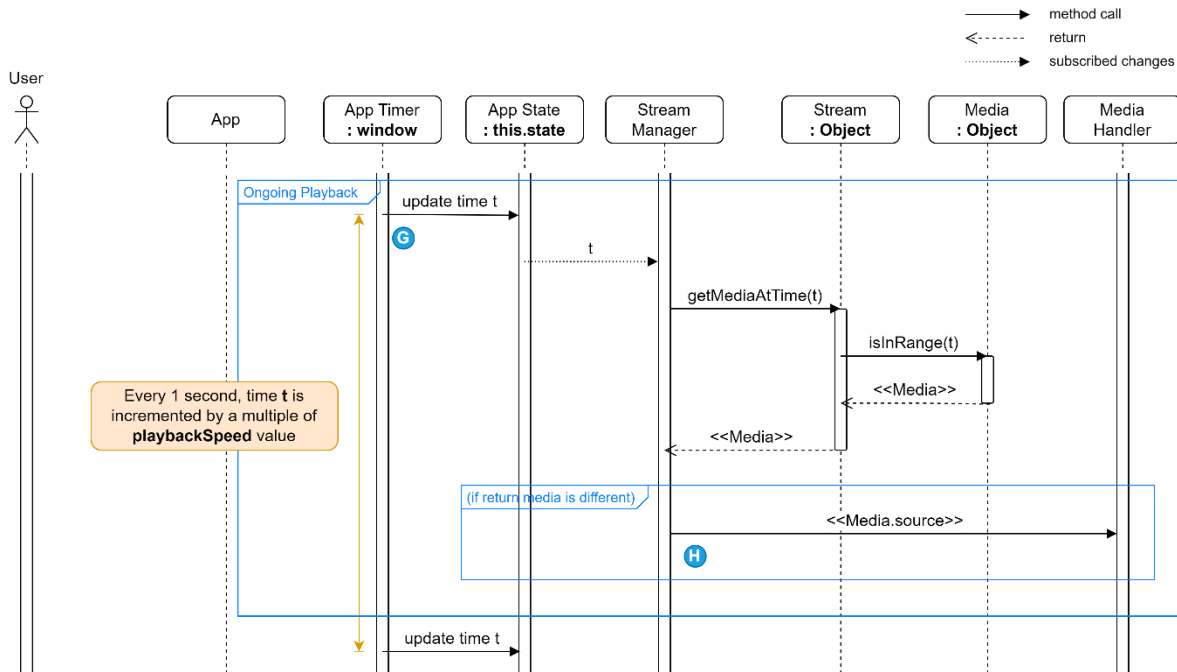


Figure 29. Playback Sequence: (2) Ongoing Playback

The second interaction depicted in Figure 23 is an ongoing playback. After the internal timer has started in (B), the first event to update the application state time  $t$  will fire after one second and will continuously do so every one second. The following order of events happen between the entities at every update to application state time  $t$  until the user stops the playback process or time  $t$  has reached the end of its range (refer to Figure 29, which is the middle part of Figure 23):

- (G) Time  $t$  is incremented by a multiple equivalent to the playback speed value and the timer interval, which is one second in this case. For instance, if the user starts the stream with a 4x playback speed, then every 1 second, the internal timer will update time  $t$  by 4 seconds.
- The *Update Media View* will run as described in the prior section.
- (H) If the returned Media object is not the same Media object that was playing previously, the new Media object will be played starting at its corresponding elapsed time to time  $t$ .

Any subsequent returns from *getMediaAtTime(t)* with the same media object, or the lack thereof, will result in no action to the rest of the playback entities. This deliberate behavior is introduced to minimize unnecessary interruptions to the media player, especially in cases when playback is ongoing and multiple players are shown. As time  $t$  is constantly updated, the *Stream Managers* are constantly trying to play each media at a particular point in time, but each player will not receive the method call at the same time. Even if each method call can be completed very quickly, the time elapsed may accrue and stack as it reaches the last player affected. These delays are also very noticeable when viewing mediums with continuous streams of data such as video and audio.

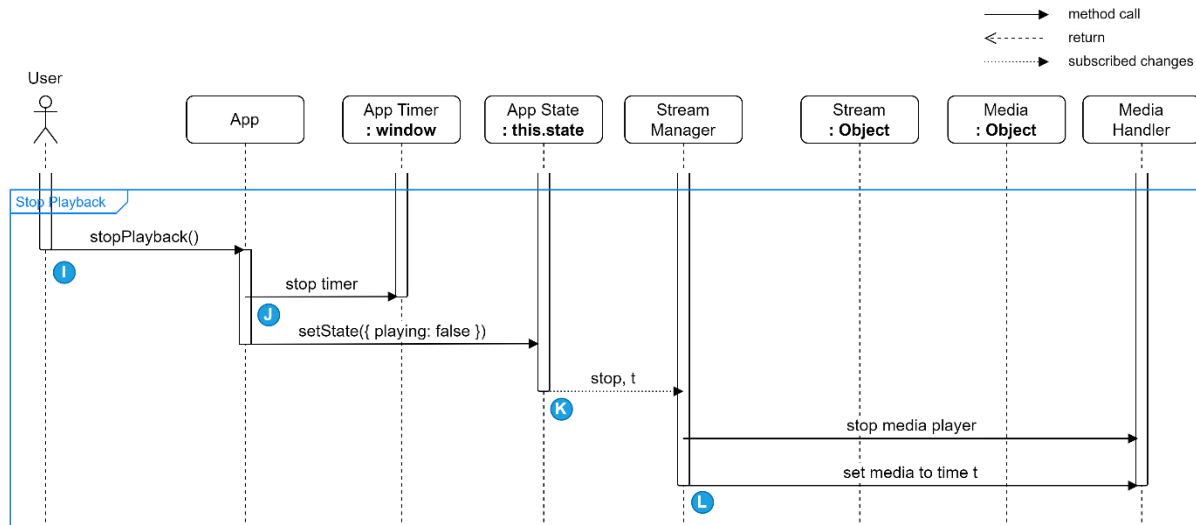


Figure 30. Playback Sequence: (3) Stop Playback

The last interaction shown in Figure 23 is the stopping of playback. The sequence of event is as follows (refer to Figure 30, which is the bottom part of Figure 23):

- (I) The user triggers the stop playback action.
- (J) The *App* stops the JavaScript timer and sets the *App State* playing to *false*.
- (K) The *Stream Manager* notices the changes to the playback state and notes the last known time *t*. The last known time is where all media should be paused at.
- (L) The *Stream Manager* signals the *Media Handler* to pause the player and recalibrate all showing Media to the last known *t*. This action is added to compensate for delays in earlier playback state changes that may have accumulated.

There are two parallel actions maintained independently throughout the playback process. The first action is the constant update of time *t*, which is managed by the system timer. The second action is the conditional start and stop of the media player based on its source media's metadata.

In the initial iteration, keeping time with the Timer seems like an unnecessary function, however, it became the core to circumventing the limitations of existing media player libraries' API. The *Stream Managers* manage time offsets of a stream based on the timer. When playback starts, the *Stream Manager* will start the media player (through the *Media Handler*) from its elapsed time that corresponds with whatever time *t* was when playback became active. The *Media Handler* does not know about time *t* after playback was started. When transitioning between media objects, the *Stream Manager* knows when to start the next media object via the timer, as the timer still keeps track of all the time updates during the time gaps. When the *Stream Manager* triggers the playback of the next media object, the *Media Handler*, again, only knows to start playing and from where, without knowing time *t*.

All media player libraries that were considered for WFE Navigator’s development were developed for displaying one player at a time. The media player library’s API accepts a default start state of the player, but one persistent default state is not enough to fully synchronize the controls between multiple streams. This is due to the nature of each stream consisting of a sequence of multiple files with arbitrary lengths of gaps in the capture times between each sequential pair of media files. None of the features offered by the media player library’s API provides a way to inform the media player of these empty gaps. This limitation is worked around by having the player component listen to both the *App* playback state and the internal system time *t*. The Timer is needed to determine which media, if any, to play at what elapsed time. The *App* playback state is needed to determine whether to play the media or not. Without one or the other, the *Media Handler* is unable to resolve what action to proceed with, because the player cannot play a media without knowing where in the media to start playback or how long to wait before it can start the next media in the sequence.

## 5.4 Domain specific models

Figure 31 is a domain model of a WFE Stream object. The stream object is a custom class object that contains information related to a WFE stream. The stream object contains the following properties: sequence of media, earliest time, latest time, location, equipment, date, and description.

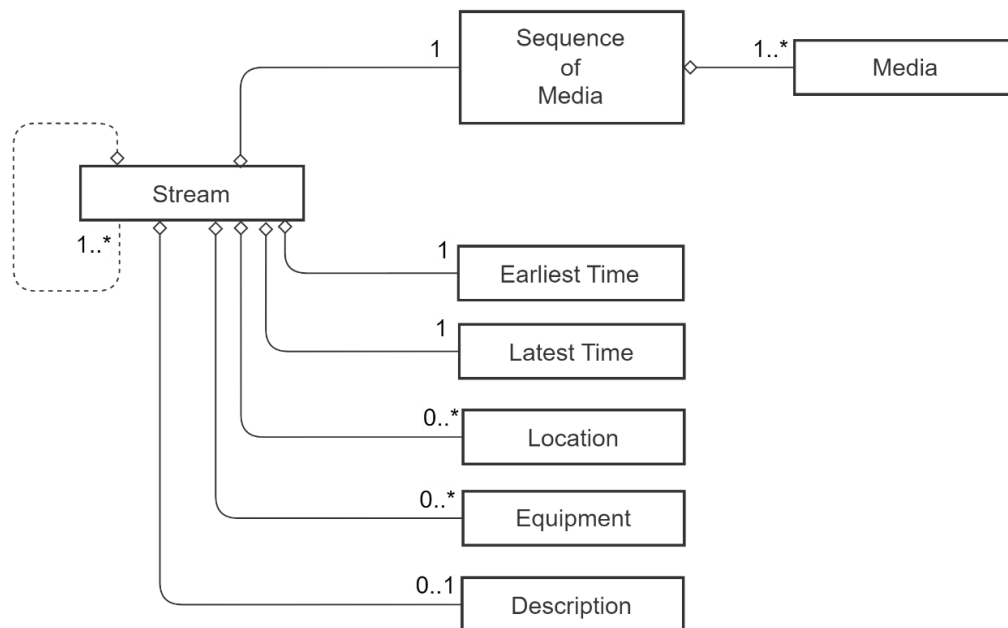


Figure 31. Domain model of a WFE Stream.

A WFE stream may be a collection of media or a collection of streams (dashed line). As the current state of the WFE Navigator only handles a stream as a collection of media, a stream will be defined as a collection of media objects in discussions going forward. A notable relationship depicted in Figure 31 is that a stream object must contain one or more media objects, so a stream object cannot be empty.

Though the condition is not conveyed explicitly, the associated collection of media objects must have no duplicates.

A stream object may have zero or more values for location and equipment. Since a stream contains a collection of media objects, the location and equipment values are derived from distinct values of associated media objects. It is possible for location and equipment values to be unknown. The Location also may be inaccurate if the equipment was moved while recording a single media file.

The earliest time and latest time fields in the stream object are derived from the earliest and the latest timestamp of the collection of media objects it contains. Since both the earliest and latest time fields are derived values, both fields will never be empty. It is important to note that the duration based on the earliest time and latest time fields does not reflect the duration of viewable data in the stream. The calculated duration only represents the overall temporal range a stream object covers.

Figure 32 shows the domain model of a WFE Media object. The Media object is a custom class object that contains information relevant to an individual media file. An instance of a Media object represents one file of collected data. Every property of a Media object is not derived from another object, unlike the Stream object.

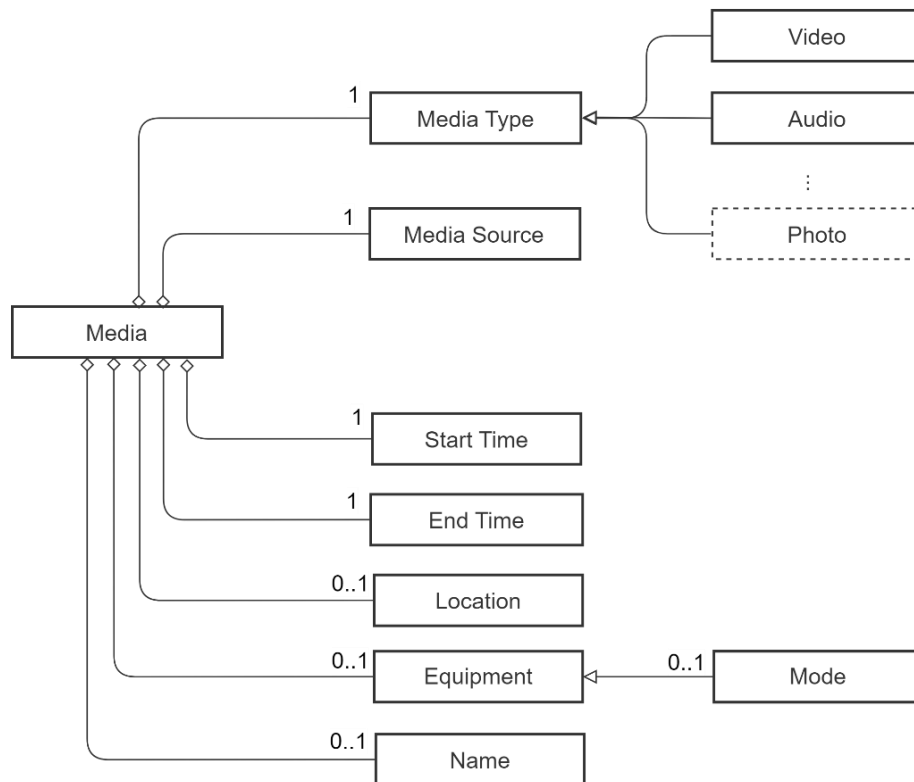


Figure 32. Domain model of a WFE Media.

A WFE Media object contains the following properties: media type, media source, start time, end time, location, equipment, and name. Since a Media object is a reference to a media file, a Media object can



only be classified as one data type. Possible values for media type includes, but are not limited to, video, audio, photo, and text. Currently, the WFE Navigator can handle video and audio media file types. Media type of Photo is encapsulated by a dashed line in Figure 32 indicates that it has not been completed in the current state of the WFE Navigator. The media source property is a text field that points to the location of the media file in local storage.

A Media object must also have associated temporal data to be useful. The duration property is a pair of datetime values, the start time of the recording and the end time of the recording. The date property is stored in numerical values and is most used as labels for display.

Location and equipment fields may have zero or multiple values but are most likely to have one value. In the BeamCoffer dataset, possible values for location are *PS A*, *PS B*, *PS C*, *PS D*, *PS E*, *PS F*, *PS G*, *Huddle*, and *Unknown*. The possible values for equipment are GoPro and Zoom. Only some equipment has specific recording modes. For instance, audio recordings were collected using Zoom on both MS and XY modes. GoPro cameras were not recorded using any specific settings, and so the mode was left blank.

## 5.5 Custom Classes and Methods

Figure 33 shows the properties and methods of WFE Stream and Media objects. Each box represents an object class with the top half listing the property fields and the bottom half listing the class methods. The property fields depicted in Figure 33 is largely the same as shown in the domain models (Figure 31 and Figure 32). The model also shows the one-to-many relationship between Stream object and Media objects. In the BeamCoffer dataset, a stream must have one or more media objects. However, a media object may be associated to any number of streams or none.

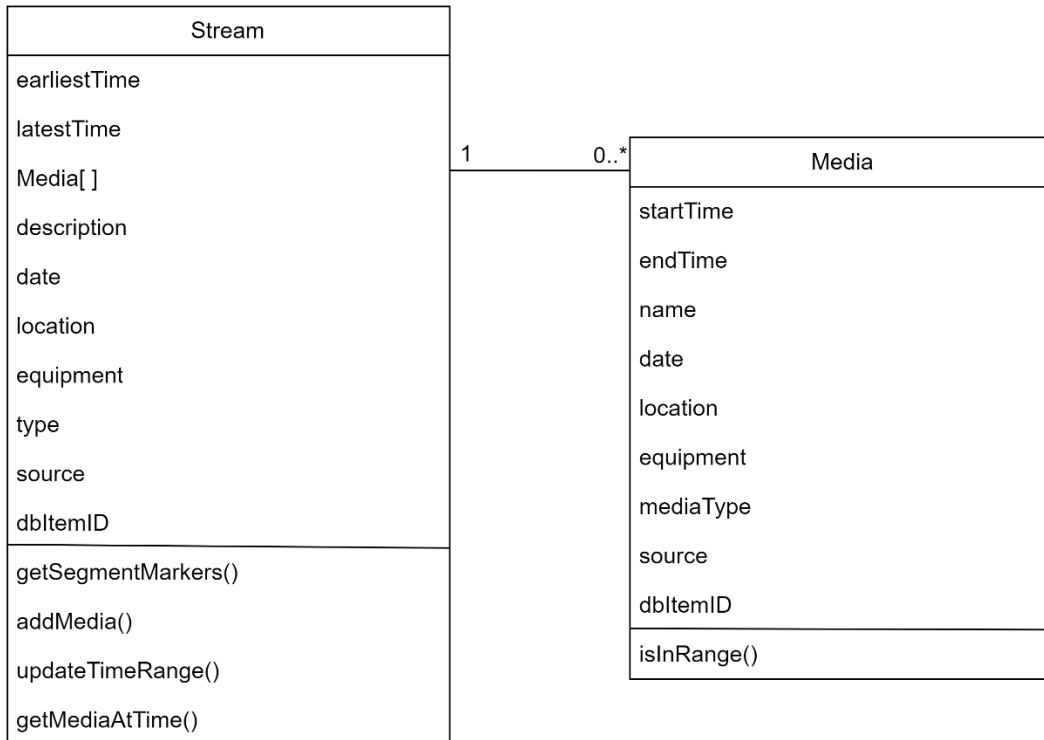


Figure 33. Properties and methods of Stream and Media objects.

The most interesting method in the implementation of these custom classes is *getMediaAtTime(t)* method in the Stream object. *getMediaAtTime(t)* is the driving method for Update Media View (Figure 24) and gets called every update cycle (arbitrarily set to 1 second in the current system) when playback is active. Input *t* is the current playback time in Unix time as an input and returns a Media object or null otherwise. *getMediaAtTime(t)* loops through the Media object array of that Stream object and calls *isInRange(t)* on each Media object in order of Media object start time, where *t* is the current playback time in Unix time. The loop continues until Media file that contains data at time *t* (time *t* is within range of the Media object's *startTime* and *endTime*) is found and returns that Media object; or no Media object is within range, and null is returned. A *Stream Manager* handles the returned value and displays the data accordingly.

If a Media object is returned, the *Stream Manager* passes the Media object to both the *Media Handler* and *Audio Controller* for display in the *Stream View* area. If a null value is returned, the *Stream Manager* will display a UI message, and neither *Media Handler* nor *Audio Controller* is instantiated. When playback is not active, *getMediaAtTime(t)* is called when a user moves the *Timeline Slider* knob to a different time *t*.

## 5.6 Management of Audio Branches

Another implementation detail I would like to discuss is the audio routing of the WFE Navigator. As the WFE Navigator is a browser-based application, the WFE Navigator needs a web-compatible audio tool that supports both volume adjustments and panning of 2-channel audio outputs. Web Audio API is a

JavaScript API for processing and playing audio in web applications pioneered by Google. It is versatile and relatively well supported on most browsers today. Among various capabilities of the Web Audio API, volume and panner manipulation is offered.

The challenge of managing the audio output is routing multiple audio sources that are constantly changing. Typically, we think of an audio processing flow from origin (source of audio) to output (speakers or headphones), however, the WFE Navigator's starting point of the audio branches is the output node (speaker), which does not change throughout a user's session. Hence, I had to construct the audio routes backwards, starting from the output node.

Figure 35 shows a configuration of audio components depicting 3 streams in view, where stream #1 and stream #2 have data at the current playback time, and no data for stream #3. When the WFE Navigator is initially loaded, the *Stream View* (Figure 21) is empty, with no streams in view. At this point, the WFE Navigator creates an Audio Context and initializes an audio output node in the Audio Context within the *App* component. When a stream is added to view from a query, a *Stream Manager* is instantiated with a panner node and a gain node. The *Stream Manager* connects the output of the gain node to the input of the panner node, and the output of the panner node to the input of the output node. The Gain-Panner-Output connection persists until the stream is removed from view.

If the stream has data at current playback time (either active state or inactive state), the *Stream Manager* passes the reference of the Media to the *Media Handler*. When the *Media Handler* is instantiated, an audio source node is created and its output is connected to input of the gain node in the *Stream Manager*, completing an audio route from source to output. In a transition from one media to another media in the stream, the *Media Handler* is destroyed during the transition where there is no data, along with the audio source node and its connection to gain node. When the next media is set to start, the *Media Handler* is re-instantiated with the new media, and a new audio source node is created and connected to the gain node in the *Stream Manager* again.

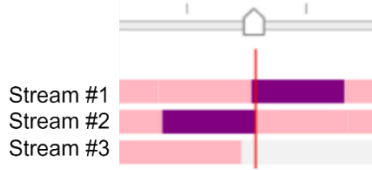


Figure 34. Timeline snippet for audio reference

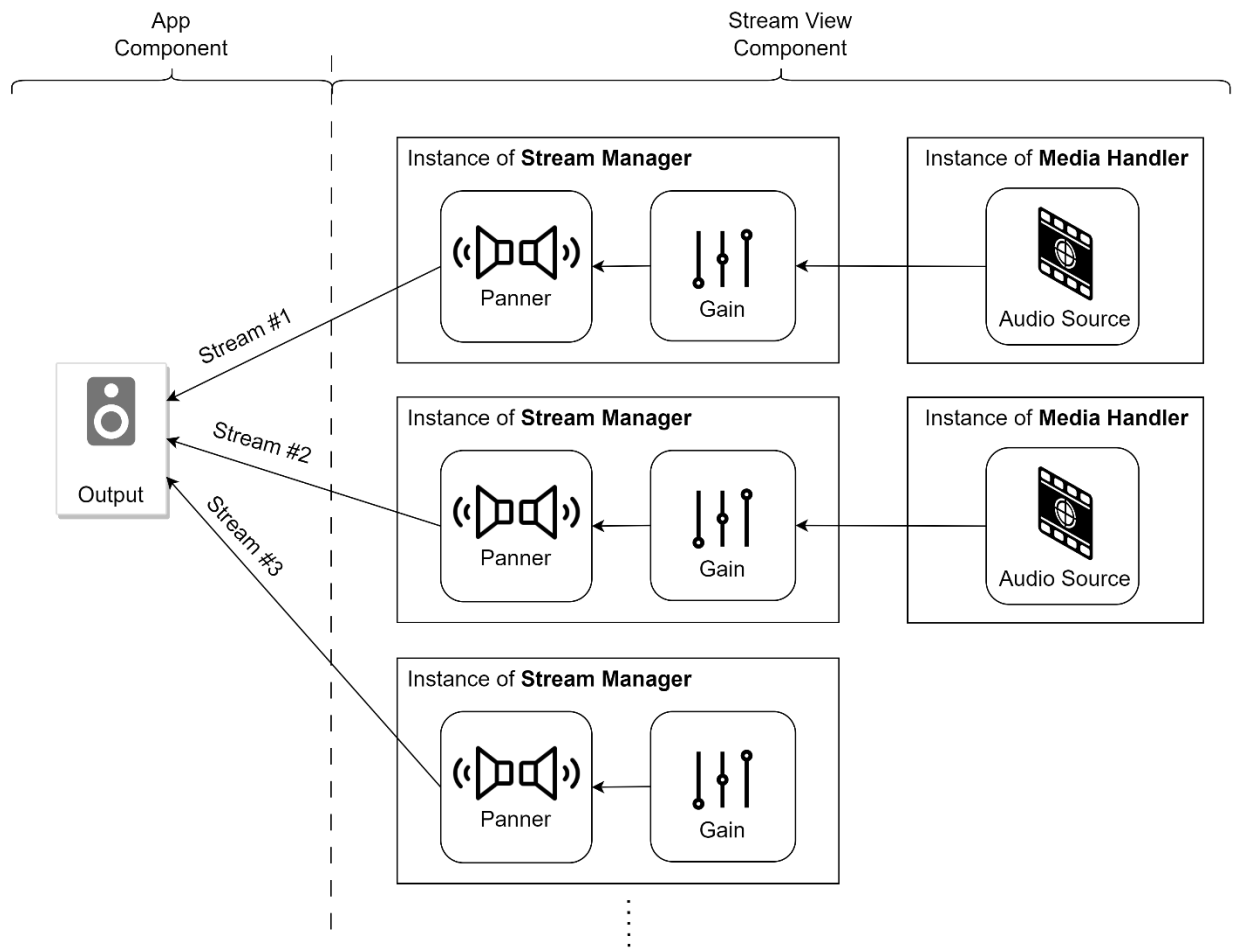


Figure 35. How audio goes through gain and panner controls before outputting to the output speaker, matching the state of Figure 34.

## 5.7 User Interface Considerations

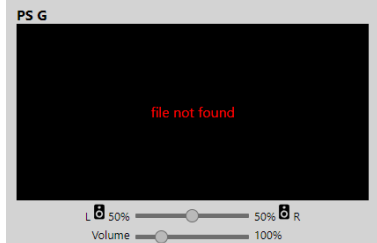
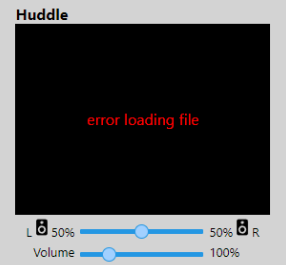
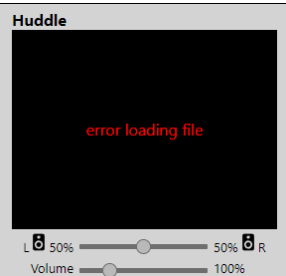
While the user interface was not a main factor in design decisions and considerations, some user interface elements added functionality to features we have.

### 5.7.1 What the Stream Manager Shows

Table 2 shows how the *Media Handler* and *Audio Controller* change what is shown based upon the user's viewing option. The option to hide a particular stream supersedes any other display options, whether there is media available for playback at time  $t$ , the file cannot be found, or file cannot be loaded. The controls for hiding or showing a stream is in the *Stream Controller* (see Figure 18 for reference). Hiding a stream overrules the display options for the accompanying *Audio Controller*.

Table 2. All possible Media Handler Interfaces.

Media hidden by user (A)	Has media at time $t$ (B)	Media muted by user (C)	Player interface
✓	superseded by (A)	superseded by (A)	<p>PS A</p> 
		superseded by (B)	<p>PS B</p> 
	✓		<p>PS C</p> 
	✓	✓	<p>PS B</p> 
	✓ but media file is not found in local storage		<p>PS F</p> 

Media hidden by user (A)	Has media at time t (B)	Media muted by user (C)	Player interface
	✓ but media file is not found in local storage	✓	
	✓ but there is error loading file		
	✓ but there is error loading file	✓	

There are 3 possible interface modalities for *Audio Controllers*: showing no *Audio Controller*, showing an interactable *Audio Controller*, and showing a disabled *Audio Controller*. The *Audio Controller* will always be superseded in cases when the user has chosen to hide that stream, or when there is no data for playback at that point in time. In any other cases, the *Audio Controller* display is governed by the user's option to mute streams, located adjacent to the hidden media option in the *Stream Controller*.

Figure 36 and Figure 37 show the enabled and disabled state of the *Audio Controller*, respectively. When the *Audio Controller* is disabled, users will not be able to make changes to the volume or panner values. The volume and panner values will be preserved and will persist throughout the stream across the media sequence.



Figure 36. *Audio Controller* in enabled state.



Figure 37. *Audio Controller* in disabled state.

## 5.7.2 Usability

Some usability factors were added in consideration to prevent user errors along the development of WFE Navigator. The WFE Navigator enables buttons when the application is in a state where the button's action can be performed. The buttons are disabled when the action cannot be performed. This concept is applied to start playback button, stop playback button, add matched streams button, and remove matched streams button.

Figure 38 shows the state of playback buttons when playback is stopped. Figure 39 shows the state of playback buttons when playing is active. During playback, the playback speed multiplier text box is disabled to prevent users from editing the speed multiplier mid-playback; this ensures that every stream is running with the same speed multiplier. The playback speed multiplier's value range is also constrained to accept non-zero positive numbers, to make sure the WFE Navigator is playing the streams forward (not rewinding). The playback speed multiplier is supported up to 16x the normal speed. This supported range is based on the capabilities of the media player library.



Figure 38. Playback Controller in inactive state.

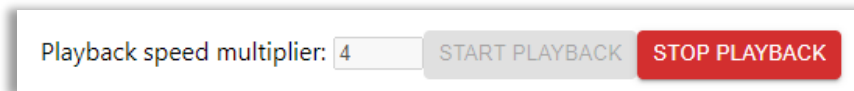


Figure 39. Playback Controller during ongoing playback.

The 'QUERY DATABASE' button in the *Query Controller* is always enabled for users to filter streams and retrieve the stream data. The message below the query button tells the user how many streams in the database match the user's query. If there are 1 or more matched streams, both the add and remove matched streams buttons are enabled, as shown in Figure 40. Figure 41 shows the add and remove

matched streams buttons disabled, which happens when the user’s query does not find any match, or when the WFE Navigator is first loaded.

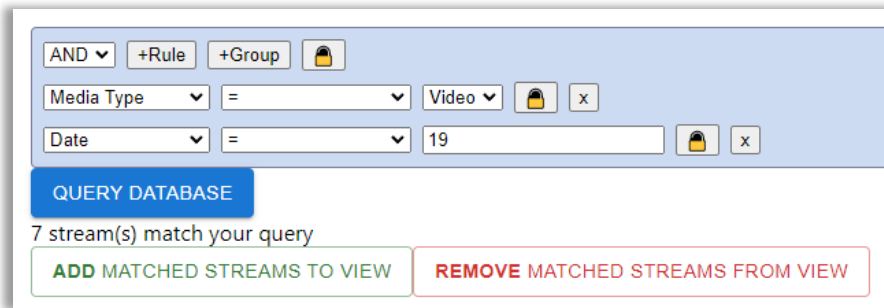


Figure 40. Query Controller when there are matched queries.

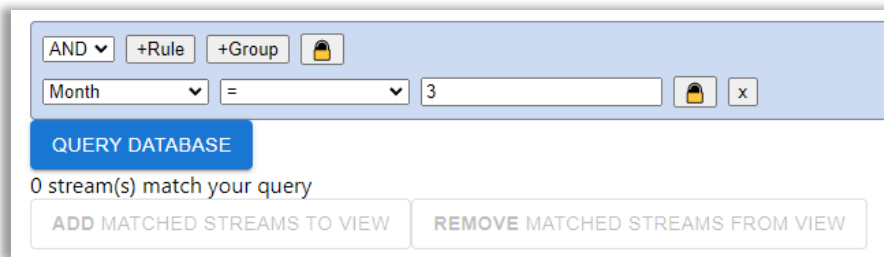


Figure 41. Query Controller when there is no matched query.

## 6 Evolution of Concepts

The WFE Navigator was designed to realize the abstraction of the concept of streams. The design of the WFE Navigator follows and adapts to the definition of streams throughout the development process.

### 6.1 Definition of Stream

Socha et al. identified the concept of collaborating streams early in the WFE research. A stream is a “time-based sequence of data from one particular data source” (Socha, Adams, et al., 2016), which are assembled with either captured moments in time, captured portions of time or a combination of both. A stream abstraction is “a continuous sequence of data [that] hides the implementation details of files storing data”. This definition allows us to interact with various collections of data as if they are streams.

Socha et al. defined collaboration of streams as “when events or segments map to the same time, or to the same concept”, where a concept is associated with a tag of various property types (e.g. task, type of activity, topic, day in a week). Through the earlier stages of the WFE Navigator’s development, the tool follows this definition of streams closely, but the following revelations from my work suggest that we revisit the question of what a stream is:

- The concept of stream is not represented anywhere other than within the directory structure.



- The sets of streams derived from the directory structure is one of the many ways to assemble streams.

In the BeamCoffer, Socha organized the files of the dataset based upon the day of capture, location, and equipment, as illustrated in Figure 2. All files from the same day of capture, location, and piece of equipment were placed in the same folder. Wei Wang's tool (Wang et al., 2016), traversed this directory structure, recorded each file's metadata to the BeamCoffer metadata database as a separate entry. The database entries did not explicitly contain any stream relationships. Instead, this WFE Navigator was designed to view all media files that had the same location-equipment-day as one stream.

This approach can result in unexpected behavior, such as Figure 42. Streams tagged with PS A and Video file type over 3 separate days which shows a stream view when a user adds streams from location *PS A* of type video. The resulting stream view shows 3 separate streams that share the same location and media type. Should these be three separate streams, or a single stream that spans three days? The definition of collaborating streams suggests that both answers could co-exist: a new *PS A* stream could be created with these three streams as sub-streams. The tool does not currently support that ability.

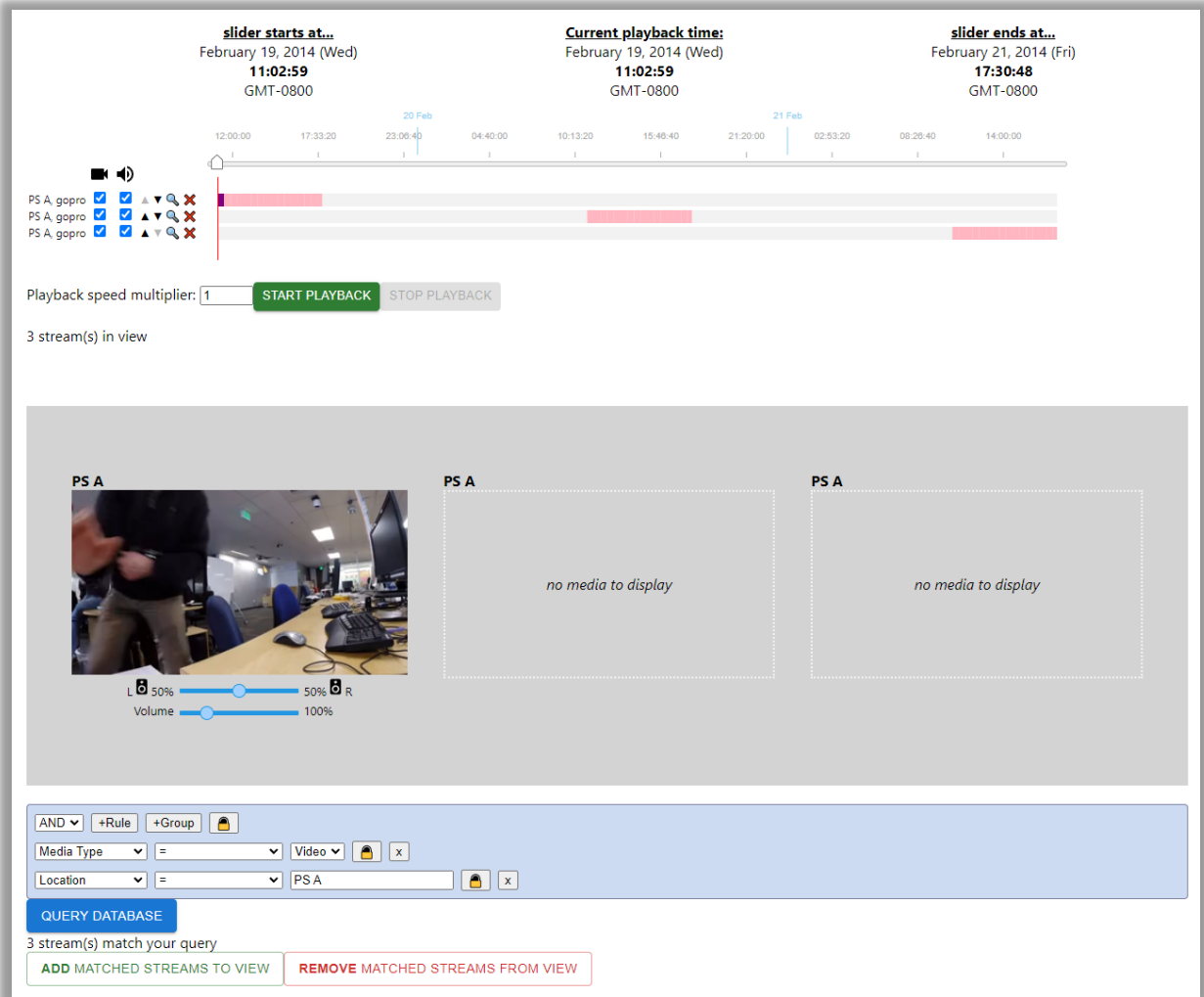


Figure 42. Streams tagged with PS A and Video file type over 3 separate days

To simplify the WFE Navigator, we augmented the original BeamCoffer metadata database with a Stream table and an associative table between Stream and Media. In the future, it would be good to modify Wei Wang’s tool to automatically do that.

The Stream table and a Stream-Media associative table could be used to support further assembly of streams. What media to include in a particular stream depends on what the researchers want to study and analyze. For instance, a stream may be a set of recordings that follows one study subject throughout the day, which can cross over multiple equipment and locations.

I’m proposing the following additional elaboration to the concept of streams:

- A single media file may be part of multiple streams.

An open question is whether the media files in a stream may overlap with respect to collection date. It may be that some media files may have parts that overlap, but the researcher must define which media file always take precedence, so that the resulting stream conforms to the “one particular source” (Socha, Jornet, et al., 2016) definition.

## 7 Conclusion

The WFE Navigator is a unique solution for a unique problem created by the emergence of datasets for WFE research. The WFE Navigator is currently the best navigational tool for researchers to browse WFE datasets. It automates the cross referencing of dataset to its metadata, eliminating the need to handle file sequences at the file level, the need to manually find elapsed time for playback, the need to juggle between multiple media player controls and many more. I believe the most significant value added to WFE dataset analysis is the removal of human errors from the navigational process altogether.

The requirements for the WFE Navigator are quite straightforward, but the implementation was not as simple. There are some limitations due to supporting tools not being made to cater to more than one target media, but the resulting application has fulfilled our needs and expectations, as well as inspiring additional features that can further enhance the WFE Navigator.

This work on WFE Navigator has contributed to the nuance and evolution of the concept of streams. Part of this work is to understand and apply the concept of streams. Design decisions for the WFE Navigator were grounded by the original definition of streams. However, as the WFE Navigator took shape, it uncovered rifts in our concept of streams that was previously limited by our organization of the dataset (that was a solution to another data organization problem in the first place). It confirms that Socha’s interpretation of streams is accurate but brings to attention the issue that it may not be precise enough.

The WFE Navigator is a great leap in WFE dataset handling and could not have been done to this extent without previous contributions from Soto Ogo and Wei Wang, both in analysis and implementation. The WFE Navigator has a lot of room to grow, and insights gained through this work will be helpful in informing further development and design decisions to improve upon it.

### 7.1 Constraints and Limitations

The current iteration of WFE Navigator was developed while considering that most data set users will not be software developers. Therefore, we tried to make user configurable features as easily accessible as possible and well communicated throughout our user interface.

A major constraint of the WFE Navigator is that it is not a standalone tool. The WFE Navigator must be used in conjunction with other tools that were developed for BeamCoffer-like datasets. The WFE Navigator consumes data in a specific format to function. These tools currently include the metadata database and the python script used to generate it. The python script goes through each collected data file and extracts relevant details to generate the metadata database. This database and its data must exist for the WFE Navigator to visualize any form of data.

Another constraint of the current iteration of the WFE Navigator is how it is scoped to cover only the navigational aspect of the dataset. This means that users are not able to add, edit, or delete any metadata of existing streams and media files. If any user finds an error or discrepancy in the metadata, the user is unable to make changes to the database and consequently the visualization of that data. The user has to utilize other tools to access and make changes to the database separately.

Currently, the WFE Navigator does not have any feature to support real-time collaboration. The user must have the WFE Navigator executable, the metadata database, and the media files stored on the same machine. In order to share an application view with another user, the other user must also have the executable, the metadata database and the media files stored and run locally on their machine. The user then has to query for the desired streams and manually add them into view.

The size of the dataset and streams may limit finer navigation. If the user's application view includes a set of streams that span a large amount of time, then accurately browsing through the streams will be more difficult as the streams in view will be proportionally contained within the width of the timeline. For instance, if the timeline width is 100 pixels wide, then moving the slider knob for 25 pixels would cover 15 minutes of an hour-long stream and 2 hours of an eight-hour stream.

## 7.2 Future Work

While this iteration of the WFE Navigator has resolved some major pain points, namely as the lack of synchronization of playback and streams, there are still many more features we would have liked to include that would enhance the utility of the navigator.

Currently, the WFE Navigator can display 2 types of data sources as streams: video and audio files. Future work to expand the WFE Navigator's compatibility is adding support for various type of temporal data sources, such as photo streams, health monitoring data (i.e. heart rate, body temperature), or weather data. The display of each data type would be customized to the type of visualization that conveys that data type best. There may be many ways to visualize a data type, and the decision is best discussed with the users of that dataset.

Incorporating additional or external navigational tools or systems may help improve limitations on fine grain browsing of streams. One such tool could be the Swifter (Matejka et al., 2013). The Swifter is a video navigational tool that utilizes key frames to help with scrubbing the video timeline.

Another domain to improve upon the current WFE Navigator would be the ability to add, edit and delete data from the metadata database. Adding these capabilities would open up the application to features such as adding annotations and tags to streams and generating new streams.

The WFE Navigator displays a sequence of multiple files as a singular continuous data source. A useful addition would be to generate a data source that is actually a singular continuous source, so that users can view this data outside of the WFE Navigator. This would mean adding video rendering capabilities to the WFE Navigator to allow the application to stitch multiple video or audio files and export them as one single file. This can also be expanded to include configurable export settings, such as target resolution, file type, or data quality.

The WFE Navigator is not quite deployable to non-software developer users as of right now since there are some setups specific to each operating system. Currently, Asher Dang, an undergraduate student, is continuing this work to create executables for Windows operating systems and Mac operating systems, as well as adding features to enrich the WFE Navigator. User validation testing would be more feasible after this adjustment as we would be able to test the WFE Navigator starting from launching the application to other specific viewing use cases to our target users.

## 8 Bibliography

- Martin, R. C. (2003). *Agile Software Development, Principles, Patterns, and Practices*. Pearson Education.
- Matejka, J., Grossman, T., & Fitzmaurice, G. (2013). *Swifter: Improved Online Video Scrubbing*.  
[www.hulu.com](http://www.hulu.com)
- Ogo, S., & Socha, D. (2019). *A System to Support Qualitative Research on Large Multimedia Data Sets*.
- Socha, D. (2015). *BeamCoffer Dataset: Professional Software Developers Collaborating in the Wild*.  
<https://sites.uw.edu/socha/beamcoffer-dataset/>
- Socha, D., Adams, R., Franznick, K., Roth, W. M., Sullivan, K., Tenenberg, J., & Walter, S. (2016). Wide-field ethnography: Studying software engineering in 2025 and beyond. *Proceedings - International Conference on Software Engineering*, 797–802. <https://doi.org/10.1145/2889160.2889214>
- Socha, D., Jornet, A., & Adams, R. (2016). *Wide Field Ethnography and Exploratory Analysis of Large Ethnographic Datasets*.
- Socha, D., & Sutanto, K. (2015). *The “Pair” as a Problematic Unit of Analysis for Pair Programming*.
- Wang, W., Socha, D., Olson, C., & Lagesse, B. (2016). *Building Cloud Based Software Systems To Support Collaborative Analysis Of Large, Multi-stream, Multi-modal Datasets of Physical-Cyber-Social Systems*.