ABSTRACT

With the distributed and rapidly increasing volume of data and expeditious development of modern web browsers, web browsers have become a possible legitimate vehicle for remote interactive multimedia presentation and collaboration, especially for geographically dispersed teams. To our knowledge, although there are a large number of applications developed for these purposes, there are some drawbacks in prior work including the lack of interactive controls of presentation flows, general-purpose collaboration support on multimedia, and efficient and precise replay of presentations.

To fill the research gaps in prior work, in this dissertation, we propose a web-based multimedia collaborative presentation document system, which models a presentation as media resources together with a stream of media events, attached to associated media objects. It represents presentation flows and collaboration actions in events, implements temporal and spatial scheduling on multimedia objects, and supports real-time interactive control of the predefined schedules. As all events are represented by simple messages with an object-prioritized approach, our platform can also support fine-grained precise replay of presentations. Hundreds of kilobytes could be enough to store the events in a collaborative presentation session for accurate replays, compared with hundreds of megabytes in screen recording tools with a pixel-based replay mechanism.

A WEB-BASED COLLABORATIVE MULTIMEDIA PRESENTATION DOCUMENT SYSTEM

by

Chunxu Tang

B.S., Xiamen University, 2013

M.S., Syracuse University, 2015

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering.

Syracuse University

June 2019

To my parents.

ACKNOWLEDGMENTS

Owning our story and loving ourselves through that process is the bravest thing that we will ever do.

— Brené Brown, *The Gifts of Imperfection*

The work presented in this thesis could not have been created without the encouragement and guidance provided by many others. I would like to acknowledge those who have helped me throughout this effort.

First and foremost, I would like to thank my advisor, Dr. C.Y. Roger Chen, who has been my mentor in my Ph.D. life. I am really grateful that he decided to accept a random master student to his group, who just walked into his office to ask to work with him on a sunny day. He has not only taught me how to accomplish excellent research work, but also the perseverance and courage in overcoming difficulties. Without his help, I could not even start my research, nonetheless struggling through the doctoral program.

There are many other faculty members helped me a lot in the past few years. Dr. Fawcett introduced me to the world of programming through his hard coursework and showed me his passion for cultivating young minds. Dr. Yu, who I have worked with in several courses, provided lots of advice on teaching and research. I would like

to thank the rest of my dissertation committee: Dr. Sponsler, Dr. Qiu, and Dr. Du for their insightful advice on my research work.

My labmates and many other friends also helped me a lot to make the long Ph.D. life more enjoyable. In particular, Beinan, Maria, Elie, and Eyoel made the cold and snowy Syracuse winters easier to bear. Ao, Yiou, Zhiruo, Qinyun, Zhi, Zilong, and Zhiyuan encouraged me in the most difficult and depressing part of the journey.

Last but not least, I would like to acknowledge my mother Fengyun Liu and my father Jun Tang for all of their support and encouragement in the past four years through my doctoral life. They told me the meanings of human lives and the essential courage to purse for the meanings. Without their patience and help, I could not make it through. This is dedicated to them.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Multimedia Systems

Multimedia systems provide storage for various types of multimedia, organize the media components, and distribute presentation services to end users. Because of the popularity of multimedia, lots of researchers have studied and implemented a variety of multimedia platforms. When a multimedia system is created for presentations, interactiveness plays a crucial role as it emphasizes users' flexible controls on presentations. The interactiveness here contains preparation of multimedia materials and presentation flows, interactive controls of multimedia during presentations, as well as the replay of presentations.

Preparation of presentations requires not only the storage of various multimedia objects but also the modeling of temporal and spatial relations among the objects. Some of previous work, including [83], [73], and [158], makes valuable contribution to creating graphical representations for temporal modeling. For example, in [83], Little and Ghafoor proposed an Object Composition Petri Nets (OCPN) model to describe the temporal relations among data objects involved in the multimedia scenario. A crucial shortcoming of prior work in this domain is the lack of interactive temporal control in presentation sessions. This implies difficulty in supporting the dynamic updates of the graphical representations, along with the accurate interactive replay

of operations on the multimedia objects after a presentation. For example, a presenter may insert a new media object in a session, which is not arranged on the temporal model beforehand. Wahl et al. [158] improved the real-time temporal control by introducing speed interaction, although their work still does not support dynamic updates of structures of the models. Regarding multimedia spatial relations modeling, there is also a similar issue of lacking interactive controls in previous work, such as [61], [67], and [143].

Besides the lack of interactive modeling, another gap in prior work is the absence of rich functionalities in multimedia presentations. For a specific presentation, the presenter may want to prepare the multimedia materials together with controls beforehand. A graphical tool is necessary here to provide editing mechanisms for these materials and events. In a presentation, the user may present not only the materials planned but also multimedia objects loaded from external resources. Additionally, the predefined temporal and spatial models can also be updated interactively at this step. Multiple multimedia objects such as videos, images and web pages are arranged on different panels and a user can interact with these objects by firing multimedia events in real time. Moreover, all the events here should be recorded in simple messages for replay purpose, such that users can review the whole presentation session accurately and efficiently afterward. A typical scenario is that students replay class lecture contents to prepare for exams. Some previous multimedia systems, such as [67] and [59] concentrate on the multimedia modeling and synchronization without real-time editing and interactive controls. For the replay mechanism, some prior work can achieve that with the help of screen recording tools, usually through recording

all actions occurring in a presentation as a video. This requires more space to store the videos and more bandwidth usage for distribution of the artifacts. In summary, we find that no prior work can provide the rich functionalities for multimedia presentation mentioned above.

1.2 Real-Time Collaboration

Real-Time Collaboration (RTC) has a long history in both research and engineering domains. For geographically dispersed teams, it is usually essential for teammates to have a remote collaboration tool for conferencing and presentation purposes. Additionally, considering the distributed and rapidly increasing volume of data and expeditious development of modern web browsers, it is necessary to provide an organized web-based group-aware platform to support collaboration among a large number of users. In the platform, not only are the artifacts of multimedia shared, but the controls on the artifacts are broadcast and synchronized.

Some of the pioneering work in collaborative multimedia systems covers GroupSketch [50], VideoWhiteboard [139], and Liveboard [31], where users' drawings are captured by cameras and projected to remote screens. Based on this technology, different screen sharing products such as Microsoft Skype [92], Cisco WebEx [162], and Google Hangouts [47] were developed and are now widely used. Recently, with the rapid development of modern web technologies, lots of web-based collaboration tools, including Collabode [44], RichReview++ [172], and Tele-Board [165] were developed.

To our knowledge, even though various web-based groupware tools have been developed for versatile purposes such as whiteboard drawing and document editing, no system can integrate these functionalities to suit general-purpose multimedia collaboration. Kim et al. [63] made a valuable contribution in this direction by proposing an MVC architecture for ubiquitous collaboration. Their tool still only handled static media like whiteboard drawings and images, without effectively working on other types of media with dynamic contents such as videos and web pages. It would be beneficial if there was a uniform collaboration framework that easily extended to support various types of media.

Furthermore, for current popular screen sharing products, in a specific session, both the contents of the media and manipulations on the media are transmitted by capturing the present display continuously, leading to large consumption of network bandwidth. However, we propose to split the contents of a presentation into a static and a dynamic part. The static media resources, for example, a video or a PDF document, can be transmitted to attendees of a collaboration session beforehand, while the dynamic events occurring in a session such as muting a video are broadcast and synchronized on the fly. In that case, a collaboration session is organized as the combination of static materials uploaded to the browser from the local file system or loaded from an external URI and dynamic events encapsulated in simple messages. With these messages, the precise recording and replay of these collaboration events can also be achieved, differentiating our work from traditional collaboration platforms. In our study, hundreds of kilobytes can be enough to store the events in a session, compared with hundreds of megabytes used in screen sharing tools. Our target is not

to substitute the current screen sharing tools, but to provide an efficient collaboration platform as a supplement to these applications.

1.3 Design Features and Contributions

To cover the research gaps in multimedia systems and real-time collaboration mentioned in the previous two sections, we propose a web-based collaborative multimedia presentation document system. Specifically, it has the following features:

1. **It realizes the temporal and spatial scheduling on multimedia objects in one graphical representation.** We create a new graphical representation, document net, based on activity diagrams and Petri nets, for both temporal and spatial modeling. Moreover, the nodes in a graph are related to the multimedia objects in the document system, so that object-oriented design, temporal modeling, and spatial modeling are integrated together. A graphical editor is also provided in our system where users can draw diagrams to prepare the models.

2. **It supports real-time interactive controls of temporal and spatial models.** Not only can the models be prepared beforehand but also they can be modified on the fly. When the models are updated, the presentation flows will be automatically adjusted. To achieve this, we present a rendering approach to convert a document net into an event timeline. The new modifications of the document net are reflected on associated positions on the timeline.

3. **It supports collaboration on multimedia.** Not only static media (PDF documents, images, etc.) but also dynamic media (videos, web pages, etc.),

which usually carries dynamic contents, are supported in our system. Additionally, considering the demands for presentations, we implement annotation tools for multimedia, consisting of highlighting, free drawing, and shapes insertion. This feature also indicates that our collaboration framework is open to extension, especially considering that different users may have distinct collaborative presentation purposes, and we cannot include all of them in our platform. A developer can add supports for collaboration on other kinds of media according to his/her specific demands.

4. **It is event-driven and message-based.** This indicates that all actions that have occurred in a presentation are captured as events. Thus, a presentation consists of a stream of multimedia events. These events are linked to the nodes in a document net for preparation, recording, and replay. Furthermore, all events can be prepared and represented as specially designed simple messages. With the help of these messages, our system can provide very efficient, accurate, and event-based replay functionality, which differentiates our work from previous work with a pixel-based replay approach. Moreover, because of the simplicity of messages, the media files can be transmitted beforehand. During the presentation, only minimal numbers of simple messages of events are sent and handled. This implies that our system has a very low bandwidth usage, compared with current video conferencing products such as Microsoft Skype and Google Hangouts.

5. **It is context-aware.** This feature indicates that most of the media controls are related to the objects underneath. Currently, to our knowledge, most of the web-based collaboration tools are position-based or proportion-based, which implies that media controls are synchronized through absolute or relative positions. By contrast, we propose an object-prioritized hybrid synchronization approach, which is more accurate and efficient.

6. **It is web-based.** Considering that the significant development of web technologies, web browsers have become a possible platform to serve multimedia presentation functionalities. The users can access our system from various platforms, including desktops, tablets, and mobile phones, as long as web browsers are supported. This significantly eliminates the complexity of setup efforts on different platforms.

Based on the research gaps in prior work and design features in our system, our contributions consist of:

1. A hybrid message-based event-driven multimedia model with the combination of the object-oriented modeling, temporal modeling, and spatial modeling, with representations by text files, timelines, and graphs, interchangeably.

2. Rich functionalities in multimedia presentations including preparation of temporal and spatial models, real-time interactive controls of these models, extraction and loading of external dynamic web resources, storage of presentations in simple messages, and precise replay of presentations.

3. Support of general-purpose multimedia collaboration, including videos, images, PDF documents, web pages, Google Maps, and external web elements, through a flexible distributed multimedia collaboration protocol and framework.

4. A message-based object-prioritized synchronization approach for accurate and efficient multimedia collaboration.

1.4 Organization of the Dissertation

This dissertation is organized into seven chapters. As an overview, Chapter 1 covers a general context of our work in multimedia systems and real-time collaboration. Chapter 2 demonstrates a survey of technologies which are used in our work. Chapter 3 presents a detailed description of our presentation document system, including the underlying models and specific implementations. Chapter 4 explains the approaches used in continuous updates of external web resources. Our distributed context-aware collaboration framework is discussed in Chapter 5, and the approaches used in collaborative web browsing is introduced in Chapter 6. Chapter 7 concludes the dissertation and discusses our future work. More details are provided below.

Chapter 2 surveys the web technologies used in our work. Specifically, the technologies consist of inner mechanisms in rendering web pages (Section 2.1), the concept of the event loop in JavaScript (Section 2.2), and Chrome extensions (Section 2.3).

Chapter 3 covers our interactive multimedia presentation document system from multiple perspectives. The chapter starts with the description of our hybrid model in Section 3.1. Section 3.2 demonstrates the object-oriented model, Section 3.3 depicts

the temporal modeling including our proposed document net structure, and Section 3.4 describes the spatial modeling. We explain the details of our document system implementations in Section 3.5. Section 3.6 provides details of related work, and Section 3.7 concludes the chapter.

Chapter 4 discusses and compares various approaches for continuous updates of external web resources. Section 4.1 introduces the background of traditional approaches in continuous updates of external web elements. Section 4.2 presents the challenges in implementing these continuous updates. Afterward, two different approaches: continuous updates with a server and without a server are evaluated and compared in Section 4.3 and 4.4. Section 4.5 presents two methods to load external web resources on our application. Section 4.6 provides details of related work, and Section 4.7 concludes the chapter.

Chapter 5 concentrates on the description of a distributed context-aware collaboration framework. The framework is discussed from aspects of challenges in this domain in Section 5.1, framework methodologies in Section 5.2, and architecture components in Section 5.3. We evaluate the collaboration system and show the results in Section 5.4. Section 5.5 discusses the related work in real-time collaboration, and Section 5.6 concludes the chapter.

Chapter 6 presents the methods used to achieve collaborative web browsing. Section 6.1 and Section 6.2 introduce the background and challenges in collaborative web browsing. Section 6.3 emphasizes on the approach to synchronize browsing actions through a Chrome extension. Section 6.4 discusses the related work, and Section 6.5 gives a summary of this chapter.

Finally, we conclude the dissertation in Chapter 7. We also give some interesting suggestion to extend our work for future research.

2. SURVEY OF TECHNOLOGIES

2.1 Rendering of Web Pages

Web browsers are widely used in our daily lives to browse different web pages. As our system is based on web browsers, it is necessary to understand the inner mechanism of web browsers. A typical reference architecture of web browsers is shown in Figure 2.1. Here, the major components are

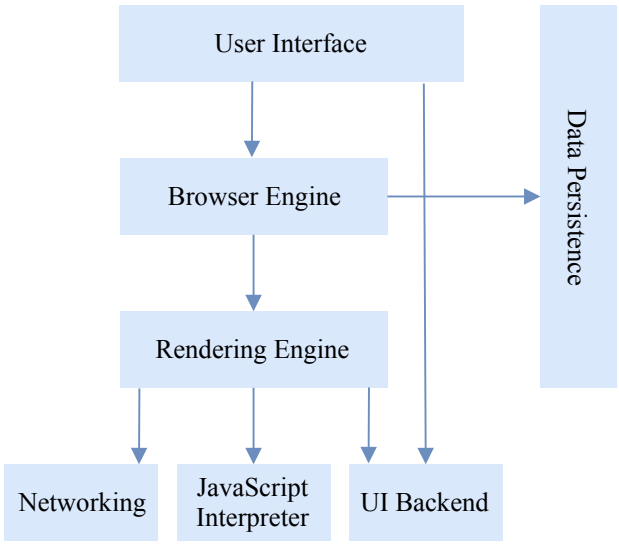


Fig. 2.1.: Reference architecture for web browsers.

- **User Interface**, representing UI components such as the address bar of a web browser.
- **Browser Engine**, receiving instructions from the User Interface and manipulating the Rendering Engine.

- **Rendering Engine**, rendering contents of web pages according to HTML layouts and CSS styles.
- **Networking**, sending/receiving network messages.
- **JavaScript Interpreter**, interpreting and executing JavaScript code.
- **UI Backend**, drawing basic shapes and windows.
- **Data Persistence**, storing various types of data in different forms of storage, such as cookie and local storage.

Our document system significantly utilizes the mechanisms in the rendering engine to monitor various events. An example of rendering a web page in the WebKit [163] is shown in Figure 2.2. WebKit is an open-sourced browser engine used by Safari. Moreover, Google Chrome's engine, Blink [48], is forked from and developed based on the WebKit. In the rendering procedure, HTML contents are converted into a DOM (Document Object Model), and style sheets are parsed to create a CSSOM (CSS Object Model). These two steps are executed in parallel. And after the two models are established, they are merged to form a render tree, which is used to render the display.

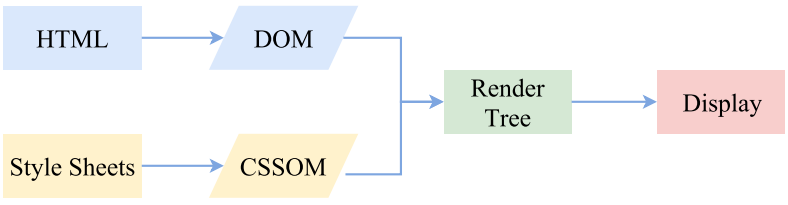


Fig. 2.2.: Rendering of a webpage in WebKit.

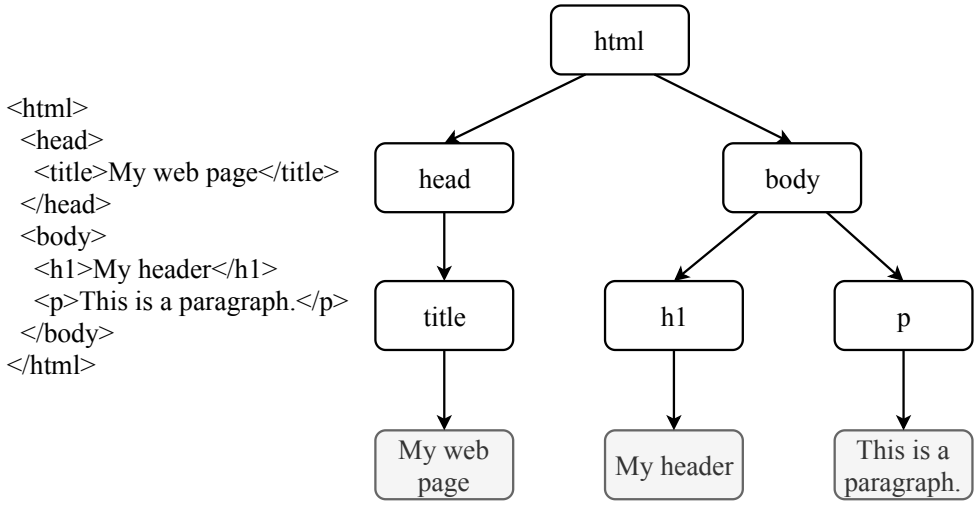


Fig. 2.3.: An example of DOM.

A DOM is a tree structure where each node contains an object, representing one part of the web page. An example of DOM is illustrated in Figure 2.3. The left part of the figure is the HTML of a web page. The generated DOM is shown on the right. Here, every node depicts a tag in the HTML, and each link represents a parent-child relationship between two nodes. Additionally, event handlers can be attached to the nodes. And when a specific event occurs, the registered handlers will be invoked. For example, a click event handler can be registered to monitor the *click* event of a button.

2.2 Event Loop in JavaScript

The Event Loop in JavaScript is the cornerstone for concurrency models. JavaScript code runs single-threaded, so only one task can be executed at one time. This limitation significantly reduces the programming difficulty in concurrent programming but causes concern of performance. To solve the problem, the event loop model, shown in

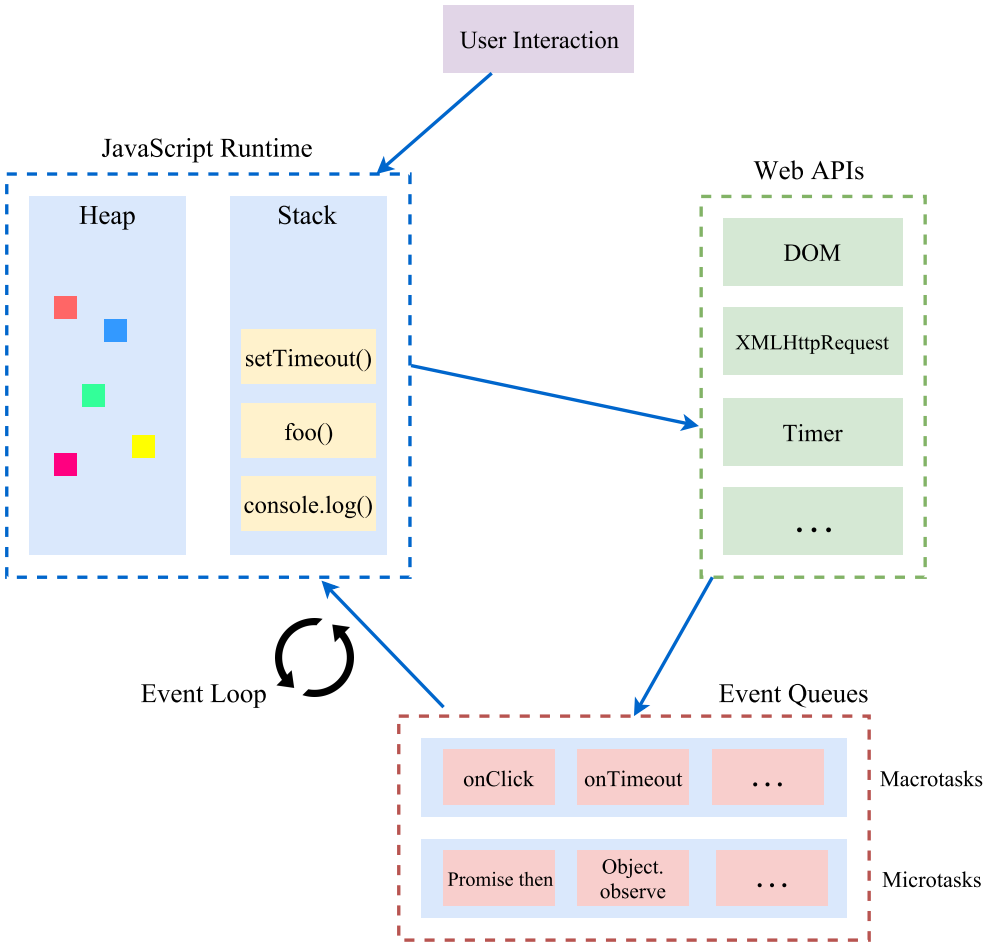


Fig. 2.4.: The event loop in the execution of JavaScript code.

Figure 2.4, is introduced to handle incoming events. There are two memory models in the JavaScript runtime environment: heap and stack. A heap contains unstructured regions of memory. Objects are usually allocated in a heap. By contrast, a stack employs a last-in, first-out queue structure. Function calls form a stack of frames. Additionally, the web browser keeps track of events that have taken place but have not been processed in the event queues. Every time, the browser checks the head of a specific event queue to see whether there is an event to process. If yes, the associated event handler will be executed, and the event is popped from the queue. The tasks

are categorized into two types: macrotasks and microtasks [118]. Macrotasks contains the tasks called, usually generated by document objects, such as the handler of a *click* event. On the other side, microtasks include some smaller tasks which need to be executed as soon as possible such as events related to the Promise. This mechanism allows the tasks executed before the UI is updated.

Based on the event loop model, take a *click* event on a button element as an example. First of all, an *addEventListener()* function is pushed onto the JavaScript runtime stack and executed to register a handler to monitor the *click* event. Afterward, when the associated button is clicked, the event is captured as a DOM event, and the callback function is pushed into an event queue, specifically, a macrotask queue. From the queue, the callback function is executed and used-defined mechanisms are realized.

2.3 Chrome Extensions

The Chrome browser provides extensions to control browser functionalities and behaviors such as injecting JavaScript code in target web pages, adding items in the context menu, modifying the theme of a new tab, etc.

A Chrome extension consists of HTML, JavaScript, and CSS. It starts with a manifest, which contains the basic information of the extension, such as the name and version, as well as permissions and external URLs. An extension follows the least privileges and privilege separation principles [115]. Besides a manifest, an extension may also contain a background script, UI elements, a content script, and

```
{
  "name": "Extension Example",
  "version": "1.0",
  "description": "My example of a Chrome extension.",
  "permissions": ["storage"],
  "manifest_version": 2
}
```

Fig. 2.5.: An example of a manifest.

an options page. The communication between scripts is achieved by synchronous or asynchronous Chrome message passing APIs.

- Background script. A background script usually registers event handlers for browser events. For example, it can listen to selections on the context menu of Chrome.

- UI elements. This part contains a popup web page with related JavaScript files. These files formulate the UI and actions of the extension.

- Content script. A content script is isolated from core extensions, and it injects JavaScript code to execute when the original web page is loaded in the browser. It can read and modify contents from various websites.

- Options page. The options page adds customizations to the extension.

3. INTERACTIVE MULTIMEDIA PRESENTATION DOCUMENT SYSTEM

3.1 Model Structure

We propose a hybrid model which contains three modes for the document system: text-based messages, event timelines, and complex models. The three representations can be converted mutually.

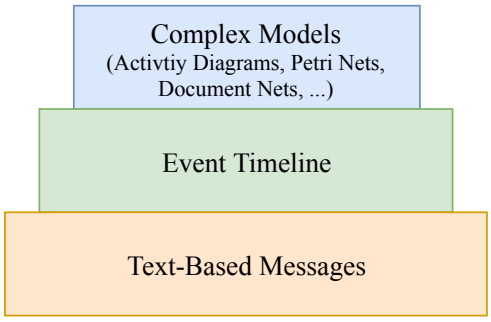


Fig. 3.1.: The structure of the hybrid model.

We design simple messages to depict the events in a specific presentation. And the messages can be converted to an event timeline, where all events are arranged based on timestamps. Furthermore, the event timeline can be transformed into a complex model such as activity diagrams and Petri nets. Note that our model is not restricted to a specific complex model. The major requirement for the complex model is that it can be converted to an event timeline mutually.

3.2 Object-Oriented Design

3.2.1 Introduction

Object-oriented design is widely used in the implementation of multimedia systems. A multimedia object can be exploited to represent a type of media, including specific media materials and related media controls. As our document system concentrates on functionalities in presentations, we also define objects related to presentations.

3.2.2 Document Object

Definition 3.1 *(Document Object) A Document Object (DocObj) is defined as the basic multimedia unit on the document system, and it represents functionalities of a specific type of multimedia.*

A document object has the following features:

- The base *DocObj* class can be inherited to encapsulate functionalities of a type of multimedia. For example, to represent videos, we create a *VideoObj* class which inherits from the base class.

- The controls on the multimedia are represented by functions in an object. For example, the *play* action of a video is represented by a *play()* function in a video object.

- The state of a document object is recorded in data members. For example, the current time of a video is recorded in the *current_time* variable in a video object.

Additionally, we categorize document objects into *static* or *dynamic*. Static objects are objects whose contents will not change after they are loaded to the document system. For example, a PDF document is a static object. Dynamic objects, by contrast, can change after the initial load to the system. For instance, we collect a live stock price table which continuously updates in response to the real stock market. For presentation purposes, we support essential annotations for both types.

From the perspective of object-oriented design, a document object is first represented by a base *DocObj* class, which contains the essential information of each media object.

- data members
  - **source**, the source, URI for instance, of the document object. A source is classified as local or remote. For a local source, the document object's target media file is uploaded and stored in the local web browser. For a remote source, the material is accessed from an external URI.
  - **obj_id**, the unique identifier of the document object.
  - **obj_type**, the type of media the object represents.
  - **panel_id**, the unique identifier of the panel the object resides. The definition of a document panel is described in Section 3.2.4.

  - **height**, the height of the user interface of the object.
  - **width**, the width of the user interface of the object.
  - **type**, static or dynamic type of media.

- methods
  - **render()**, that renders the user interface of the media on the web page. This function needs to be overridden by a inherited class.
  - **setHeight()** and **getHeight()**, the mutator and accessor of height.
  - **setWidth()** and **getWidth()**, the mutator and accessor of width.

Furthermore, in the document system, as the objects rendered on the web page are usually media objects of classes inherited from the base class. As illustrated in Figure 3.2, each specific type of media needs to have a subclass of the DocObj class. And in each subclass, more functions and properties are introduced to represent specific details of each type of media.

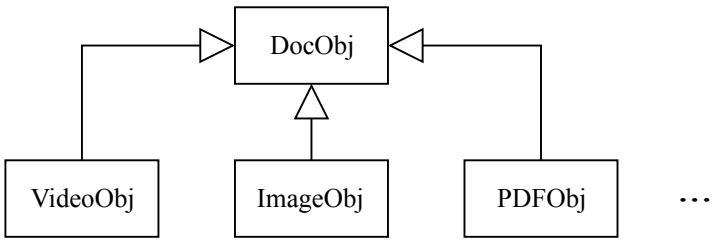


Fig. 3.2.: Classes of various document objects.

For example, in a *VideoObj* object, it also contains functions such as *play()*, *pause()*, and *updateVolume()*, and data members such as *current_time* and *playback_rate*.

3.2.3 Document Event

Definition 3.2 *(Document Event) A Document Event (DocEvent) is defined as an event that occurs on a document object, triggered by an object function.*

Based on the definition above, document events have the following attributes:

- Document events are highly related to document objects. A document event must be fired by a document object. For example, the *play* event of a video is triggered by the *play()* function in a *VideoObj* object. The *move* event of an image is triggered by the *move()* function in an *ImageObj* object.

- Every control of a media needs to be represented by a document event. In the presentation document system, the document events are the only way to capture the controls of a media.

- Similar to document objects, document events have a base class DocEvent. This class contains basic data members an event should contain. A possible multi-level inheritance is introduced in Figure 3.3. From the base DocEvent class, the next level contains the classes of various types of media. And the following level contains the specific event classes of each type of media.

3.2.4 Document Panel

Definition 3.3 *(Document Panel) A Document Panel (DocPanel) is defined as a container for a document object.*

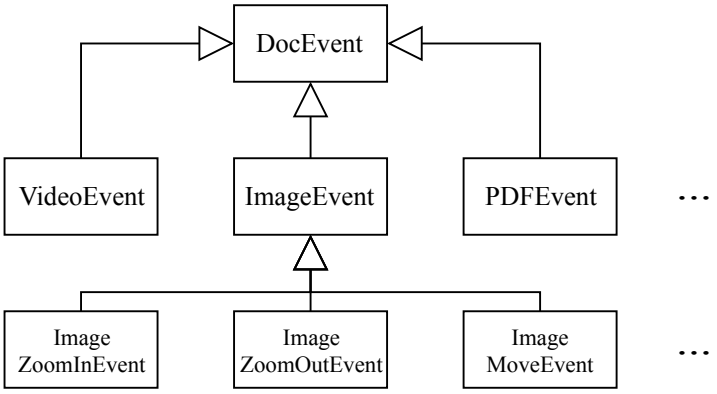


Fig. 3.3.: Classes of various document events.

The document panels are rendered on the document system web page, arranged in a specific spatial relation. Each panel contains at most one document object. From the perspective of object-oriented design, every panel includes

- **panel_id**, the unique identifier of the document panel.
- **curr_doc_obj**, the document object currently rendered on the panel. If there's no object rendered, *NULL* is used to fill the field.

3.2.5 Document Annotation

Definition 3.4 *(Document Annotation) A Document Annotation (DocAnnotation) is defined as an annotation attached to a document object, implemented in a layer-based structure.*

As each document annotation cannot exist without a related document object, the annotation can be stored as a data member in the object. Additionally, as we demonstrate above, the annotations are recorded in multi-layers. Specifically, we introduce three types of annotations: highlighting, free drawing, and shapes insertion.

Highlighting is a common demand in presentations. For example, a presenter may want to highlight some crucial text on a web page in a presentation session. By following the W3C Web Annotation Protocol [150], the structure of our highlighter is shown in Figure 3.4.

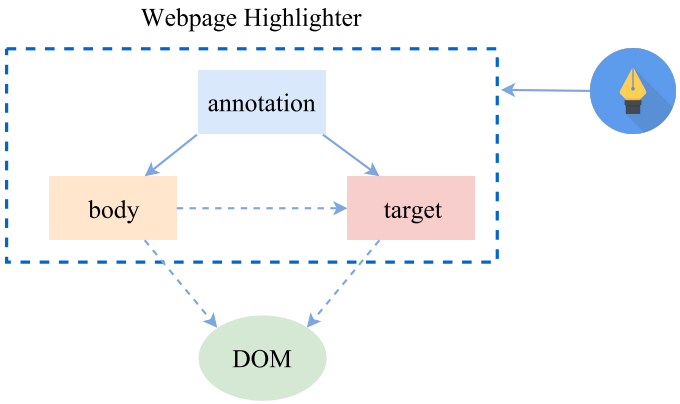


Fig. 3.4.: Web annotation model.

Here, for every annotation, it has at least two essential fields: *body* and *target*. They both refer to DOM elements. For highlighting text, the body is the new *span* node we create, and the target refers to the original text node, whose text is wrapped and highlighted by the new span node. While, there are also some other fields in every annotation, such as context (*highlighter-context* in our tool), unique identifier (assigned to every annotated object), and annotation type (text or image).

We capture every highlighting event by monitoring the sequence of *mousedown*, *mousemove* and *mouseup* events. After this sequence of events is fired, we fetch the range of the text selected via HTML5 Range API [99]. However, there is an issue about how to serialize the highlight to a string for storage and transmission. We

use an approach similar to texthighlighter [141]. In this method, the highlight is represented in a path from the root node in the DOM.

Take the annotation in Figure 3.5 as an example. After the text "o, wo" is highlighted, a new span is created to wrap the text as its child node. To find the path, we traverse from the original text node to its parent and obtain the index 0 because it is the only child of node p. Afterward, we visit node p and get the index 1 of it in the siblings. We repeat this procedure until arriving at the root node and the path created is [1, 1, 0]. Furthermore, for the replay, we just need to traverse from the root node and follow the path to find the target node. Because we also have the range of text highlighted via HTML5 Range APIs, we can easily replay this highlight.

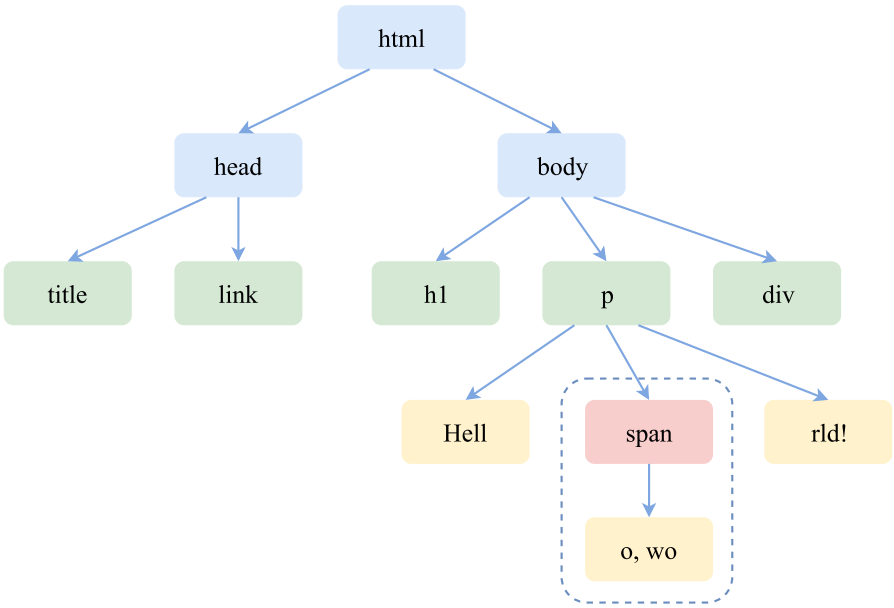


Fig. 3.5.: An example of highlighting a snippet of text.

Providing that the highlights are implemented on the text node, free drawing, as a basic requirement for web-based whiteboard tools, is realized on a topper canvas layer. The functionality is achieved with HTML Canvas APIs [98] by monitoring the series

of *mousedown*, *mousemove*, and *mouseup* events. Moreover, we exploit Fabric.js [35], a JavaScript HTML5 canvas library to allow users to draw various shapes on the canvas.

3.2.6 Document Presentation

Definition

Definition 3.5 *(Document Presentation) A Document Presentation (DocPresentation) is defined as the combination of a series of document events and related document objects, including temporal and spatial relations among the objects.*

According to the definition above, a document presentation can be modeled as a stream of document events that occurs in temporal order, with document objects representing the multimedia resources, as shown in Figure 3.6. A crucial design in our document system is that the placement of a document object onto a document panel is also modeled as a document event whose event type is the insertion of the target type of media.

Preparation

As we categorize the media materials into static or dynamic, for some static materials such as videos, images, and documents, these resources can be prepared beforehand, instead of downloading them on the fly. These multimedia materials are encapsulated into associated document objects. Meanwhile, we also allow inserting

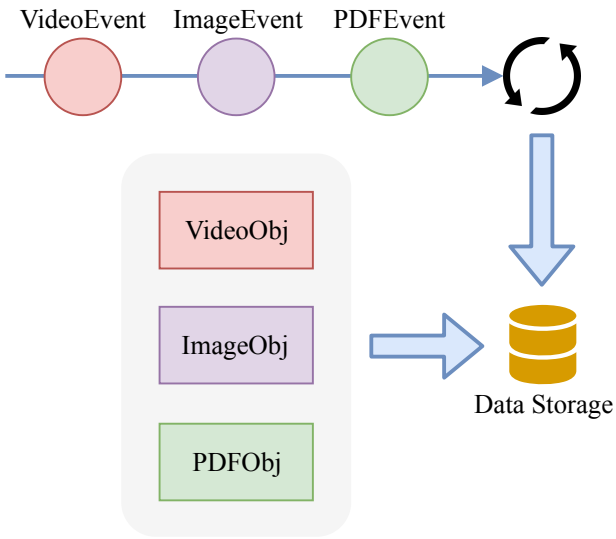


Fig. 3.6.: Data view of a document presentation.

a new multimedia material during a presentation via an external URI. These static or dynamic materials are also wrapped into document objects placed on document panels.

Besides the preparation of static materials, the events in the presentation can also be prepared via document events. Unlike some traditional presentation programs such as Microsoft PowerPoint [93] and Google Slides [46], which organize a presentation in pages, we concentrate on the multimedia events in a delivery. Consequently, we allow users to prepare events such as playing a video, zooming in an image, etc. The details of depicting the temporal and spatial relations are discussed in Section 3.3 and 3.4, respectively.

Replay

After attending a specific presentation, it is quite common that an attendee may want to replay the presentation. For example, after an online class, a student may plan to watch the class content again for a review. Previous work usually solves the problem by recording the whole presentation in a video. By contrast, because we model each presentation state as the execution of a stream of media events, replaying the presentation can be accomplished by re-executing all the events, which is much more efficient.

Persistence

For persistence in this model, the events are saved to the data storage sequentially. Document objects related to the events are also persisted into the data storage. The data storage may be on the browser side or server side, depending on specific demands.

3.3 Temporal Modeling

3.3.1 Introduction

Temporal modeling involves the description of temporal relations among multimedia objects, as well as temporal controls on these objects. In our work, to support a multimedia presentation document system, the system needs to fulfill the following requirements:

1. The complicated system features modeling.

- Sequence. Sequential execution might be the most common system feature in presentations. Here, all events are arranged in sequential order and executed one by one. For example, a presenter plays a video for a while and then pauses the video. This feature can also be utilized to model the causal relationship, namely happen-before relationship, among events.
- Concurrency. Concurrency implies a fork operation, causing the presentation flow to be transferred to multiple concurrent flows. In a presentation, an instance of applying concurrency is playing a video related to a specific page of a PDF document, when displaying the PDF material.
- Synchronization. Multiple presentation flows may be joined together, leading to a single flow afterward. Take the example mentioned when explaining concurrency, after playing the video for a while, the video is removed and the presentation re-concentrates on the PDF file.

2. Explicit notations for controls on multimedia presentation flows. Considering that several multimedia objects may be presented in a session, the graphical representation needs to have corresponding explicit notations for each flow, including start and termination. Additionally, a user may want to assign a positive real number as the interval between two events. This mechanism should also be depicted in the diagram.

3. Accurate execution (rendering) of temporal models. For temporal models prepared before presentations, they should be executed accurately according to the temporal relations interpreted. This is also called multimedia synchronization.

4. Dynamic updates of temporal models. To emphasize the interactive control of temporal relations, a presenter may update a temporal model during a presentation. The original temporal model should be adjusted automatically as a result.

5. Accurate presentation replay based on temporal models. All temporal events occurred in a presentation can be recorded and replayed afterward.

6. The models need to be linked with the objects described in the previous section. Especially, the document events should be reflected in the graphical representation.

3.3.2 Timeline

The timeline approach is the most basic temporal specification scheme. It arranges the events sequentially on a timeline, as illustrated in Figure 3.7. Some pioneering work, including [43], [58], and [28] applied basic or improved timeline diagrams for temporal modeling. For example, in [58], the authors proposed a timeline tree structure to model synchronous and asynchronous events.

Even though these diagrams are not good at modeling complicated system features, they are simple to be interpreted and executed at associated timestamps, as events are all organized on one timeline.

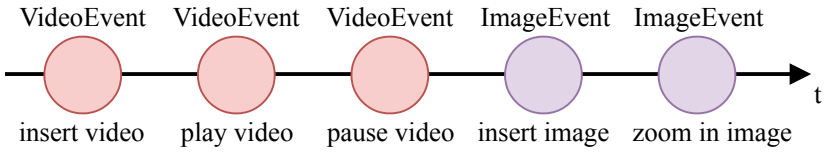


Fig. 3.7.: An example of applying a timeline for temporal modeling.

3.3.3 Sequence Diagram

Sequence diagrams organize objects based on a time sequence. The interactions among the objects are arranged in parallel vertical lines. Besides system modeling, currently, sequence diagrams are actively used in testing [72, 122, 137] and verification [30, 78, 174]. An example of applying a sequence diagram to our system is shown in Figure 3.8.

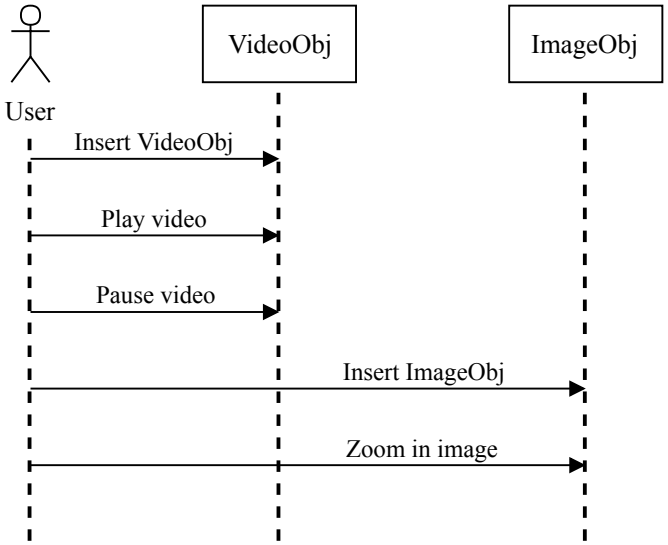


Fig. 3.8.: An example of applying a sequence diagram for temporal modeling.

Here, the interactions among three objects including *User*, *VideoObj*, and *ImageObj* are depicted. The user first inserts a video via a VideoObj onto the document

system, plays the video for a while and then pauses it. Afterward, an image is rendered and zoomed in.

Some prior work, including [123], [124], and [57], has applied sequence diagrams for temporal modeling of multimedia applications. For instance, in [124], Sauer and Engels proposed temporal modeling in extended sequence diagrams in either time point or time interval relations.

The sequence diagram provides a clear demonstration of the interactions between the user and multimedia, but it can be overwhelming to represent complicated system features such as concurrency and synchronization.

3.3.4 Activity Diagram

Activity diagrams are another widely used type of UML diagrams, primarily exploited to depict the flows of control and sequences of actions related to activities. The graphical representations consist of nodes and edges. The activity states are denoted with round-cornered boxes. The transitions are shown in arrows. Branches are shown in diamond boxes. Guard conditions, or decisions, are included in a branch, returning a boolean expression to signal whether the requirement is fulfilled or not. Forks and joins are also supported in an activity diagram. A fork node is represented by an arrow entering a vertical bar with multiple arrows leaving the bar. By contrast, a join node is exactly the opposite, with multiple arrows entering the synchronization bar and one arrow leaving the bar. Currently, activity diagrams are widely used in workflow modeling [6, 32, 42] and testing [20, 60, 82].

According to the study of Wang el at. [82], an activity diagram is formally defined as a 6-tuples $D = (A, T, F, C, a_I, a_F)$, where

1. $A = \{a_1, a_2, ..., a_m\}$ is a finite set of activity states;
2. $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions;
3. $C = \{c_1, c_2, ..., c_n\}$ is a finite set of guard conditions;
4. $F \subseteq (A \times T \times C) \cup (T \times C \times A)$ is the flow relationship between the activities and transitions;
5. $a_I \in A$ is the initial activity state, and $a_F \in A$ is the final activity state.

Here, we apply an activity diagram to model the same scenario depicted in Figure 3.7 and 3.8, shown in Figure 3.9. The solid circle on the very left of the diagram is the initial node a_I of the model, and the solid circle with a hollow circle inside on the right is the activity final node a_F. The actions are all represented in hollow circles. Note that it is difficult to model each duration between two actions in plain activity diagrams.

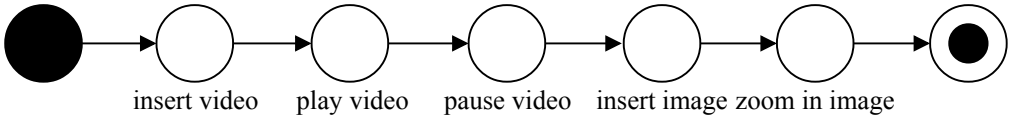


Fig. 3.9.: An example of applying an activity diagram for temporal modeling.

Activity diagrams are good at describing the dynamic behaviors of systems, while there are several issues when applying them to our document system:

- Activity diagrams lack the capability to model temporal relations among presentation media objects, especially the delays related to transitions. Guelfi and Mammar [53] made a valuable contribution of inventing timed activity diagrams by inserting timers into original activity diagrams. Because they do not expect every transition to have a timer, an additional timer block is necessary to control the events, making it complicated for end users to prepare presentations.

- In prior work, basic activity diagrams do not support dynamic updates of temporal models. The modifications of presentation flows may cause significant updates of the original model, causing difficulty in multimedia synchronization.

- Activity diagrams cannot differentiate the event of inserting a media object and other media related events. Although these are all events in our design, it can be clearer if the insertions of media objects are emphasized.

3.3.5 Petri Net

Petri nets, originated from C.A. Petri's doctoral thesis [112] are graphical representations for modeling dynamic system behaviors. They are particularly suited to represent the activities in a distributed system. Currently, Petri nets are widely used in modeling various systems, including manufacturing systems ([34], [76], and [177]), web services ([102], [55], and [138]), and workflows ([144], [145], and [119]).

There are three types of objects in a Petri net, including *places*, *transitions*, and *directed arcs*. An example of a Petri net is shown in Figure 3.10. Directed arcs connect places and transitions, but not between places or transitions. Places are represented

by hollow circles, used to identify the states of the system. Transitions are depicted in bars. A transition may fire when one or more specific events occur, transferring the place to another. A place may potentially contain tokens. In the figure, place p1 processes three tokens. The number of tokens here is used to model the number of resources denoted by the place.

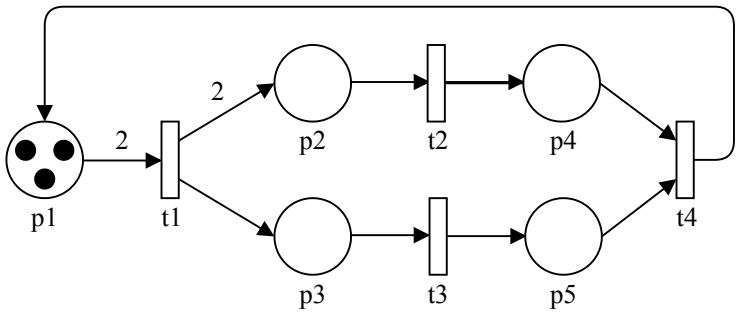


Fig. 3.10.: A Petri net.

According to the study of Wang [160], a Petri net is defined as a 5-tuple $N = (P, T, I, O, M_0)$, where

1. $P = \{p_1, p_2, ..., p_m\}$ is a finite set of places;
2. $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions;
3. $I : P \times T \rightarrow N$ is an input function that defines directed arcs from places to transitions;
4. $O : T \times P \rightarrow N$ is an output function that defines directed arcs from transitions to places;
5. $M_0 : P \times N$ is the initial marking.

There is a crucial concept, *marking*, in a Petri net. A marking is utilized to record the number of tokens assigned to the places in a Petri net. For example, in Figure 3.10, the initial marking M_0 is $(3\ 0\ 0\ 0\ 0)^T$. The position and number of tokens may change the execution of a Petri net, thus tokens can be used to model the execution of the net. Specifically, there are several firing rules that a Petri net needs to follow:

1. Enabling rule. A transition t is said to be enabled if each input place p of t contains at least number of tokens equal to the weight, $w(p,t)$, of the directed arc. Take the transition t_1 in Figure 3.10 as an instance, t_1 is said to be enabled because the place t_1 contains 3 tokens, larger than $w(p1,t1)$, which is 2.

2. Firing rule. To fire an enabled transition t, $w(p,t)$ tokens need to be removed from each input place p of t, and $w(t,p)$ tokens are deposited to each output place p of t.

3. Only an enabled transition can be fired. By contrast, an enabled transition may not be fired.

In Figure 3.10, the initial marking M_0 is:

$$M_0 = (3\ 0\ 0\ 0\ 0)^T.$$

Here, transition $t1$ is enabled. If we fire this transition, $w(p1,t1) = 2$ tokens need to be removed from place $p1$, and $w(t1,p2) = 2$ and $w(t1,p3) = 1$ tokens added to place

$p2$ and $p3$, respectively. This update is shown in Figure 3.11. As a result, the new marking, say M_1, is:

$$M_1 = (1\ 2\ 1\ 0\ 0)^T.$$

If we continue this procedure to fire transition $t2$ and $t3$, then the new marking, M_2, is:

$$M_2 = (1\ 1\ 0\ 1\ 1)^T.$$

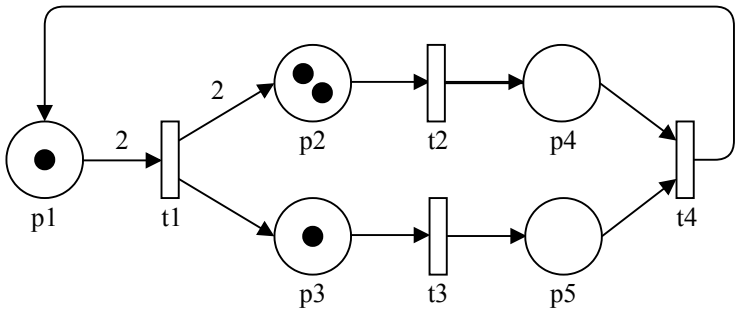
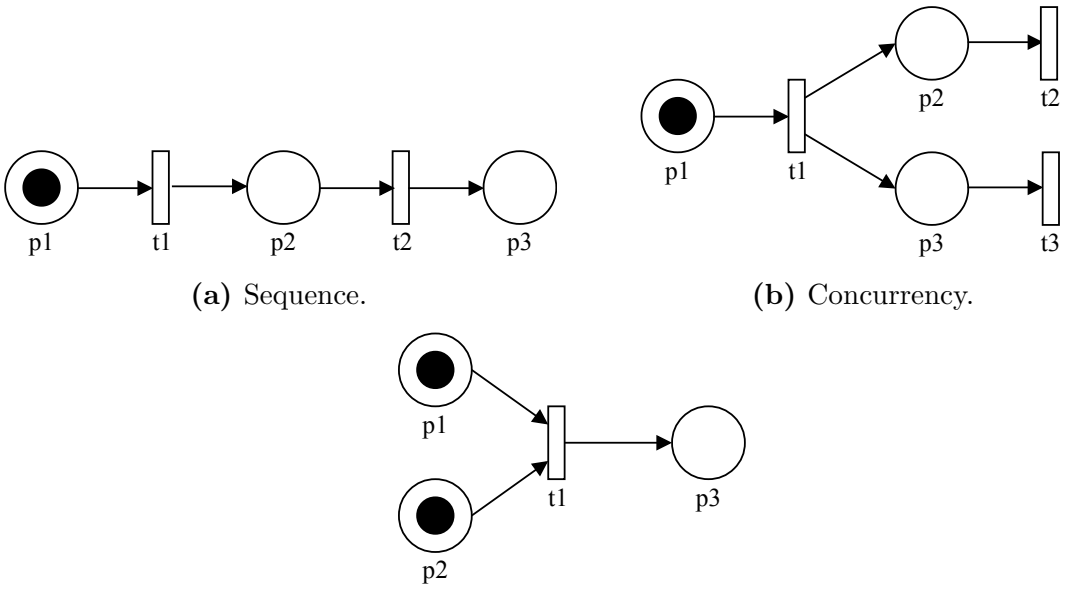


Fig. 3.11.: Transfer of tokens by firing transition $t1$.

A Petri net is strong at modeling multiple characteristics of event-driven systems, including sequence, concurrency, and synchronization, illustrated in Figure 3.12. Considering the strengths of Petri nets, Some pioneering work including [83], [27], and [59] has applied Petri nets to model temporal relations in multimedia systems.

Although Petri nets are very powerful in modeling event-driven systems, a problem when applying them to our document system is that they cannot indicate the time elapsed between events. To solve that, we would like to introduce a variance of Petri net.



(a) Sequence.

(b) Concurrency.

(c) Synchronization.

Fig. 3.12.: Modeling system features.

3.3.6 Timed Petri Net

Timed Petri nets [117] extend basic Petri net by introducing timing variables in transitions. There are two widely used timed Petri nets: *deterministic timed Petri net* (DTPN) [117] where the time labels are deterministic, and *stochastic timed Petri net* (STPN) [95] where the time labels are random. As the delays among events in our document system are deterministic, we focus on the deterministic timed Petri net in this section.

Formally, for a DTPN, a function τ is introduced to associate transitions with deterministic time delays. A simple example of applying a DTPN to our document system is demonstrated in Figure 3.13. Here, a 5 time units delay is attached to transition t_1 and a 8 time units delay is associated to transition t_2. With these time

labels, it is clearer to model the timing information in a presentation, as a user can explicitly assign a time delay to every document event.

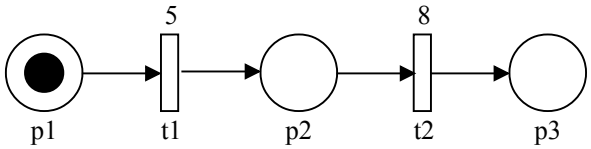


Fig. 3.13.: A timed Petri net.

Although timed Petri nets include timing variables based on the original Petri nets, there are still some drawbacks preventing us from directly applying them to our document system.

- Timed Petri nets lack special symbols to represent terminating presentation flows. Because of this, they are not good at modeling the complicated presentation flows of various media objects simultaneously.

- To our knowledge, in previous work, no usage of Petri nets can support real-time interaction on the predefined temporal models.

- Similar to activity diagrams, timed Petri nets cannot differentiate media object events from media action events, which makes it difficult to achieve some useful functionalities such as collapsing multiple nodes into one.

3.3.7 Document Net

Definition

To overcome the issues of previous models, based on activity diagrams and timed Petri nets, we propose a new modeling notation named **Document Net**, which is highly correlated with the document concepts defined in Section 3.2. A document net consists of basic controlling nodes, document event nodes, and essential transitions. Each transition is associated with a timing variable, similar to a deterministic timed Petri net. For transitions, they can fork new nodes or join multiple nodes together, used to model concurrency and synchronization. Tokens are also introduced to nodes so as to model the real execution flow in a presentation.

A document net is formally defined as a 8-tuple $N = (D, T, A, w, \tau, s_I, s_F, M_0)$, where

1. $D = \{d_1, d_2, ..., d_m\}$ is a finite set of document states;
2. $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions;
3. $A \subseteq (D \times T) \cup (T \times D)$ is the set of arcs from document states to transitions and from transitions to document states.
4. $w : A \to \{1, 2, 3, ...\}$ is the weight function on the arcs.
5. $\tau : T \to \mathbb{R}^+$ is a timing function to associate each transition with a positive deterministic time delay.
6. $d_I \in D$ is the initial document state, and $d_F \in D$ is the final document state.

7. M_0 is the initial marking.

Notations

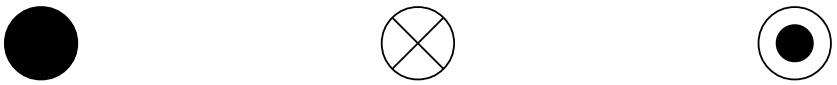
In a document net, states are depicted by various circles. Similar to activity diagrams, there are three basic types of nodes: initial nodes, flow final nodes, and document final nodes, as shown in Figure 3.14a, 3.14b, and 3.14c. These three types of nodes do not have any document events attached but are used to signal the begin/end states. By default, an initial node contains one token.

- Initial node. This marks the start of the document net, shown as a solid circle. There can be multiple initial nodes in a document net, although in a real presentation, we usually have only one starting point.

- Flow final node. This indicates the end of a presentation flow of a specific multimedia object, depicted as a hollow circle with a cross inside. It indicates the removal of a multimedia object from a document panel.

- Document final node. This stops the whole presentation activity, as well as all presentation flows. The notation of a document final node is a circle with a solid circle inside.

Besides the three basic control nodes, we also design notations of DocEvent nodes, shown in Figure 3.15a and 3.15b. DocEvent nodes are all depicted in circles, and we categorize them into two types: media and control. The reason for this division is to emphasize the difference between the event to insert a multimedia object and the

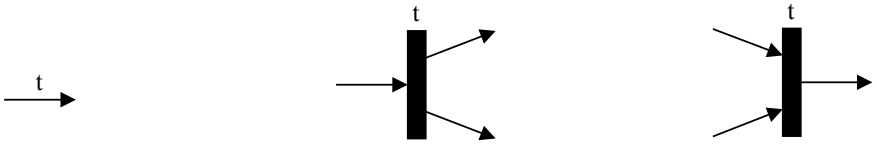


(a) Initial node. **(b)** Flow final node. **(c)** Document final node.

Fig. 3.14.: Basic control nodes in a document net.

(a) DocEvent node (media). **(b)** DocEvent node (action).

Fig. 3.15.: DocEvent nodes in a document net.



(a) Transition (basic). **(b)** Transition (fork). **(c)** Transition (join).

Fig. 3.16.: Transitions in a document net.

control of the object. Although these are both events, there is unlikeness between these two events. The insertion of a multimedia object needs a container, DocPanel, to hold the object, as well as a load of local or external resources. By contrast, a normal media control, such as muting a video, is only executed on a specific existing multimedia object. In our design, a DocEvent node (media) is depicted by a circle with a plus sign inside, and a DocEvent node (control) is just a plain circle. In a real document net, to differentiate the particular media object, we can also assign a color to corresponding DocEvent nodes. For example, all DocEvent nodes related to a video are marked in red, and all DocEvent nodes of an image are colored in green.

Transitions also fall into three categories, as shown in Figure 3.16a, 3.16b and 3.16c: basic transitions, fork transitions, and join transitions, akin to notations in activity diagrams. Transitions are used to connect nodes, as nodes cannot be connected directly. Each transition also contains a time delay and weight. A transition is enabled only when the number of tokens in the previous node (document state) d is at least equal to the weight of the transition.

- Transition (basic). This is the basic type of transition, used to model sequential execution. The DocEvent nodes connected are executed continuously with a deterministic time delay after the transition is enabled and fired. By default, the weight of a basic transition is one.

- Transition (fork). This group of transitions is utilized to model concurrent presentation flows. By default, a fork transition consumes one token and generates as many tokens as its number of outgoing edges. In Figure 3.16b, as the number of outgoing edges is two, the fork transition generates two tokens, one for each following DocEvent node.

- Transition (join). Join transitions are created to model synchronization among various presentation flows. After the transition, the input flows are joined together to a single flow. By default, a join transition is enabled only when every input DocEvent node has one token. In Figure 3.16c, the join transition's weight is two as there are two input DocEvent nodes.

Modeling

Sequence. Sequential executions are modeled with basic transitions, basic control nodes, and DocEvent nodes. Figure 3.24 illustrates an example of a sequence, based on the diagram shown in Figure 3.9. Here, the time delay after a node is marked with a real time in seconds, and events of the video and image are labeled in distinct colors. All events are fired in sequential order. The user first inserts a video block, plays, and pauses the video. Afterward, the presentation flow of the video is terminated and the focus is moved to an image inserted. Afterward, the image is zoomed in. Finally, the whole presentation is terminated.

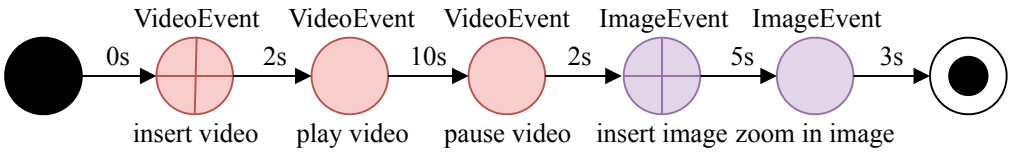


Fig. 3.17.: Modeling sequence.

Concurrency. Concurrency is a common pattern in a multimedia presentation. We exhibit an example of concurrency in Figure 3.18, by forking two additional presentation flows of an image and a PDF document, 2 seconds after pausing a video. In our document system, as the image and PDF document are shown in parallel, they cannot be placed on the same document panel.

Synchronization. Based on the diagram in Figure 3.18, we incorporate a join transition to synchronize the presentation flow of the image and that of the PDF document and continue the presentation flow of the video, as shown in Figure 3.19.

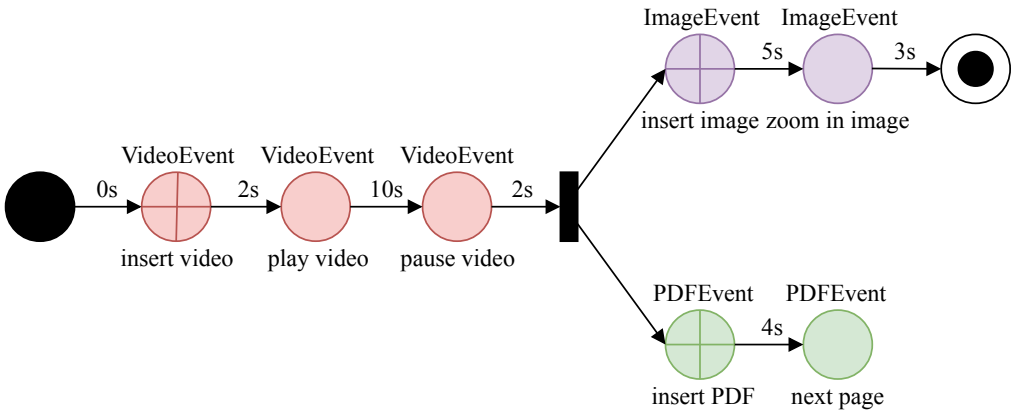


Fig. 3.18.: Modeling Concurrency.

The concurrent image flow and PDF flow are synchronized back to present the main video object.

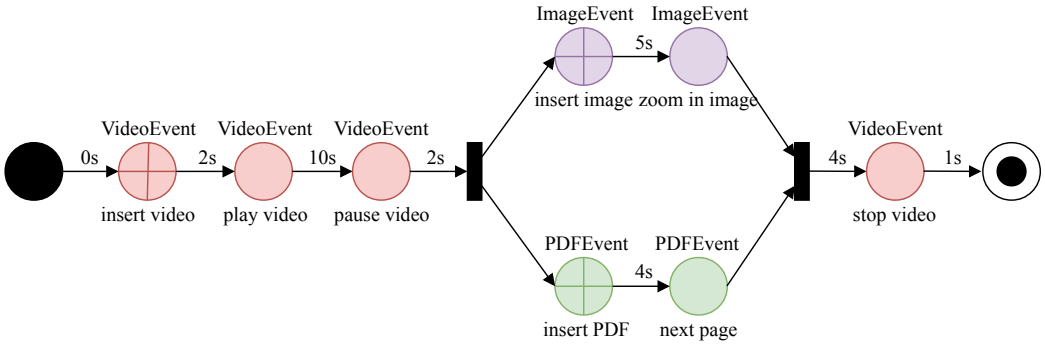


Fig. 3.19.: Modeling Synchronization.

Multi-Level Layout

Because we differentiate the media nodes from control nodes, a document net can consist of multiple levels. The series of subsequent media control nodes are allowed to be collapsed to a media object node, as shown in Figure 3.20. The figure is still based on the example in Figure 3.17. The document event nodes with event *play video* and

pause video are collapsed into the video object node. And a plus sign is applied here, indicating the node can be unfolded to show the hidden nodes.

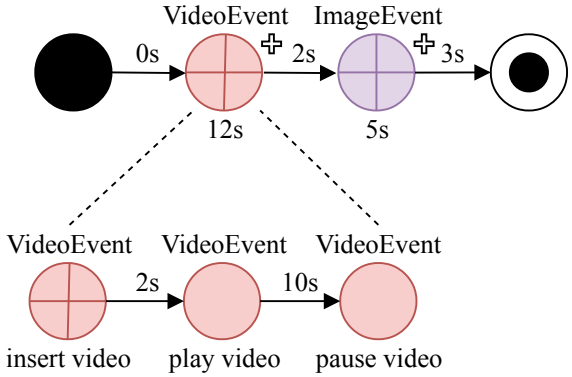


Fig. 3.20.: Multi-level layout in modeling sequence.

Another example is shown in Figure 3.21, based on the synchronization scenario described in Figure 3.19. Here, the events of the video, image, and PDF objects are collapsed, generating a much simplified diagram.

This mechanism generates a collapsed diagram, clearer for demonstrating the relations among multimedia objects. And the specific media control events are only shown when necessary.

Multimedia Synchronization

Multimedia synchronization is a crucial concept in document nets, especially employed for presentation replay and insertion/deletion of events in a presentation. Because all time delays in a document net are deterministic, a complex document net can be rendered into a sequential timeline.

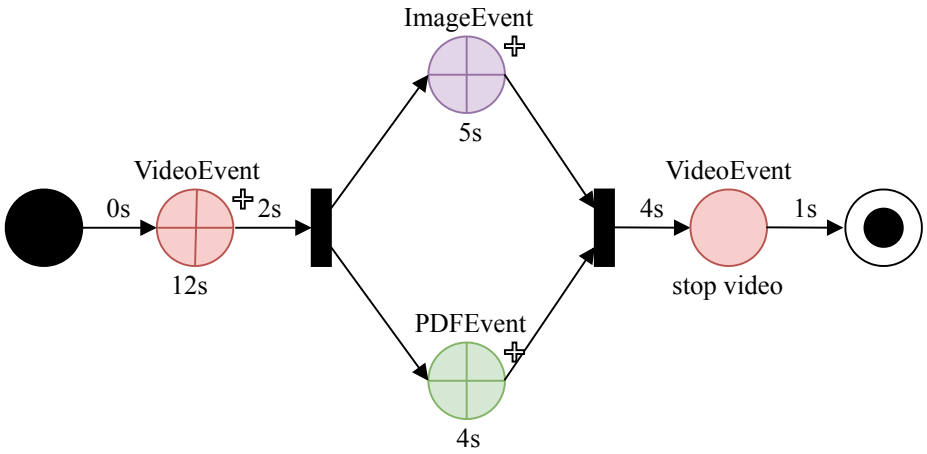


Fig. 3.21.: Multi-level layout in modeling synchronization.

As the timeline is sequential, the replay of a presentation can be achieved by executing the events one by one, according to the specific timestamp to fire an event. Take the diagram in Figure 3.18 as an instance, the original document net can be converted to a timeline shown in Figure 3.22. For the concurrency of image and PDF insertions, the two concurrent events are also condensed into two sequential events with 0 as the time interval. And sub-sequential media actions are arranged on the timeline according to their delays.

For insertions of new events, the new events can also be arranged on a timeline based on the timestamps these events took place. An example is shown in Figure 3.23. Suppose that during the presentation, the presenter pauses the flow at the vertical line, where 1 second has elapsed after the insertions of image and PDF. The presenter injects a web page to the document system before the presentation flow is resumed. As the new event has a timestamp attached, the event can be wrapped into a document event node, and placed between the original node with *insert PDF* event and the

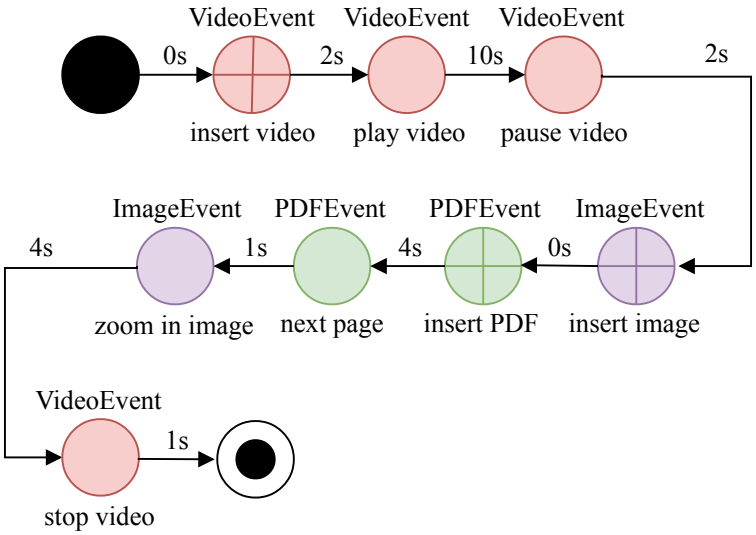


Fig. 3.22.: Rendering of a document net for presentation replay.

node with *next page* event. To remove a prepared event, the presenter can just skip the event, directly jumping to the next event.

3.3.8 DocEvent Structure

In the structures of document nets, the DocEvent nodes play a crucial role in modeling the events occurred in presentations. To simplify the implementation, we do not create an extra document transition class but wrap the necessary information of transitions into DocEvent nodes. Although we categorize the DocEvent nodes into two types, they have similar implementation skeletons. From the perspective of object-oriented design, every DocEvent node object should have the following data members:

- **media_type**, the type of media where the document event takes place. For example, for the event of playing a video, the media type here should be "video".

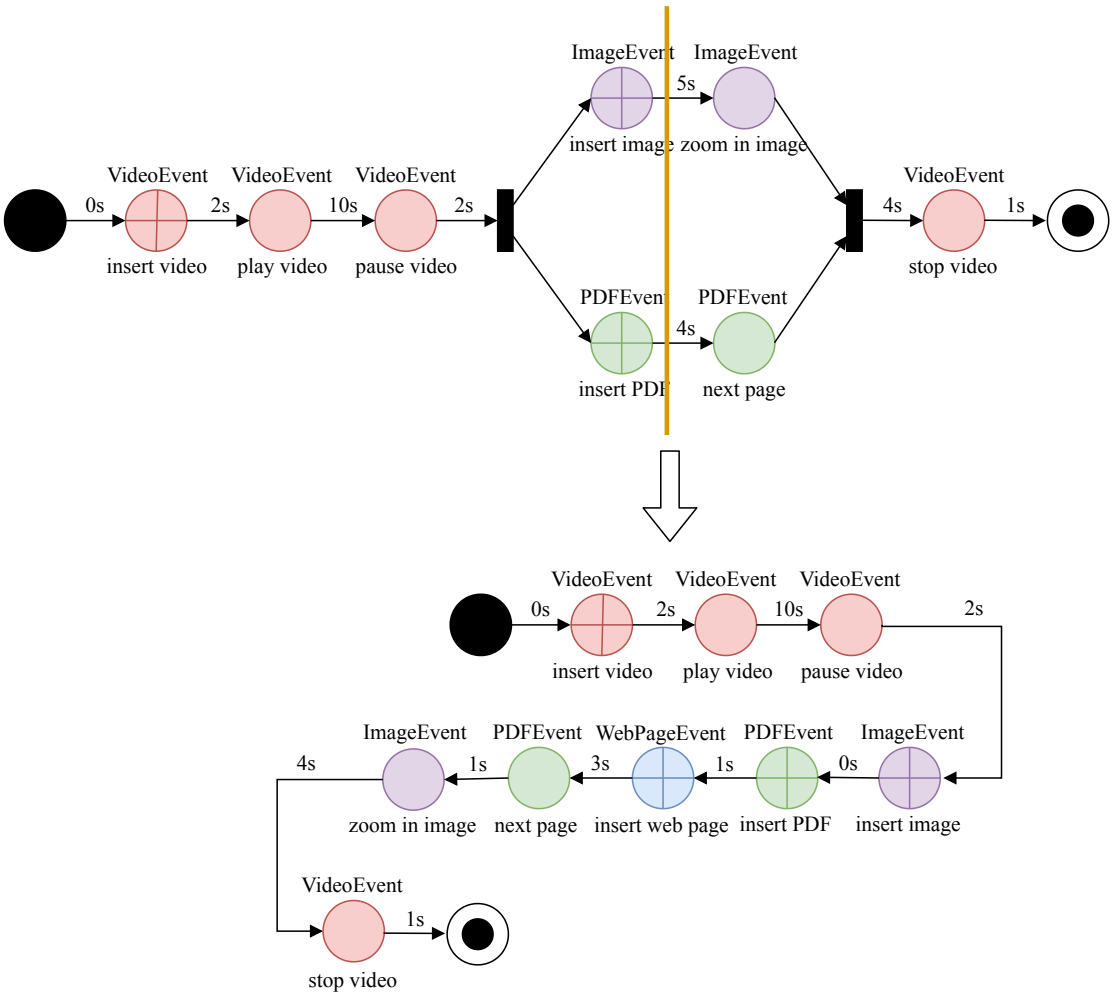


Fig. 3.23.: Rendering of a document net for new events.

- **media_id**, the unique identifier of the document object the event linked to. This is used to identify the specific media object where the event fires.

- **event_type**, the type of the event represented by the document event object. In our design, we capture a media event by monitoring the structure of DOM to achieve efficiency and portability. Thus, the event type here usually refers to a DOM event. For example, the event type of pausing a video can be "but-

ton_click". Notably, there is also an event type of the insertion of a document object. For example, the event type of inserting a video object is "insert_video".

- **description**, the semantic human-readable description of the event. For example, the description of zooming in a Google map can be "zoom in a map".
- **enabled**, the boolean expression to mark whether the event has been enabled or not.
- **fired**, the boolean expression to mark whether the event has been fired or not.
- **time_delay**, the elapsed time to fire the document event after it has been enabled. This is used to model the transition, such that we do not need to create an extra transition object.
- **in_doc_event_ids**, the array of input document events' ids. In our document system, the current document event node cannot be enabled until all of the input document events have been fired.
- **out_doc_event_ids**, the array of output document events' ids.

Take the diagram shown in Figure 3.19 as an example. After the DocEvent of pausing a video is fired, because of the fork transition, both of image insertion and PDF insertion DocEvents are enabled and fired subsequently after 2s. Afterward, the image and PDF are presented concurrently. When the image has been zoomed in and the PDF has been scrolled to the next page, the stop video event is enabled. After 4s, the event is fired, and the presentation flow comes back to the video. Token

information of each DocEvent is omitted, as by default, the number of elements in *in_doc_event_ids* variable is exactly the number of tokens required to enable an event.

3.3.9 Temporal Constraints

In Figure 3.17, 3.18 , and 3.19, there are no flow final nodes. Then why do we need to introduce flow final node in a document net? A flow final node indicates the end of the presentation flow of a specific media object, so that the media object can be destroyed and removed from a document panel. Afterward, another media object may occupy the space. For example, in Figure 3.24, we illustrate an example of inserting a flow final node after pausing the video. After the video flow is terminated and the video is removed from the panel, the new image object can be placed in that panel. The concrete panel is determined by *panel_id* member in the image object. The lifespans of the video object and the image object are shown in Figure 3.25b. At timestamp t, the video object must be destroyed.

By contrast, if the video flow is not terminated, like that in Figure 3.17, we should not place the new image on the same panel as the video. Here, the lifespans are illustrated in Figure 3.25a. After timestamp t, the new image object is created, and the video object still exists. The image object needs to be placed on another document panel.

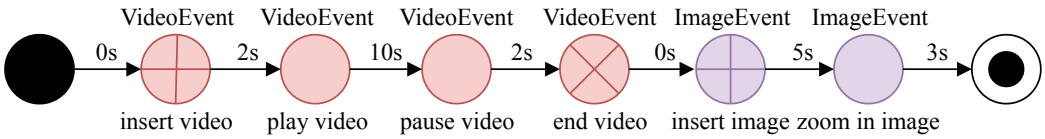
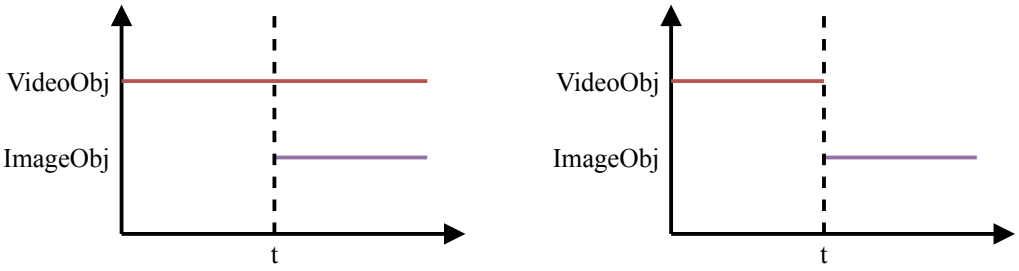


Fig. 3.24.: Modeling sequence with a flow final node.



(a) Lifespans based on Figure 3.17. **(b)** Lifespans based on Figure 3.24.

Fig. 3.25.: Lifespans of VideoObj and ImageObj.

3.4 Spatial Modeling

3.4.1 Introduction

In our document system, not only the temporal but also the spatial relations among document objects are modeled. Similar to the requirements of temporal modeling, the spatial relations should also be reflected in the document nets, supporting interactive controls and replay.

As a document panel is a container for document objects, the spatial relations among document objects can be represented by the spatial relations among the panels. Additionally, as document objects can be created and destroyed, document panels are always in the document system, maintaining much more stable spatial relations.

Considering the spatial modeling, there are two major questions we need to answer:

1. **How to represent the spatial relation?** One possible layout is shown in Figure 3.26, where seven document panels are arranged horizontally and ver-

tically. We need to come up with an approach to represent the layout. The representation can also be serialized and deserialized for persistence purpose.

2. **How to represent the spatial updates in a document net?** The core in the temporal modeling is the document net. To be consistent, the spatial models should also be prepared in document nets. Additionally, a spatial update may need to influence the original document net by firing some events.

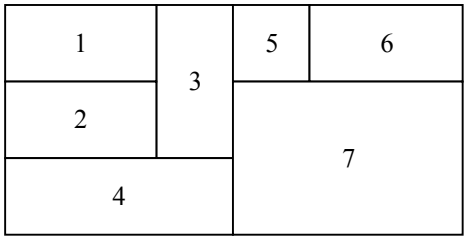


Fig. 3.26.: A document panel layout.

3.4.2 Slicing Tree

Slicing tree, introduced by Stockmeyer [136], is a type of slicing structures, used for floorplan design. It recursively slices a floorplan horizontally or vertically to formulate a tree structure. Some efficient slicing algorithms [106, 167, 169, 173] have also been invented and discussed in previous work, to find the optimal cuts of a floorplan. Moreover, Lai and Wong [68] proved mathematically that the slicing tree is a complete floorplan representation.

Based on the layout in Figure 3.26, we can create a slicing tree shown in Figure 3.28. Here, node + represents a horizontal slice, and node * means a vertical slice. A Polish expression can be obtained through the postorder traversal of the tree.

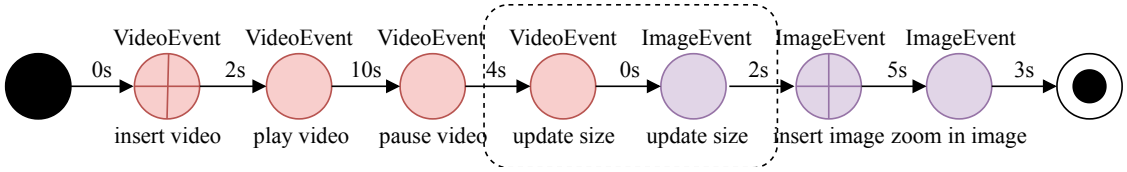


Fig. 3.27.: Modeling layout updates in a document timeline.

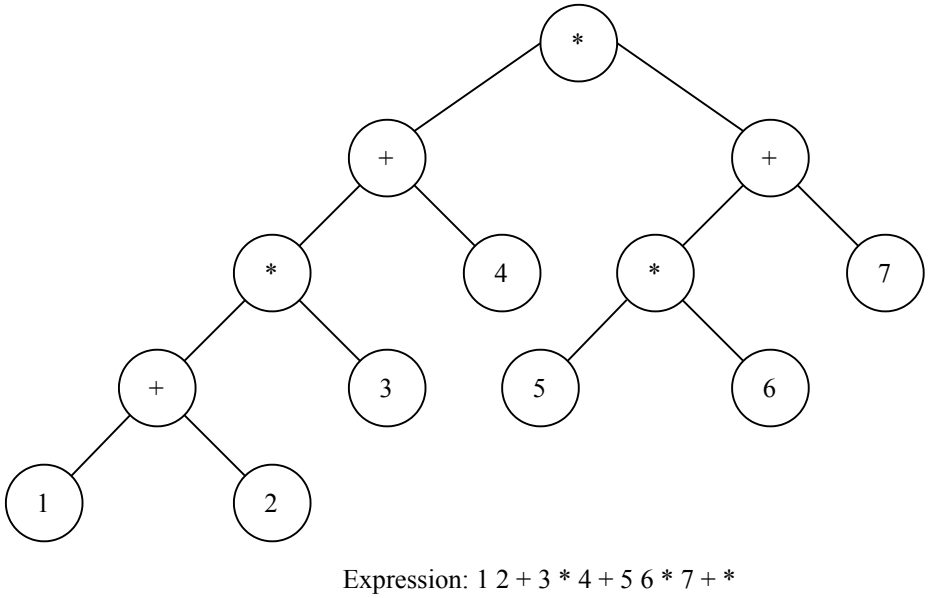


Fig. 3.28.: Representation of a floorplan in a slicing tree with expression.

The expression can further be used to model spatial relations. Deserialization of the tree can also be achieved by parsing the Polish expression to reconstruct the slicing tree.

As a cut may lead to two blocks with distinct sizes, each internal node of a slicing tree also needs to store ratio information of the two children. As a result, when a presenter drags a slicing line to change the cut, only the ratio information stored in internal nodes are updated, leaving the tree structure untouched.

3.4.3 Modeling Updates

It is common that in a presentation, the presenter changes the layout, as shown in Figure 3.29, based on the scenario in Figure 3.17. At first, both of the video and image objects cover half of the space, sliced vertically. Afterward, the presenter updates the ratio of space the video covers to 40% and image to 60%. As every control in the document system is modeled as an event, this update also needs to be represented in events and reflected on the document net.

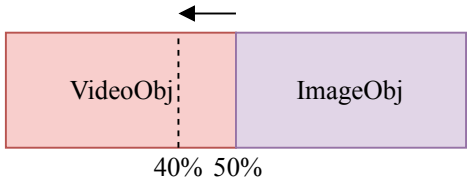


Fig. 3.29.: An update of area ratio.

In our design, the update of layout in Figure 3.29 can be represented by two document events, related to the size updates of both of the video and image objects. The two events are inserted into the document timeline for the record, as shown in Figure 3.27.

However, a question may be raised here about how to decide the specific document objects to update sizes. One straightforward approach is for the root internal node whose ratio is updated, use preorder traversal to check every descendant, and for a leaf node if necessary, fire a document event to update the object's size.

3.5 Implementation

3.5.1 Overview

Based on the modeling approaches described in the previous section, we developed an interactive multimedia presentation document system in a client-server structure. We provide graphical interfaces for rich functionalities including presentation preparation, control, and replay, totally on web browsers. The backend server takes charge of serving multimedia resources, managing user accounts, and storing presentation data. The server is implemented in Node.js [26], which is an event-driven, non-blocking, and cross-platform JavaScript run-time environment. We use MongoDB [96], a document-oriented NoSQL database, to store data in the platform.

3.5.2 Presentation Preparation

Before a presentation, users can upload essential multimedia files to the server, so that users can still have access to the files on other machines. Users may also choose not to upload the files, but provide the URIs of the materials instead, and the URIs will also be stored on the server side.

Besides the preparation of multimedia files, users can also prepare specific document events. To achieve that, we provide a graphical interface written with the help of GoJS [103], which is a JavaScript library for building interactive diagrams on the web. An example of using our preparation tool is shown in Figure 3.30. Here, the basic document net controlling nodes as well as prepared multimedia materials are

shown on the left palette. A user can drag and drop various components onto the canvas to formulate a diagram of multimedia events. The diagram can be saved to a JSON-format file with the information of nodes and links, and the diagram will further be transformed into an event timeline for interactive controls, as discussed in Section 3.3.7. Every document event is serialized to a JSON-format message for storage purposes, based on the document event structure described in Section 3.3.8.

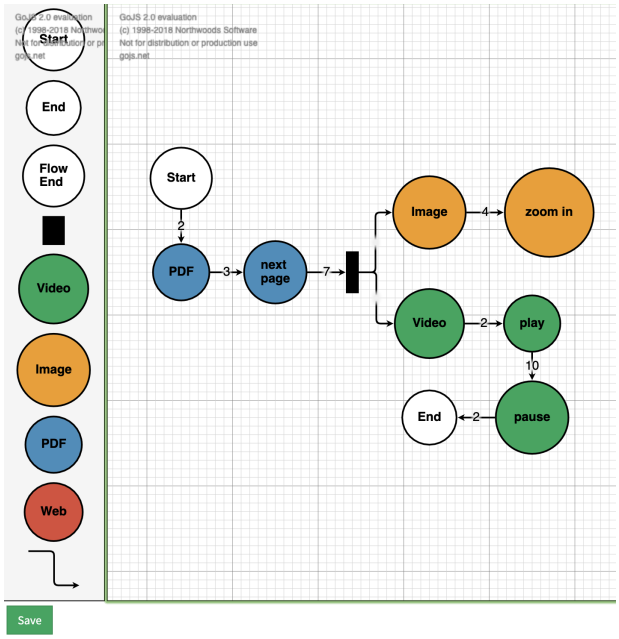


Fig. 3.30.: The tool for preparing multimedia events.

3.5.3 Presentation Control

One crucial feature that differentiates our work from prior work is that we provide rich functionalities on interactive controls of multimedia presentations. In our systems, videos, images, PDF documents, web pages, Google maps, and external dynamic tables are all supported for presentation purpose with essential annotation

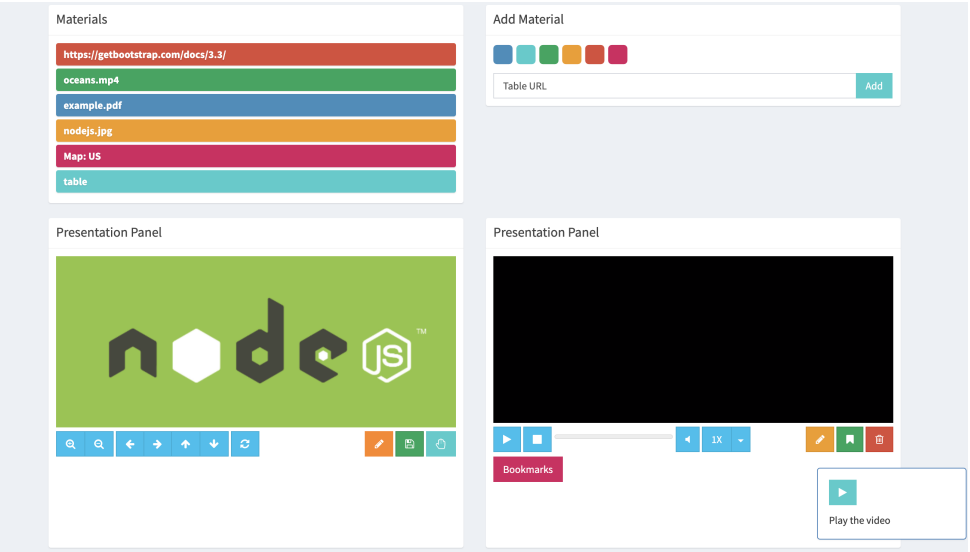


Fig. 3.31.: The graphical interface for presentations.

tools. The user interface of controlling a presentation is shown in Figure 3.31. There are four major UI components in the figure: *Materials*, *Add Material*, *Presentation Panels*, and *Event Toolbar*.

The *Materials* panel contains the material resources in a presentation session. In the figure, there is a web page (https://getbootstrap.com/docs/3.3/), a video (oceans.mp4), a PDF document (example.pdf), an image (nodejs.jpg), a Google Maps block, and an external table block. Besides the material files prepared beforehand, a presenter can also insert new materials in a presentation session from the *Add Material* block. The added materials will also be shown in the *Materials* panel. The *Presentation Panels* are containers for document objects. In the figure, an image and a video are presented simultaneously. Currently, each panel covers half of the space. A presenter is allowed to modify the spatial ratio, according to specific presentation demands. The *Event Toolbar*, floating on the bottom right of the figure, provides the

mechanisms to play/pause the replay of the prepared events. A semantic description of the next event is also displayed on the panel. Users are allowed to pause the replay and update the prepared model by adding new events on the fly. The timeline will be adjusted accordingly, and the updates will also be reflected on the original document net. After a presentation, the user can view the latest version of the document net which depicts the real presentation flows.

In our system, a presentation can be recorded and stored as the combination of multimedia materials and related events, represented by simple messages described in Chapter 5. An example is shown in Figure 3.32. In Figure 3.32a, the messages of files used in a presentation are illustrated. And Figure 3.32b shows some of the events that took place in a past session.

3.5.4 Presentation Replay

The replay is a crucial mechanism in our presentation document system. For the media events stored in a presentation, the events can be replayed one by one sequentially according to their timestamps. Moreover, as each event is represented by simple messages in JSON format, we can use only tens of kilobytes to store a specific presentation. Additionally, most of the events are related to multimedia objects, so the replays are accurate and efficient. We also provide a toolbar, similar to that for controlling presentations, to control a replay session. A user is allowed to play/pause the replay and omit the delay by directly playing the next event. The details of the replay mechanism are discussed in Chapter 5.

```
"files": [..., {
   "location": "/files/hashcode1.pdf",
   "type": "application/pdf",
   "name": "example.pdf"
}, {
   "location": "/files/hashcode2.mp4",
   "type": "video/mp4",
   "name": "oceans.mp4"
}, ...]
```


(a) Messages of multimedia files.


```
"events": [..., {
   "media-type": "video",
   "media-id": "video-block",
   "event-type": "button-click",
   "seq-id": 6,
   "timestamp": 10000,
   "description": "play video",
   "data": {
      "id": "video-playpause",
      "current-time": 0
   }
}, {
   "media-type": "pdf",
   "media-id": "pdf-block",
   "event-type": "button-click",
   "seq-id": 7,
   "timestamp": 12000,
   "description": "next page",
   "data": {
      "id": "pdf-next "
   }
}, ...]
```


(b) Messages of multimedia events.

Fig. 3.32.: Messages of a presentation.

3.6 Related Work

Distributed multimedia system (DMS) has a long history for both research and engineering purposes. Some of the pioneering work in this domain, including [2, 21, 81, 140] mainly concentrated on video conferencing, where various types of media are transmitted via video frames. Their target is to add interaction and collaboration into video-based multimedia systems. For example, in [21], the authors created a multimedia desktop collaboration system, which provides a shared workspace for

collaboration. The system supports efficient full-motion video and multiple video windows. Baura et al. [5] introduced multimedia systems into teaching courses related to computer architecture, organization, and design, indicating the capabilities of multimedia for education. Because it is difficult to evaluate the influence of interactivity in video-based multimedia system, Branch et al. [14] created a series of experiments, and they claimed that lognormal distributions are better than exponential distributions in describing interactive behaviors, especially where high levels of accuracy are required. Li [75] studied the influence of video interaction patterns for MOOCs and based on their findings, they provide some suggestions like detecting the change of video interaction patterns and providing quick access for revisiting videos. Furthermore, notice that the quality of services has an impact on multimedia systems, especially in wireless network condition. Berhe [10] proposed an architecture of the Distributed Content Adaptation Framework (DCAF), which adapts contents depending on the user's preferences, device capabilities, and network conditions.

With the gradual popularity of mobile devices and cloud services, some researchers studied multimedia systems based on mobile-cloud platforms. For example, Sreeramaneni et al. [134] proposed a context-based intelligent multimedia system, optimized upon efficiency for mobile users. They claimed that their context-aware decision-making algorithm improved mobile device performance and consumed less energy. Lin et al. [80] invented an efficient generic algorithm to solve the load balancing problem, especially considering that every server cluster only handles one type of media, in a cloud-based multimedia system. By contrast, Wen et al. [164] also proposed an algorithm for load balancing issue in cloud-based multimedia system, while they

consider the load of all servers and network conditions. On the other side, providing that security is another important aspect in distributed multimedia systems, Yang et al. [171] combined multimedia data state and role access control to propose a mixed security multimedia cloud transmission and storage system.

Recently, some research work concentrated on more effective interaction and collaboration on multimedia systems. For example, Lee et al. [70] combined virtual reality technology and context-aware computing to build a virtual storytelling application. They claimed that their approach could be beneficial to various applications such as education and entertainment. Moreover, Zhu et al. [179] introduced a concept of a library in every phase of development and created a complicated product oriented collaborative document management system. Ciocca et al. [24] built Quicklook2 which is an integrated multimedia system, allowing users to achieve relevance feedbacks related to queries to multimedia databases. Additionally, based on CVS conversioning system, Adler et al. [3] implemented a web-based collaborative office document system, TellTable. They also identified twelve challenges in collaborative editing software, including awareness, communication, workflow, platform independence, etc.

3.7 Conclusion

In this chapter, we proposed a web-based event-driven multimedia presentation document system. The system realized both temporal and spatial scheduling on

multimedia objects in document nets and supported real-time interactive control of these models.

The next chapter presents the approaches to continuous updates of external web resources. We evaluate and compare two models: continuous updates with a server and without a server.

4. CONTINUOUS UPDATES OF EXTERNAL WEB RESOURCES

4.1 Introduction

With the rapid development of web technologies, the web has become a very popular media for social media, online commerce, video communication, etc. At the same time, the number of web pages continues to grow at an astounding speed. According to a study by Douneva, et al. [29], there are almost 50 billion websites existed. The web has become a fundamental part of our daily lives.

In a presentation, it is common that the presenter would like to show some information linked from another external web page. Moreover, as the dynamic feature of web pages, he/she may want the information updated continuously according to the latest version of the external web page. An example is illustrated in Figure 4.1. Suppose that the presenter would like to demonstrate the current state of the stock market, and he/she extracts a specific market table from an external web page, such as Yahoo Finance, and places the table on the document system. The table is not static, and the user expects the table to be updated periodically or continuously, following the updates from the original web page.

Traditionally, this demand falls into the research category of the continuous query on the dynamic web, especially useful for web crawlers. However, our work is different

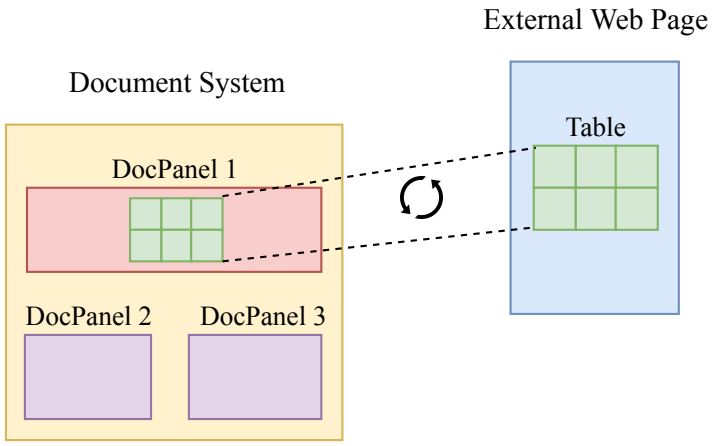


Fig. 4.1.: An example of loading an external table on the document system.

from previous work. As previous work focuses more on the performance of large scale crawling and change simulation of various web pages, our solution cares more about the performance influence of several web pages on the client side and timely updates of target web elements. We expect that a user does not need to place too many dynamic external web resources on the document system, and a user always wants to see the latest update of the element, especially providing that latency has a significant impact on user engagement. For example, based on a study of Google [132], a 400-millisecond delay caused 0.59% decrease in searches/users. At the same time, a user does not want to see much performance impact on the client side browser as our document system is totally web-based.

These expectations lead us to propose a solution for continuous queries quite different from previous work, and we would like to describe the details in this chapter. Especially, we analyze and compare two different approaches for continuous queries: continuous updates with/without a server. We also give some recommendation based on our study.

4.2 Challenges

The first challenge is how to achieve continuous updates without a server. Traditional continuous query methods deploy a backbone server to monitor updates of interesting web pages. While, in our system, this puts more burden on the setup of the system, making it difficult for a client to use. Additionally, we do not expect a user would like to monitor a large number of web elements, causing it not very essential to use a server. We compare the design with/without a server in Section 4.3 and Section 4.4.

The second challenge is to keep the system lightweight and efficient. It should not cost much CPU usage on the client side, as the computation power of web browsers is limited compared to servers'. Additionally, it should react timely to latest updates. A user always would like to obtain the latest states of the target web elements, without too long waiting. The traditional pull-based approach has a problem of precisely simulating the updates.

4.3 Continuous Updates with a Server

4.3.1 Implementation

The traditional approach to solve continuous updates of web resources, especially used in dynamic web crawlers, is to periodically query the target web page through a central server. In this structure, most of the work is put on the server side, and

the client needs to provide the target monitored web element and receive continuous updates.

Here, we illustrate the core concepts of this design by explaining an example of dynamically loading a table to our document system from an external web page, as shown in Figure 4.2,

- The user first selects the target table which needs to be extracted and loaded on the document system.
- Afterward, the table information is sent to the server for continuous updates. Here, a crucial message included is how to locate the specific table element. A possible identifier is the index of the target table of all tables in the web page. Additionally, if the chosen table already has a unique identifier, it can be directly used as the selector. However, these approaches are prone to changes in the target table node. For instance, if the table node is removed, the server will continuously query the wrong element.
- After the server obtains the information of the table element, it periodically queries the external web page and crawls the current state of the table.
- When the latest state is obtained, it is sent to the document system to update the table shown in the application.

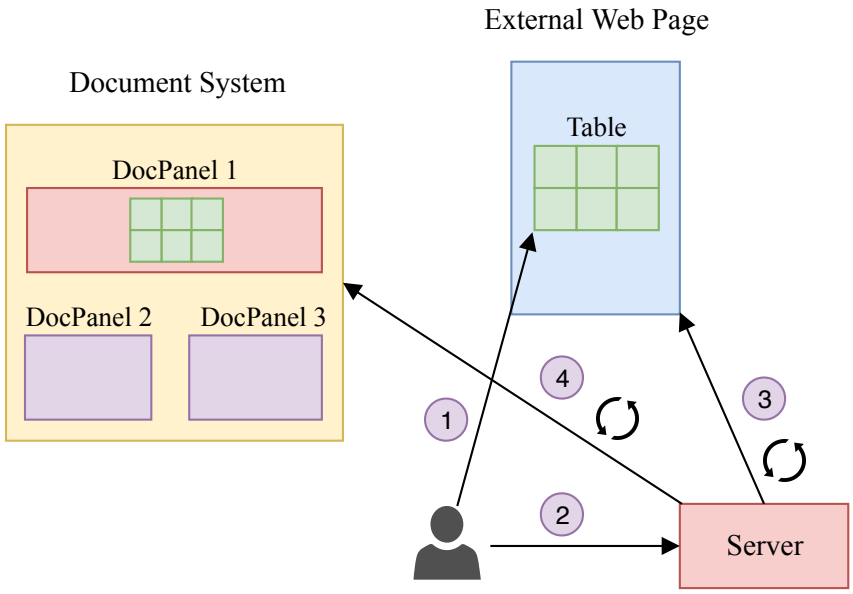


Fig. 4.2.: Continuous updates of an external table with a server.

4.3.2 Change Monitoring

The common approach used to monitor the changes of target web elements with a server is to periodically crawl the target elements and obtain the current states. A crucial issue raised here is how to determine the period. An example is shown in Figure 4.3. Here, the initial content of a table is a sequence of 1 to 6. In the 5th second, the number 5 is updated to 7, and in the 15th second, number 3 is updated to 9.

If we select the period as 15 seconds, obviously, as shown in Figure 4.4, we omit the update in the 5th second. This indicates that if the period is too large, some updates may not be captured. By contrast, if we choose 5 seconds as the period, as shown in Figure 4.5, even though we can capture all changes, there is one query which returns no update. This implies that if we set the period too small, some

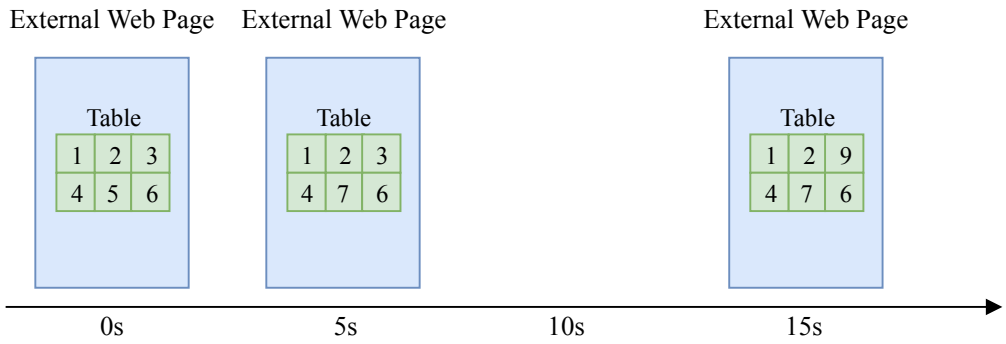


Fig. 4.3.: Dynamic features of an external table.

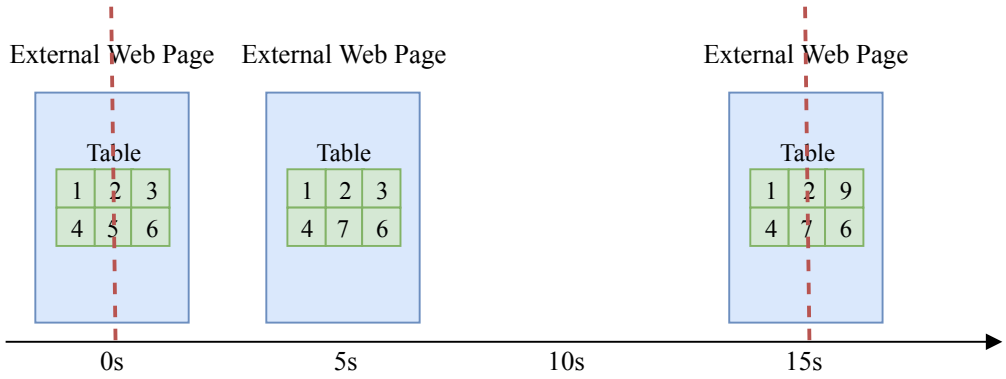


Fig. 4.4.: Continuous queries with large intervals.

computation resources will be wasted. In the example here, we have prior knowledge of the changes. Nonetheless, in practice, we usually have no idea of the distribution of updates, which makes it even more difficult to apply a pull-based approach to accurately capture changes as many as possible. Some more sophisticated simulation algorithms [62, 126] are also developed, although they still cannot capture all changes.

4.3.3 Communication

For the last step shown in Figure 4.2, the server needs to actively push updates to the client side. Here, some push-based protocols such as WebSocket [154] or server-

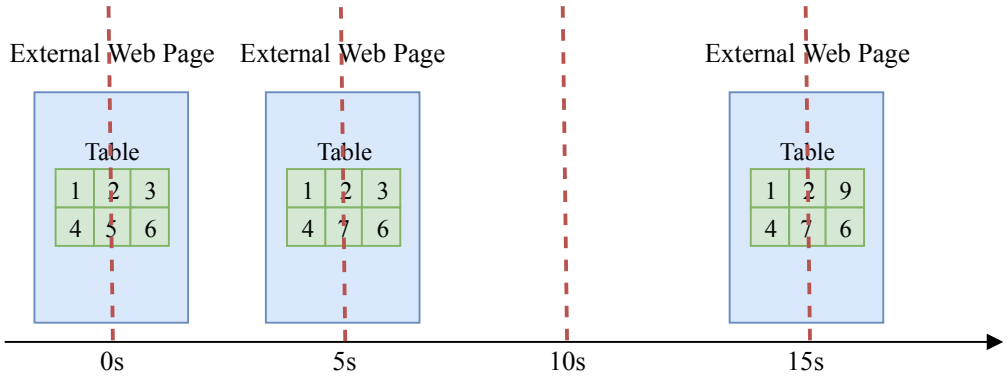


Fig. 4.5.: Continuous queries with small intervals.

side events [156] are necessary to achieve the communication. The document system registers corresponding event listeners beforehand to handle the incoming updates. An alternative way is to use polling on the document system side, periodically sending requests to the server to fetch the latest updates. We discuss the differences among these approaches in Chapter 5.

4.3.4 Summary

This design strips long running query tasks from the client, allowing the client side focusing on updating target web elements on the user interface. This is useful, especially when there are a large number of dynamic web elements to trace. However, as an additional server is necessary, it makes it difficult for users to easily set up the document system without extra external services. Furthermore, because dynamic web elements are usually updated by JavaScript code, to crawl the target elements, headless web browsers like PhantomJS [113] and Headless Chrome [49] with extra crawling techniques are essential to monitor the changes accurately.

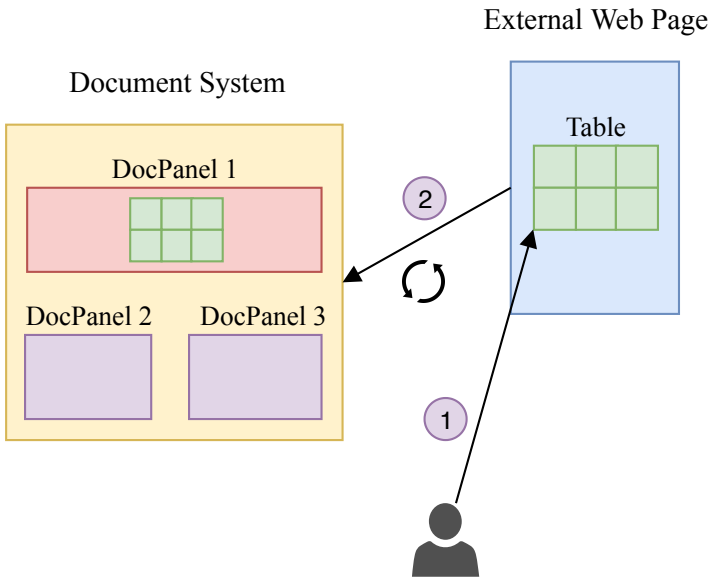


Fig. 4.6.: Continuous updates of an external table without a server.

4.4 Continuous Updates without a Server

4.4.1 Implementation

An alternative way is to monitor the updates of target web elements without intervening of servers, totally on the client side. To achieve that, as shown in Figure 4.6,

- The user still needs to select the target table element to monitor and load.
- Without a server, we want to continuously obtain the updates directly from the external web page.

However, these requirements raise some issues in the implementation. The first problem is the communication between two web pages, as there is no server-client communication. Another challenge is that the external web page has to actively monitor the changes in the target elements, which needs extra control for help.

To solve the issues above, we propose to create a Chrome extension to inject extra JavaScript code in the external web pages. As shown in Figure 4.7, a *content_script.js* file is injected in the external web page beforehand. Afterward, the user selects the target table element, and the injected JavaScript code continuously captures the changes of the element and sends them to the document system.

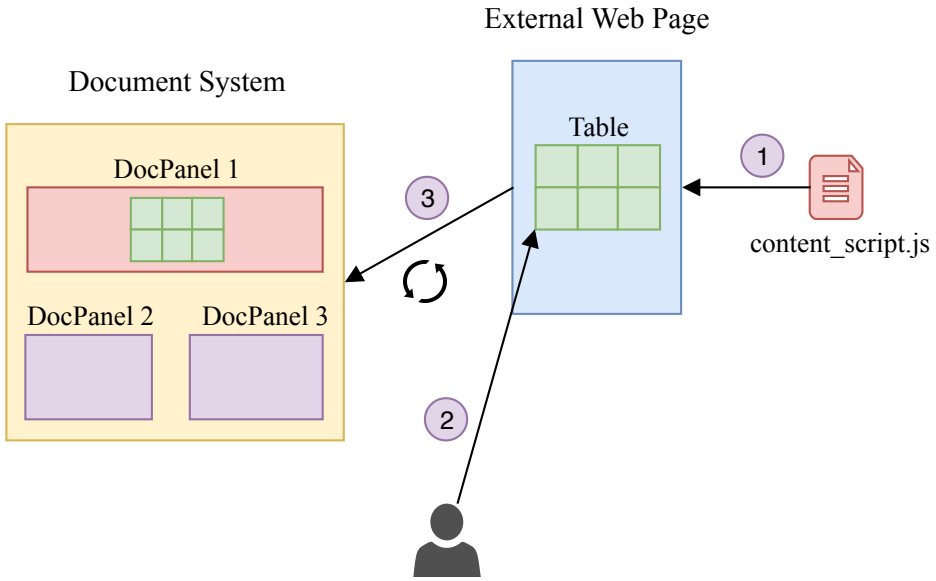


Fig. 4.7.: Continuous updates of an external table via a Chrome extension.

So instead of a central server, the continuous query task now is handled by the Chrome extension, and the document system still needs to wait for the updates and refresh the user interface.

4.4.2 Change Monitoring

A crucial design issue in this structure is how to monitor the changes in target web elements totally on the client side. The pull-based approach used in the structure

with a server can also be used here to actively query the current state of the target element.

An alternative way is to use an event-driven design with DOM events. Usually, the update of a web element can be reflected in one or more DOM events. For example, the change of value in a table cell can be captured by listening to *DOMSubtreeModified* event of the cell. The change of an image on the web page can be captured by monitoring the *load* event of the image. A more general way is to utilize the MutationObserver [100] interface, which provides the capability to watch for the changes in the DOM tree. While, an issue of the event-driven approach is that because all the monitors are on the client side, achieved by a Chrome extension if there are many changes occurring in a short period, too many listeners will be invoked, which puts a burden on the browser. This is illustrated in Figure 4.8. Here, all 6 values in the table are updated at the same time, and if we use MutationObserver to listen to the change of each cell, the mutation event will be triggered 6 times. In our experiment, when 100 table cells are updated simultaneously, as much as 6% CPU usage will be increased by, test on a MacBook Pro. Considering the computation burden on the client side, we recommend using the pull-based approach to provide a more lightweight browser extension. However, if we have enough prior knowledge that only a few events are necessary, we can exploit the event-driven method to achieve more accurate control.

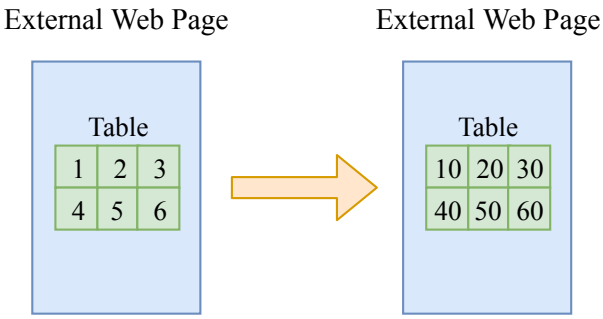


Fig. 4.8.: An example of updating several table cells at the same time.

4.4.3 Communication

Here, we use the postMessage API [149] to achieve the communication between two browser windows, overcoming the cross-origin communication problem.

4.4.4 Summary

In this structure, no server is involved and all monitoring and loading work is completed on the client side. This makes it convenient for a user to set up the usage of continuous updates of external web resources. Definitely, this will add some burden on the browser. Additionally, as the chrome extension needs to continuously query the web page, the tab of the web page has to be kept open during the usage.

4.5 External Web Resources Loading

4.5.1 Introduction

Traditional continuous queries on dynamic web pages concentrate on the updates of values, instead of styles of target web elements. By contrast, in our document

system, the external web element shown needs to be as much as possible similar to its original version, so the styles are essential for consideration and extraction. In this section, we discuss two different approaches to load external web resources: *web elements with styles* and *screenshot*, both implemented in our document system.

4.5.2 Web Elements with Styles

In this approach, we walk through all the styles, including the height, width, color, position, etc of the target element. The styles are pushed to an array and transmitted to the document system. One of the advantages of this method is that we can accomplish quite fine-grained continuous updates, as illustrated in Figure 4.9. Here, suppose we use the MutationObserver API to monitor the changes of each table cell. When only one cell's style is mutated, the new style can be captured and only this cell is updated on the document system, without any influence of other table cells. Nonetheless, a possible drawback of this approach is that because the original styles are read and applied on the document system which is a totally different web page, some collisions of the styles may occur, also makes it difficult to come up with a general approach for various types of web elements.

4.5.3 Screenshot

Unlike extracting the styles of target elements, in the screenshot mode, we capture the whole web element into a photo and synchronize the photo to the document system, such that the original styles are not big trouble anymore. Furthermore, this

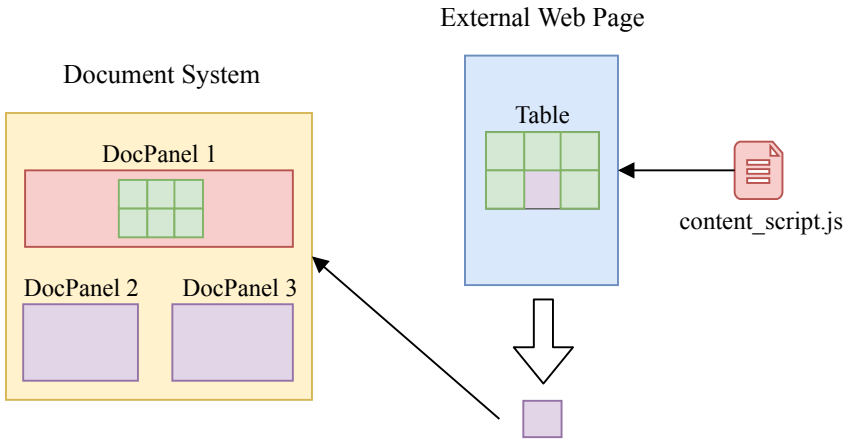


Fig. 4.9.: Continuous updates of an external table with styles.

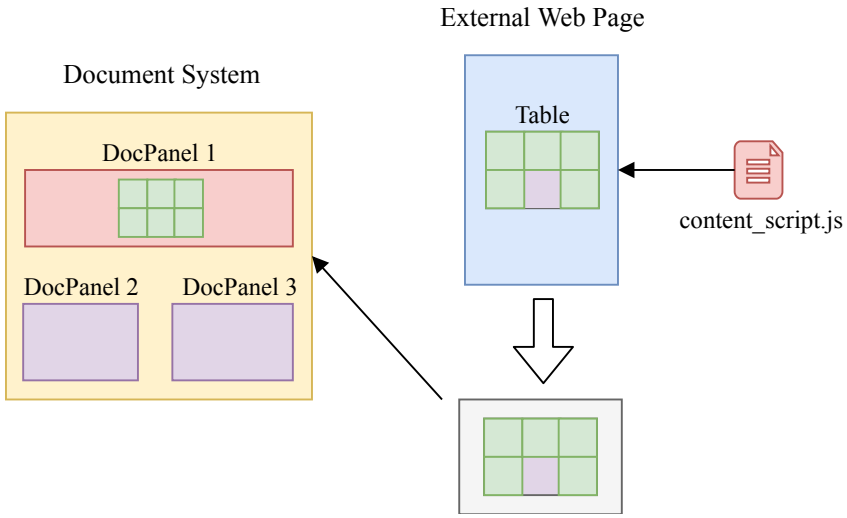


Fig. 4.10.: Continuous updates of an external table with screenshots.

method dramatically reduces the overhead of reading all styles, although as shown in Figure 4.10, the updates may be more coarse-grained compared with the previous method. Even if only one table cell is changed, we may still need to capture the whole table as a photo for continuous updates. A possible circumvent is to screenshot every table cell, which may cost more resources especially when the table is large.

4.5.4 Summary

Here, we explain the details of two approaches to load external web resources: one with extracting original CSS styles and the other is capturing screenshots. The former method achieves a fine-grained accurate control of styles, though sometimes prevented by the external web page because of web security consideration. The latter one is much simpler, nevertheless, it can only manage a coarse-grained reproduction of styles. Providing the strengths and weaknesses of these two approaches, we offer the two options in our Chrome extension, letting the users decide which one may be more appropriate for a specific page.

4.6 Related Work

As continuous obtain of specific information from a dynamic web page is a common demand especially in web crawlers, there has been lots of research work in this direction. One early work by Brewington and Gybenko [16] explained the dynamic feature of many web contents and tried to model the change characteristics on these web pages. Later on, more work was introduced to support the continuous query on the dynamic web, including [84], [85], and [108]. For example, in [108], the authors partitioned the crawling task into the tracking phase, resource allocation phase, scheduling phase and monitoring phase. They also proposed Continuous Adaptive Monitoring (CAM) to efficiently monitor the changes.

At the same time, some researchers tried to figure out how frequently does the web change. Because the majority of web pages change rarely, we do not need to

spend too many resources to crawl the same contents. Cho et al. [23] conducted analysis on more than half a million web pages over 4 months, and they claimed that web pages change rapidly, though the actual rates vary from site to site. Fetterly et al. [38] implemented the same experiments that they crawled 150,836,209 web pages one every week, lasting for 11 weeks. They reckoned that the past changes of a web page are a good indicator of future changes. Besides the analysis of web changes, Ntoulas [104] studied the evolution of the web in one year and discussed some features which may potentially influence the design of web crawlers.

With this prior knowledge, researchers began to design more efficient incremental web crawlers. For example, Sharma et al. [126] introduced a self-adjusting architecture for web crawlers. They created an algorithm to dynamically calculate the refresh time of pages and selectively update its database. Kim et al. [62] proposed a similar system, where they determine the crawling cycle time based on current collection time, collection time for the previous update, and average collection time.

Compared with all the work mentioned below using a pull-based approach, where the web crawler actively crawls target pages periodically, there is some work trying to use a push-based approach in web crawlers. For example, Mesbah et al. [90, 91] proposed a novel method to crawl AJAX-based applications through dynamic analysis of user interface state changes.

4.7 Conclusion

This chapter introduced the concept of continuous updates of external web resources, which is quite common in a presentation. We discussed the approaches to monitor the changes, especially on whether using a backend server or not. How to load the external web resources on the document system via a Chrome extension was also discussed, contrasting our work to previous work related to the continuous query.

The next chapter presents a distributed context-aware collaboration framework based on our presentation document system. We explain the details of the framework methodology and architecture components. The framework can also be extended to include more types of multimedia.

5. DISTRIBUTED CONTEXT-AWARE COLLABORATION FRAMEWORK

5.1 Challenges

When implementing a collaborative multimedia presentation system, the first challenge is how to capture every media control event and enclose necessary information into a message. As shown in Figure 5.1, an abstract media event such as playing a video is usually first represented by a semantic event, understandable by human beings. As on web browsers, web pages are represented by nodes and objects in the DOM. We need to convert a media event into one or more DOM events, easily captured by JavaScript programs. The propagation of DOM events makes it complicated to capture the media event. There exist two types of event propagation: *event capturing* and *event bubbling* [152]. In the event capturing mode, the event fired is first captured and handled by the parent node, and then if possible, it will be passed to the node's children. By contrast, in the event bubbling mode, the event triggered is first captured and handled by the deepest child element, and then if possible, it will be bubbled up to the parent node.

Another challenge is about where to register event handlers to capture these media events. In Figure 5.1, to play a video, a sequence of DOM events can be fired. Not only is the *mouse click* event fired, but the *play* event of the video is also triggered.

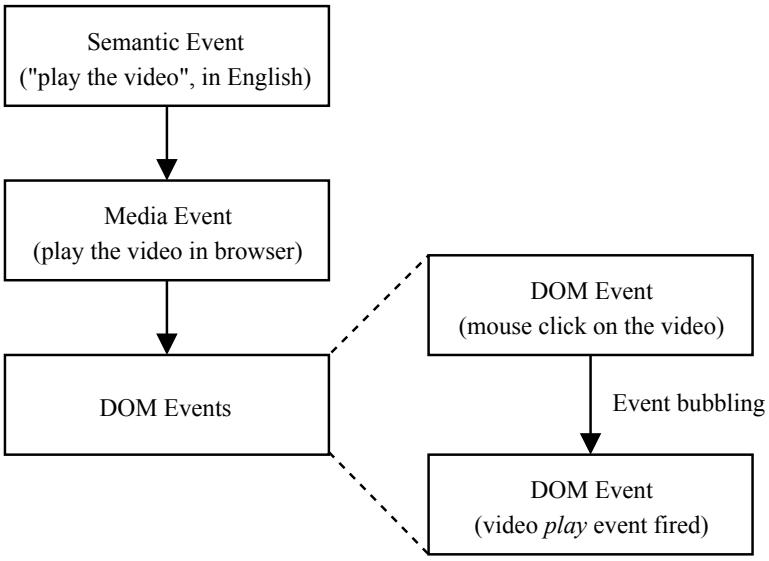


Fig. 5.1.: Conversion of a semantic event to media event, and then DOM events.

One straightforward way is to directly utilize the browser media APIs to monitor these media events such as *canplay*, *progress*, and *timeupdate*. Because most of the vanilla browser media APIs only provide low-level controls and the implementations of these APIs are variant on different browsers, currently there are a large number of popular third-party libraries such as audio.js [65], caman.js [71] and video.js [148], wrapping these browser media APIs, realizing more useful utilities, and providing more unified and elegant interfaces. For example, by April 2019, video.js, a JavaScript-based open source HTML5 video player, has more than 20,000 starts of its GitHub repository and is used on over 400,000 websites. These libraries hide the lower level browser media APIs from the developers, making it arduous to register additional handlers to monitor the original media DOM events. Moreover, some types of media, such as PDF documents, do not have direct browser APIs for necessary controls, and thus

developers have to turn to some third-party libraries, like pdf.js [101], a PDF reader library developed by Mozilla Foundation, for help.

Another possible place to capture media events is low-level mouse events such as *mousemove*, *mousedown* and *mouseup*. However, this approach leads to another challenge of replaying the events accurately on remote devices with the increasing popularity of mobile devices, many websites have applied a new design paradigm named *responsive web design*. In this style, a web page can be automatically adjusted on various devices including desktops, tablets, and mobile devices, with variant screen sizes. An example of responsive web design built with a UI library, Bootstrap, is shown in Figure 6.3. Here, one web page, *getbootstrap.com/docs/3.3*, has different layouts on a MacBook Air and an iPhone 8. The website layout is adapted by fluid grid-system to specific screen sizes. Because elements may be in different positions, the position-based synchronization approach, widely used in collaborative whiteboard tools such as AWW [1], cannot be directly applied here. Although a mouse click at one specific position can open a new link, a click at the same position may not successfully repeat the event on another display.

The final challenge is how to support general multimedia. A system can be very beneficial if it supports various types of media for interaction and collaboration. The challenge here is to design a general-purpose collaboration and synchronization protocol for multimedia, that developers can follow to support different kinds of media for efficient and accurate collaboration. Nonetheless, our study of four prior collaboration systems indiciates that they only support very limited types of media, as demonstrated in Table 5.1. Here, Google Docs supports document editing, Spread-

Table 5.1: Comparison of supports for multimedia collaboration of four studies with our work.

	Document editing	Web browsing	Whiteboard	Image annotation	Video watching
Google Doc	✓	✗	✗	✗	✗
SpreadVector [37]	✗	✓	✗	✗	✗
Collaboard [66]	✗	✗	✓	✗	✗
Kim et al. [63]	✗	✗	✓	✓	✗
Our work	✗*	✓	✓	✓	✓

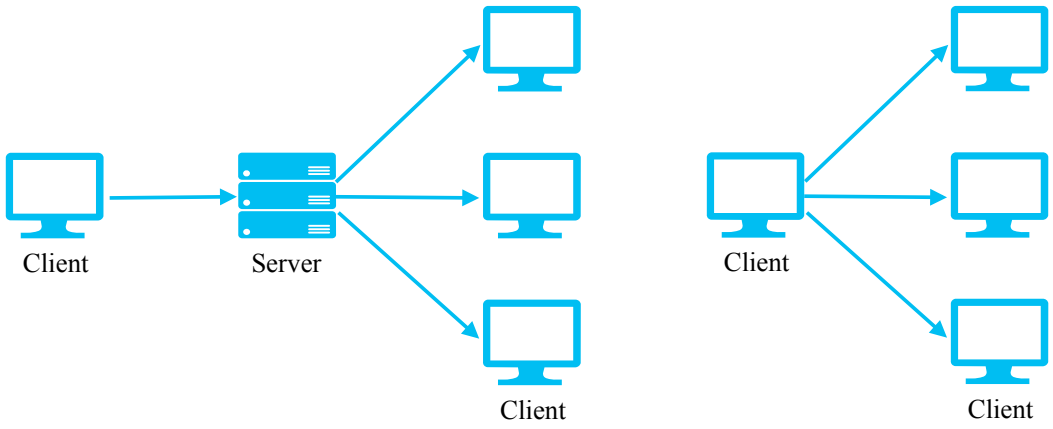
Vector [37] is for collaborative web browsing, and Collaboard [66] has a whiteboard functionality. The system described in [63] works for more types of media, even though it still only supports whiteboard and image editing. By contrast, our collaborative platform supports all types of multimedia listed in the table, especially for presentation purposes. Note that even though our system does not support collaborative document editing, it does support collaborative annotation and basic controls on PDF documents.

5.2 Framework Methodology

5.2.1 Communication Models

In modern collaboration system design, there are mainly two different models: client-server and peer-to-peer. In the client-server model, a server sits between the presenter and other collaborators. A collaborative event is sent to the server first, and afterward, the server broadcasts the message to other clients. This model separates the user interfaces from the data storage/management and thus, easily achieves scalability on the server side. Currently, some collaboration systems based on cloud

services such as Firebase [39], Cloudkit [129], and Simba [111] fall in this category. With a central service, not only a client sends requests to the server, but also the server needs to push updates to some targeted clients in a collaboration session. It is like a client subscribing to a stream of updates with some predefined specific conditions. Whenever a new event occurs, a notification is sent to the client. To achieve that mechanism, up until now, there have been four major approaches: polling with AJAX (Asynchronous JavaScript and XML) requests, long-polling, server-sent events, and WebSocket.

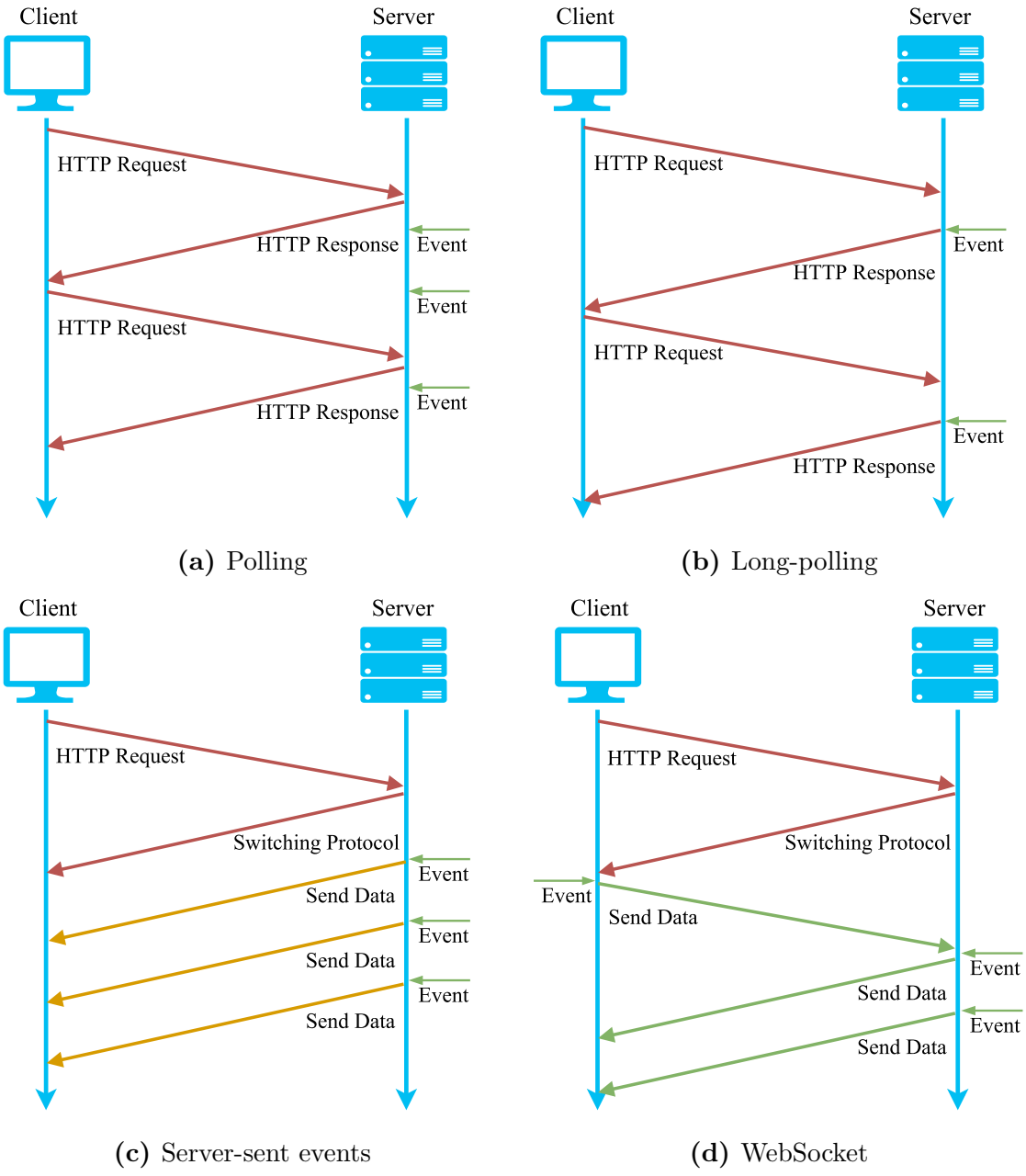


(a) Client-server model. **(b)** Peer-to-peer model.

Fig. 5.2.: Two different communication models.

In our work, to achieve the uniform interfaces of collaboration, our distributed collaboration framework does not restrict to a specific model. Because most of the current systems apply a client-server model for the reasons of scalability and manageability, we concentrate on this model in the remaining parts of this section.

- Polling with AJAX requests. This model follows the basic principles in HTTP communication, that a client continuously sends requests to the server, and the



(a) Polling **(b)** Long-polling

(c) Server-sent events **(d)** WebSocket

Fig. 5.3.: Comparison of four server pushing models.

server endlessly posts responses whether there exists a new event or not, leading to an issue that some bandwidth and computation resources are wasted.

- Long-polling. This is an improved version over polling. In this paradigm, the server holds a persistent HTTP connection and does not send a response until a new event is fired. Although this model reduces the waste of resources, the server still needs to keep the connection open without serving other clients. It could be worse especially when events are taking place infrequently.

- Server-sent events. Server-sent events (SSE) is a technology born from the HTML5 standard. Currently, most of the modern web browsers have supported it. A client first sends a request to the server to ask for a protocol switch. If the server supports server-sent events, it will send back a response, switching to EventSource [156] interfaces. Afterward, it is convenient and efficient for the server to push a stream of updates to clients. However, an issue preventing server-sent events from being widely used in collaboration tools is that the communication is uni-directional – the client cannot push data to the server afterward.

- WebSocket. Similar to server-sent events, the WebSocket API [154] and protocol [36] were also released in the HTML5 standard. The WebSocket protocol provides bidirectional, full-duplex communication channels over a single socket connection [87]. Figure 5.3d shows the procedure of the communication between the client and the server. At first, a client sends a request to the server, asking for a protocol switch. After the server receives the request, and if it supports

Table 5.2: Compatibilities of XMLHttpRequest, Server-sent events, and WebSocket on major web browsers. [1]

Browser	XMLHttpRequest	Server-sent events	WebSocket
IE 11	✓*	✗	✓
Edge 18	✓	✗	✓
Firefox 66	✓	✓	✓
Chrome 74	✓	✓	✓
Safari 12.1	✓	✓	✓
iOS Safari 12.2	✓	✓	✓
Chrome for Android 74	✓	✓	✓

WebSocket, it will switch to the WebSocket protocol. Afterward, the bidirectional channel is established, and each side can push data to the other side. According to the experiments performed by Pimentel et al. [114], WebSocket has an obvious lower latency than polling and long polling.

We have also studied the compatibilities of these four methods on browsers, and list the results in Table 5.2. XMLHttpRequest (the foundation of polling and long-polling) and WebSocket have the best compatibility in current major web browsers, and server-sent events are supported in most of them, except Microsoft IE and Edge. Because of the advantages of compatibility and performance, we recommend using WebSocket in a web-based collaboration platform when choosing the client-server model.

By contrast, in the peer-to-peer model, no servers are required. The presenter directly broadcasts messages to other attendees in this session. As the computation of devices is becoming more and more powerful, some researchers tried to employ a peer-to-peer model in a collaboration system. One recent work is Legion [146], which

[1] All browsers are compared with their current versions, by May 2019. Data is fetched from https://caniuse.com/. *: partially supported.

utilizes WebRTC [9] protocol to achieve collaboration among peers without involving servers.

In our work, to achieve uniform interfaces of collaboration, our distributed collaboration framework does not restrict to a specific model. However, because most of the current collaboration systems apply the client-server model for the reasons of scalability and manageability, we concentrate on this model in the remaining parts of this chapter. This is also the model used in our collaboration platform.

5.2.2 Stateless Events

In our design, all collaborative events occurring in a session are stateless, without containing any previous context information. This means that every event needs to be self-descriptive, wrapping all necessary information to represent the media event that takes place, and is able to be replayed precisely.

Note that in a web-based collaborative work, it is common that a client may ask for a resynchronization of the current media state because of lost network connection or late attendance. In that case, an event with the current media state needs to be broadcast to the target client. Here, this event is not contradictory to our stateless events requirement because the event to synchronize the current media state does not depend on any stored context in previous events. This is just an event with essential media state information to reconstruct current collaboration progress.

Similar to Martin Fowler's event sourcing [40], which describes that all changes to the application state are stored and can be used to reconstruct the past states,

a collaborative session consists of a sequence of media events. However, for the restoration of states, besides executing the events sequentially, we allow the direct capture of the current media state to improve efficiency in re-synchronization, as mentioned above.

The stateless events requirement introduces reliability, security, and scalability in a distributed collaboration system. Reliability is improved because the server only stores the events instead of states, making it easier for crash recovery. Additionally, only media events are transmitted, and thus very little content-related information is exposed to the outside. For instance, when a client presses a button to play a video, only the *button click* action is captured and broadcast, without unveiling any screenshots or frames of the video. Furthermore, without storing application states, it is easier to achieve high scalability on the server side. The details of the design of a scalable cloud service for collaboration are discussed in the next section.

5.2.3 Scalable Cloud Service

With the scalability provided by our design, it is possible to create a scalable cloud service to serve multimedia collaboration functionalities. Currently, there are some cloud services providing real-time updates, which is similar to real-time collaboration. For example, the Parse [109] server supports live queries where clients can subscribe to updates of objects with specific conditions. Here, based on the prior work in infrastructure, we propose a possible architecture for distributed collaboration on a cloud service, shown in Figure 5.4.

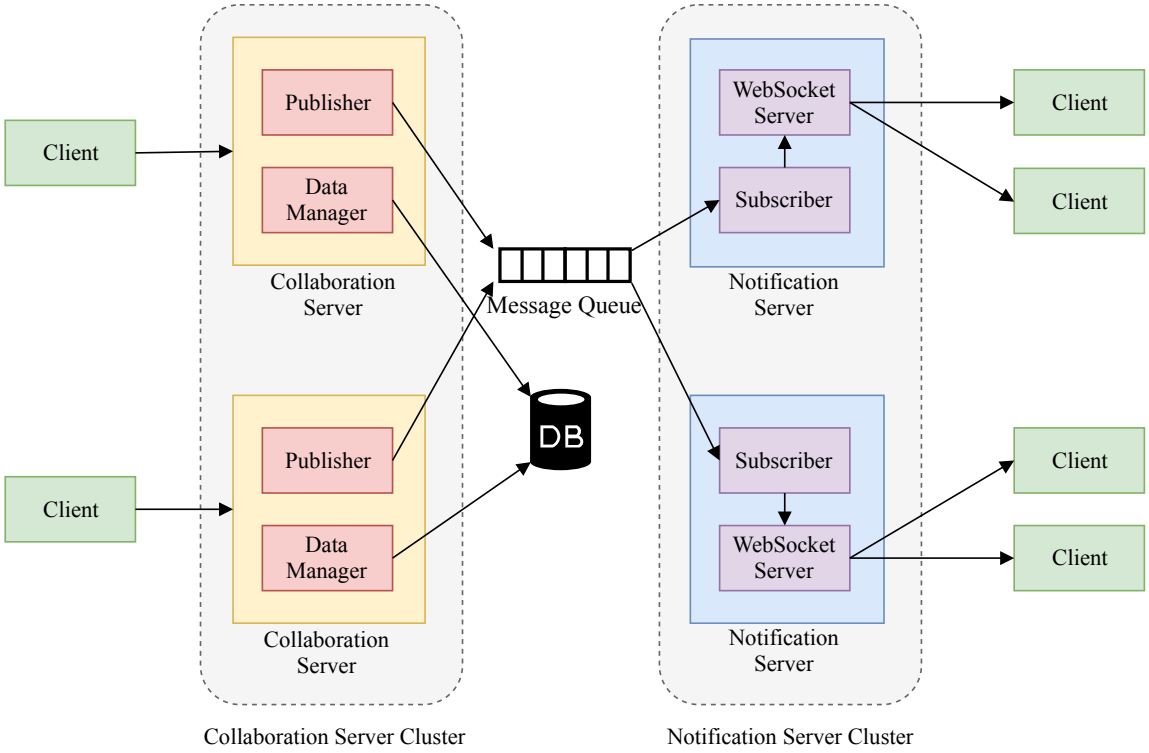


Fig. 5.4.: A possible architecture for distributed collaboration through a scalable cloud service.

Here, a key point to ensure the scalability is to separate the collaboration servers from the notification servers. The responsibilities of a collaboration server include receiving events from clients, publishing events into a message queue, and storing events to the database. The message broker works as the bridge between the collaboration server cluster and the notification server cluster. After the publisher module of a collaboration server pushes an event into the queue, a subscriber of a notification server will pull the event from the queue and send the event to target clients via a WebSocket server. Database instances are also necessary for the cloud service, especially when users would like to store collaborative sessions. The data manager module in a collaboration server takes charge of managing events stored in the database. In our

design, the specific architecture of a collaboration server is not restricted; developers can use layered, event-driven, or microservices architectures to implement the server.

5.2.4 Uniform Interfaces

A very crucial feature that distinguishes our design from other collaboration systems is that we provide uniform interfaces for distributed collaboration on various types of media. As collaboration information is transmitted in a standardized message format, it relieves the difficulty to insert a new type of media to the system. A possible structure of collaboration, based on a uniform interface, is shown in Figure 5.5.

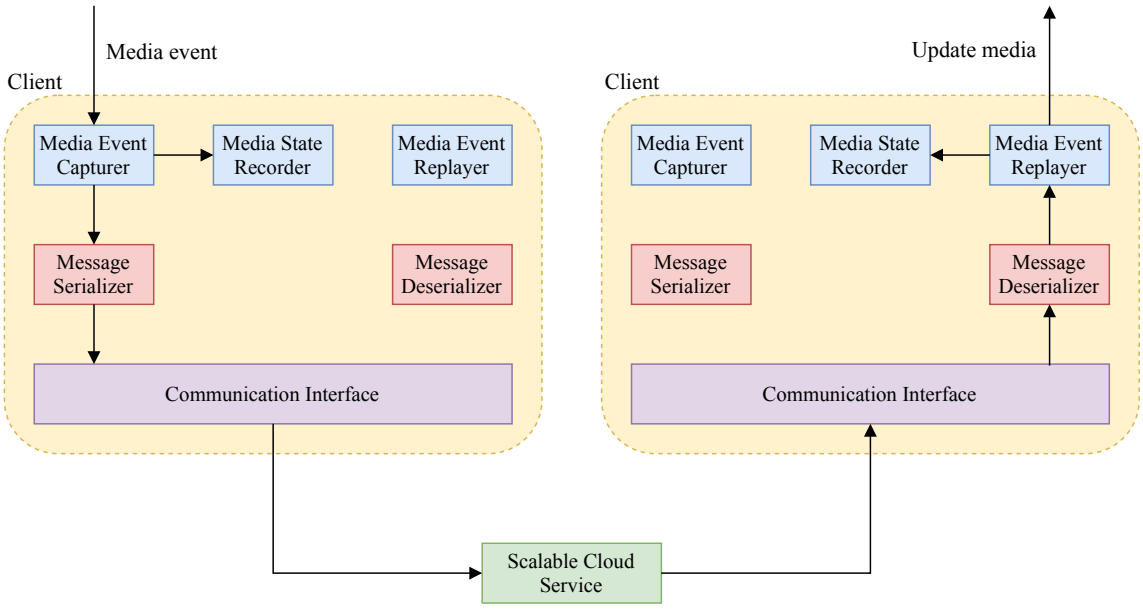


Fig. 5.5.: A possible implementation of collaboration based on an uniform interface.

Here, for the client who initiates a media event for collaboration, there is a *media event capturer* module to monitor and capture the event. A *media state recorder*

may be necessary to record the current media state for possible resynchronization. For example, the state of a video may include the play/pause state, volume, progress, playback rate, related annotations, etc. At the same time, the event is sent to a *message serializer*, which encapsulates the event into a standardized message and sends it to the backend service. After the message is routed to one target client via the backend service, a *message deserializer* takes charge of unwrapping the message and sending information to the *media event replayer* to update the current media state. Simultaneously, the change of media state may also be recorded in the m*edia state recorder* module.

With the uniform interfaces, if a developer would like to add a collaboration mechanism for a new type of media, he/she just needs to follow the structure by adding functions in the *media event capturer*, *media state recorder*, and *media event replayer*. *Message serializer* and *message deserializer* should be left intact if the format of transmitted messages is not changed. The communication is handled by the architecture, and the developer does not modify the communication interface to support new types of media. The feature of extendability significantly relieves the burden of implementing new functionalities for collaboration.

5.2.5 Object-Prioritized Collaboration

A common problem in distributed collaboration is the precise synchronization of events on various displays. For object-prioritized collaboration constraint, we restrict the synchronization to **object-based** as much as possible, instead of position-based.

In that case, for a media event, we record the object adhered to the event. This provides much more flexibility than the traditional position-based approach, as no matter where a media event fires, we only trace the object affected, isolated from the influence of positions. It is especially superior considering that currently lots of websites have applied responsive web design styles, so that there may be very different layouts on laptops and mobile devices.

We admit that not all media events can be represented as object-based. For example, a user may move an image via mouse moving events. And moving events, actually, have no objects binding. To solve this issue, we use a **proportion-based** synchronization approach that controls the aspect ratios of the target image and canvas underneath on various platforms, as shown in Figure 5.6. Here, on either monitor or mobile device, the aspect ratio of the canvas is always 1:2, and the aspect ratio of the image is always 1:1. Thus, when a user moves the image on the monitor, say to the center of the monitor, we record the new position of the image on the canvas proportionally and compute the new position of the image relatively on the mobile device precisely.

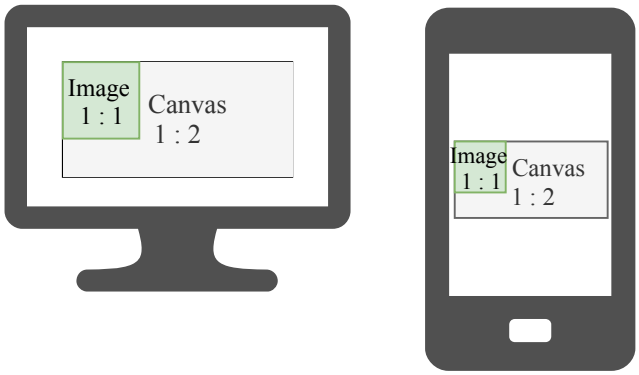


Fig. 5.6.: Proportion-based synchronization of an image on various displays.

Another scenario is related to using mouse scrolling to zoom in/out an image, a PDF document, or a map. The mouse scroll event can be captured via *wheel*, *mousewheel*, or *DOMMouseScroll* event, according to the specific implementations of the browser. With the event fired, usually, there is a *delta* value to represent how much the mouse has scrolled. With the value, we may also use a predefined scale factor to determine how much we want to zoom in/out. Here, there is no object attached because this is a mouse event. Also, no proportion or position is necessary; we just employ the value given to achieve synchronization. This can be regarded as a **value-based** approach.

In general, we employ a hybrid synchronization approach, with a combination of object-based, proportion-based and value-based methods. The details of using these synchronization methods in our system are demonstrated in Table 5.5 in Section 5.3.3. In summary, object-based is always preferential. Proportion-based and valued-based are considered as supplementary methods, only used when there are no direct objects related.

5.2.6 Non-Intrusive Collaboration

This constraint is very crucial in ensuring the extendability of our system. Extendability here refers to the capability of extending the collaboration to other types of media. Because the system is non-intrusive, a developer does not need to modify any code underneath related to the media, needing only to create another controlling layer on top of the media APIs. This feature is very beneficial when the original

media functionalities are implemented through a third party library instead of vanilla browser media APIs. An example is achieving collaboration based on a JavaScript image cropper, Cropper.js [19]. Suppose that there has already been an image cropper application implemented with Cropper.js. Afterward, developers plan to add a collaboration mechanism onto the application. With non-intrusive design, the developers do not need to modify any code of Cropper.js. Instead, they can follow our uniform interfaces, create a controlling layer on top of the library, and utilize the events provided by the library. For instance, the monitoring of a series of *cropstart*, *cropmove*, and *cropend* events gives the path of a successful crop operation. These events can be synchronized on other displays to reflect the crop operation. The implementation of non-intrusive collaboration in our platform is discussed in Section 5.3.2.

5.2.7 Access Control

Access control is crucial in protecting the security of a distributed collaborative web service. For example, in a collaborative presentation session, only the presenter is allowed to manipulate the media objects and broadcast the media events fired to other attendees. According to [135], access control policies can be categorized as **discretionary access control (DAC)**, **mandatory access control (MAC)**, **role-based access control (RBAC)** and **attribute-based access control (ABAC)**.

In a traditional DAC system, authorities are based on the identities of the requestors and some specific access rules instructing what the requestors can access and what they cannot. The word *discretionary* means that a subject could pass its

permissions to any other subjects. In practice, DAC is usually implemented in an access control matrix or an access control list, as illustrated in Table 5.3 and Figure 5.7 respectively. The access control matrix shows an example of controlling read, write and execute privileges among user 1, 2, and 3. The access control matrix is usually sparse, we could convert the matrix into a linked list as an access control list. one critical issue of this model which prevents it from wide usage in collaboration systems is that it lacks flexibility. When a user's permissions are changed, which is quite common in web services, the access control matrix(list) needs to be continuously updated.

Table 5.3: Access control matrix.

	File 1	File 2	File 3
User 1	Read Write Execute	Read	Read Write
User 2	Read	Write	Read
User 3	Read Write	Read	Read

A MAC system applies another policy by assigning access permissions from a central authority; that's why it is called *mandatory*. The system is usually related to security labels that indicate how sensitive the objects. Thus, it is widely used in military or national security systems.

In an RBAC system, the access permissions are related to roles in the system. A user is granted with one or more roles to get access to some resources. For example, in a database system, the administrator can have both read and write access to the contents in the database, while normal users can only read from the database. There

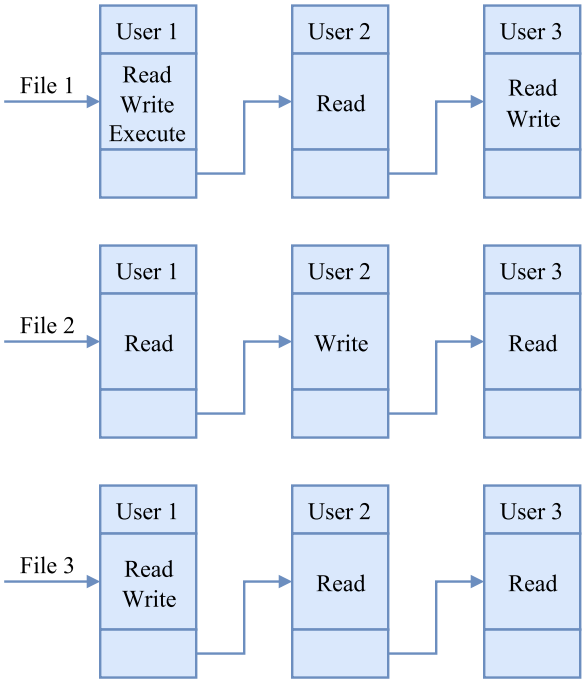


Fig. 5.7.: Access control list.

has been some pioneering work applying RBAC to collaboration environments, such as [74], [110] and [178]. Although in some work like [176] and [142], the researchers criticized that RBAC still lacks the flexibility to specify a fine-grained control of the access permissions in a complicated collaboration system.

Recently, with the rapid development of web services, ABAC has become popular. The attributes, including subject attributes, object attributes, and environment attributes, define specific characteristics used in access control. It is currently used in web-based systems, like [175], [128] and [127]. They claimed that ABAC provides flexibility in systems where access permissions are susceptible to changes.

To our knowledge, the role-based access control (RBAC) [120] model seems to be the most appropriate model for our system, especially considering that it is very natural to have various roles for users in a collaboration session – although our framework

is not restricted to a specific access control model. Take our collaboration platform as an example, we create an RBAC model, shown in a matrix representation in Table 5.4, to represent the access control. There are three roles in our model: *administrator*, *presenter*, and *listener*. By default, the owner of a presentation is assigned as the administrator automatically. The major duty of this role is to grant a presenter, who can be the administrator himself/herself. The presenter is the user who controls the events that take place in the session and broadcast them to other attendees. Moreover, the presenter can also resync the current state if a resync request is sent from other users. In a session, at one time, there is only one presenter, whereas, in the whole presentation, there can be multiple presenters. Additionally, a listener can apply to be the presenter, and the administrator decides whether the role needs to be transferred or not.

Table 5.4: Access control matrix representation of our web-based collaborative document system.

	Assign a presenter	Sync media events	Resync	Ask to be presenter	Ask for a resync
Admin	✓	✗	✗	✓	✓
Presenter	✗	✓	✓	–	–
Listener	✗	✗	✗	✓	✓

Note that we only show an example of roles assigned in our application. In fact, our framework has no restrictions on the roles in a distributed collaboration platform. Thus, a developer can assign other roles necessary for his/her specific demands.

We also add an additional constraint in access control to ensure the permissions are strictly followed according to the access control model we propose, by creating

authorization mechanisms on both the client and server side. Before a client sends a media event, the client-side JavaScript code scrutinizes the role of the user to check whether the user has enough authority to synchronize the event. To prevent possible security issues caused by client-side hacking, the server also keeps another authorization layer to ensure that the synchronization action is allowed, and that the user has the role he/she claims. The event will not be broadcast to other target clients until all the checks are passed.

An implementation example of access control in both client and server sides is shown in Figure 5.8. The information of the current access control model is stored in both client and server sides' storages. In that case, each side maintains a DBAdapter to connect to a specific database determined by developers. For example, on the server side, the model may be stored in MySQL, MongoDB, or Redis. Note that the model may be updated continuously. Suppose that the administrator assigns another presenter. The first check is executed on the client side. After the check is passed, a request is sent to the server. On the server side, after the security check is passed, the model stored in a database instance is updated to reflect the change of the presenter. Afterward, the update is sent to all clients in this session to update their local storages.

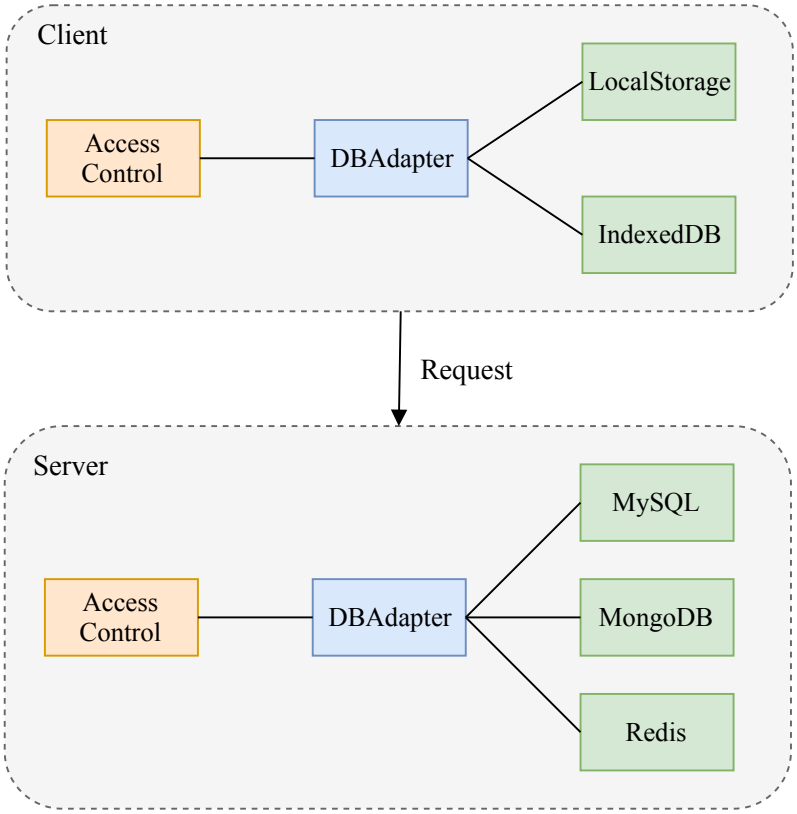


Fig. 5.8.: Storage of access control models.

5.3 Architecture Components

5.3.1 Collaboration Subject

A collaboration subject is an attendee in a collaboration session. A collaboration subject's functionalities are usually reflected by its agents, for example, web browsers. We categorize a collaboration subject to be active or passive. An active collaboration subject, considered as a sender, actively controls the collaboration artifacts and broadcasts updates to other subjects. In a presentation scenario, an active collaboration subject is a presenter. By contrast, a passive collaboration subject, regarded as a recipient, passively receives messages and updates media states. Take Figure 5.5

as an example. The client on the left side is an active collaboration subject, and the client on the right side is a passive collaboration subject.

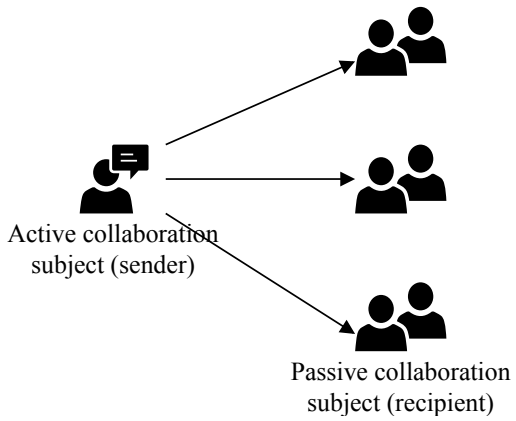


Fig. 5.9.: Collaboration subjects.

The role of a collaboration subject is not immutable. At a time spot, an active collaboration subject can be updated to become a passive one, and a passive collaboration subject can be changed to become an active one, taking charge of broadcasting media events.

5.3.2 Media Event Capturer

The main functionality of a media event capturer component is to capture an abstract media event via necessary DOM events. In our design, by using event propagation, especially event bubbling, which is now supported in all of the modern browsers, we create another layer on top of original media to capture these media events. The controlling layer contains UI components, and the target media events are converted to events fired on these components. This is possible because every control event of a media object on a web page must be triggered by a DOM event on that page.

For example, to play a video, we may need to click a button (or the video itself) to trigger the action. To zoom in an image, we may scroll up the mouse or click a button to achieve the expected effect. As a result, we create a controlling layer to capture the DOM events on top of original media objects. In Figure 5.10, we illustrate the structure of capturing video events in DOM events.

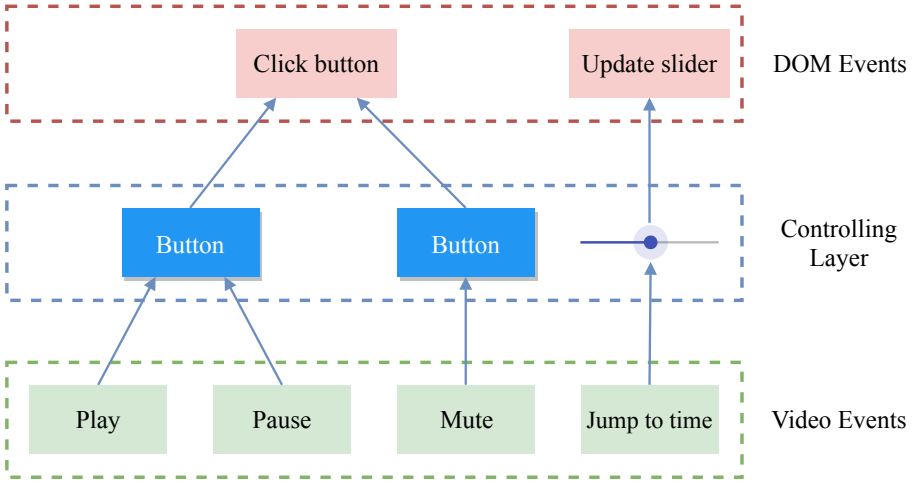


Fig. 5.10.: Structure of capturing some video events in DOM events.

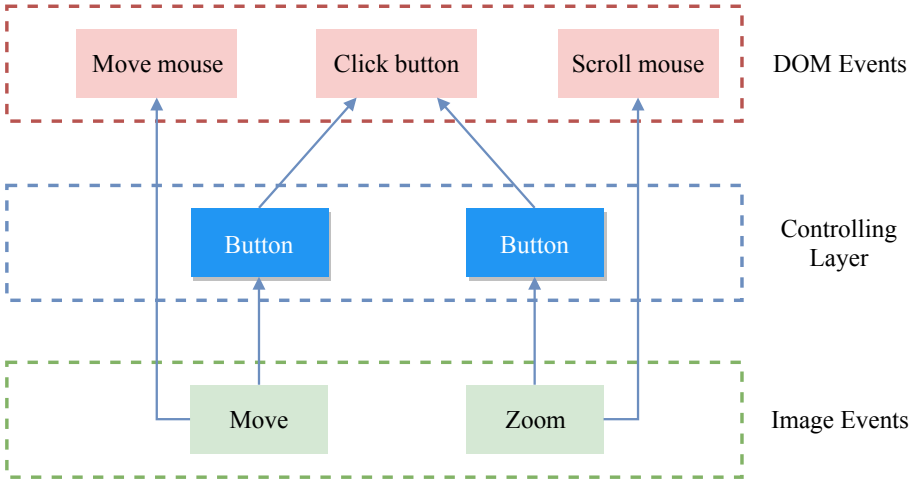


Fig. 5.11.: Structure of capturing some image events in DOM events.

In the figure, there are four controls of a video: *play*, *pause*, *mute* and *jump to time*. Every control is triggered through a UI component. For example, the *play/pause* event is toggled by clicking a button, and *jump to time* event is activated by dragging or clicking a slider or progress bar. As a consequence, every media event can be represented and captured by a corresponding DOM event. For instance, muting a video can be recorded by a click on a specific button. Providing that our framework is object-prioritized, without touching the media frames, it is very flexible and works well on various platforms.

However, it is not always true that a media event has to be controlled by a UI component. In fact, we do not have such controlling constraint in our design. One typical example is shown in Figure 5.11, related to capturing image events. Here, when a user moves an image, he/she can not only control the media via clicking a button but also achieve the same effect by dragging the image, which does not require any extra UI components. And because there are no UI components involved, we only need to directly capture the DOM events underneath. This also indicates that a media event can be captured through multiple DOM events. Thus, this is not a one-to-one, but a one-to-many relationship. Another typical example is Google maps. Some plain browser media APIs are quite low-level, and because of wide usages of third party libraries, some browser APIs are hidden from these libraries. However, for Google Maps, the case is different. Google Maps is a type of media without any native browser APIs. Additionally, Google officially provides a large amount of APIs to control maps embedded in web browsers. In that case, we register event handlers on the exposed public events. For example, for a move operation on a map, instead

Table 5.5: Summary of media events and related represented DOM events.

	Semantic event	Captured event	Synchronization
Video	play/pause video	button click	Object-based
	stop video	button click	Object-based
	mute/unmute video	button click	Object-based
	jump to time	progress bar click	Object-based
	change speed	dropdown list click	Object-based
PDF	prev/next page	button click	Object-based
	jump to page	form submit	Object-based
	scroll page	mouse scrolling	Value-based
Image	zoom in/out	mouse scrolling	Value-based
		button click	Object-based
	move	mouse down/move/up	Proportion-based
	crop	mouse down/move/up	Proportion-based
Webpage	visit a provided URL	form submit	Object-based
	load a URL on page	link click	Object-based
	prev/next page	button click	Object-based
Google maps	zoom in/out	zoom_changed	Value-based
		button click	Object-based
	move	dragstart/dragend	Proportion-based
Annotation	free drawing	mouse down/move/up	Proportion-based
	highlight text	selection change	Object-based
	insert shapes	dropdown list click	Object-based

of monitoring mouse events, *dragstart* and *dragend* events, offered by the library, can be utilized to achieve collaboration.

To summarize the handling of media events captured in our system, we demonstrate the captured events related to specific media events in Table 5.5. Note that we also list annotations as a type of media, whose events like free drawing can also be captured via DOM events.

5.3.3 Media State Recorder

Although all the media events that occur in a collaboration session are stateless, and the current media state can be calculated from the prior events, under some scenarios, such as synchronizing media state to a late attendee, we may want to complete the work rapidly instead of waiting for all operations to be re-executed. That is why we recommend using a media state recorder to maintain the current state of the media. As shown in Figure 5.5, both *media state capturer* and *media state replayer* communicate with *media state recorder* to update the current state stored in it.

The state of a media block depends on the type of media, and thus different types of media have various states. Note that all the states can be represented by simple key-value pairs. For example, for a video, the state contains the video source, current timestamp, muted or not, volume, playback rate, annotations, etc. The key-value pairs can be compressed into messages for further transmission.

5.3.4 Media Event Replayer

The main functionality of a *media event replayer* is to replay events attached with complete media event information. To achieve that efficiently, we design a hierarchical tree structure, Handler Tree, to handle the messages, as shown in Figure 5.12.

An event is propagated in the handler tree via parent-child chains until a leaf node arrives. During the propagation, the received message is parsed gradually, and finally, in a leaf handler, the message is totally consumed to update UI view if necessary.

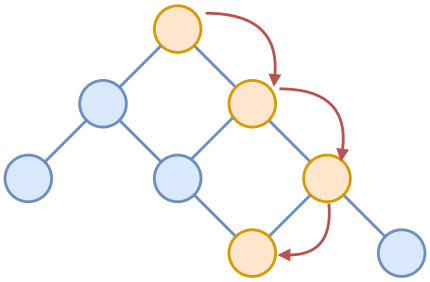


Fig. 5.12.: Propagation of event handling in a Handler Tree.

An example of using this structure in our collaboration system is demonstrated in Figure 5.13.

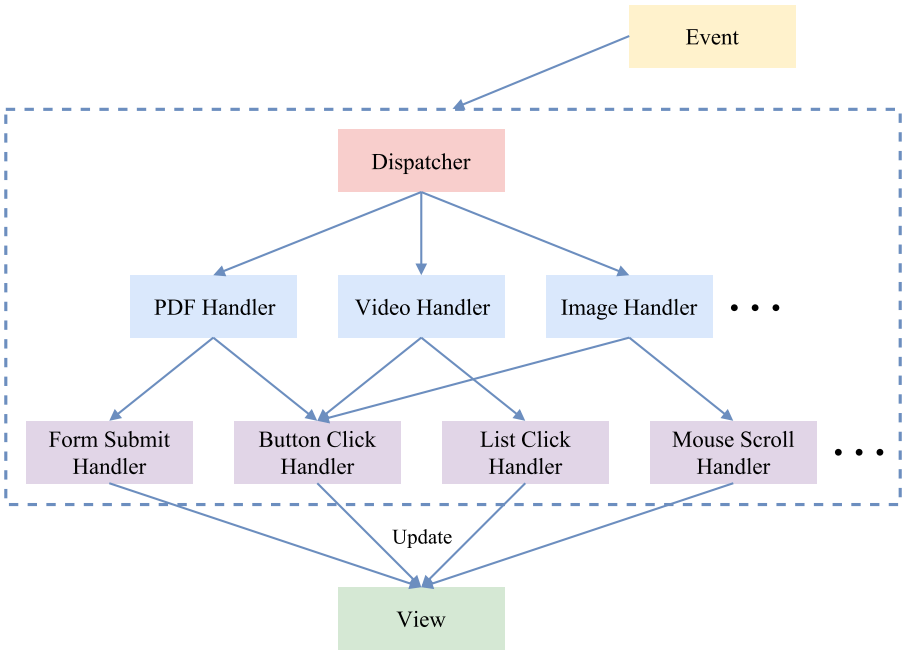


Fig. 5.13.: Propagation of event handling in our system.

Here, when an event is received by an audience, it is first sent to the root node, also acting as a dispatcher, in the handler tree. The next level in the handler tree contains various handlers for corresponding types of media. The message is sent to the appropriate media handler at this level. Then, a specific action handler is invoked

to fire the target DOM event. Take the *play/pause* event of a video described in Figure 5.15a as an instance. The message is first handled by the root handler and dispatched to the video handler according to its media type. During the handling in the video handler, because the event type is *button click*, the message is sent to the button click handler for further process. Then the handler reads the target id and triggers the click event on the element with this identifier. The handler also loads the value of current time in the data field to update the current progress of the video to 0. For various types of media, the specific action handlers may be different. Considering that our system is real-time, we design the tree to be flat to reduce the overhead.

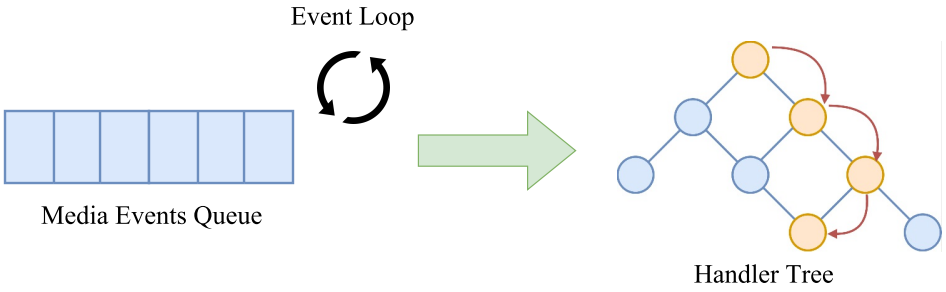


Fig. 5.14.: Replay of events.

With the help of media event replayers, the events occurring in a collaboration session can be replayed sequentially according to their timestamps, as shown in Figure 5.14. The events occurring in a session are pushed into a queue structure. A timer is scheduled to pop events from the queue and send them to the handler tree for replay.

5.3.5 Event Messages

Introduction

In our design, collaboration events are captured and represented by a standardized format of strings, and the strings are transferred between collaboration subjects. There are two different types of messages: **media event** and **control event**. A media event message represents an action that takes places related to a specific media block, such as scrolling a web page. By contrast, a control event message represents an action involving controls in a collaboration session, such as promoting a passive collaboration subject to an active one.

Media Event Message

The major skeleton of a media event message is shown in Table 5.6. The message is represented by key-value pairs.

- **Media type identifier (media-type)**. This key identifies the type of the media, such as video, audio, or image, whose event is fired.
- **Media identifier (media-id)**. This field contains the unique identifier of the target media block. It specifies the media where the event triggered.
- **Event type identifier (event-type)**. The value here is crucial for a message, as it describes the type of the event captured to represent the target media event. This is also used in other collaboration subjects to replay the media action to achieve synchronization. The specific event types included in this

field highly depend on the replaying mechanisms specified by users, and we give some more examples below. Note that for each event type, there must exist a corresponding type of each *media event replayer*. Otherwise, the control cannot be repeated on other collaboration subjects' sides.

  - *button-click*, representing the event of clicking a button.
  - *move*, representing the event of moving a media material.
  - *mouse-scroll*, representing the event of scrolling the mouse.
  - *form-submit*, representing the event to submit a specific form.
  - *highlight*, representing the event of highlighting a snippet of text or an image.

- **Sequence identifier (seq-id)**. Since the events occurring in a collaboration session are in sequential order, the events are in a happens-before relationship, namely causal dependency [86]. To maintain the order of the events, similar to the mechanism in the traditional database logging system, a unique auto-incremental sequence number is assigned to every event.

- **Timestamp (timestamp)**. This field contains the timestamp when the media event occurs and is captured. This can be useful, for example, if we want to replay all media controls based on the timeline.

- **Semantic (description)**. This is the semantic description related to the media event. It makes the event more human-readable.

```
{
  "media-type": "video",
  "media-id": "video-block",
  "event-type": "button-click",
  "seq-id": 5,
  "timestamp": 2000,
  "description": "play video",
  "data": {
    "id": "video-play",
    "current-time": 0
  }
}
```


(a) A message representing *play* event of a video.


```
{
  "media-type": "image",
  "media-id": "image-block",
  "event-type": "mouse-scroll",
  "seq-id": 8,
  "timestamp": 5000,
  "description": "zoom out image",
  "data": {
    "delta": -1.5
  }
}
```


(b) A message representing *zoom-out* event of an image.

Fig. 5.15.: Messages of a video and an image event.

- **Optional data (data)**. This field carries further information about the event, and thus it is highly implementation-specific. The information included here may be consumed by essential *media event replayers* on another collaboration subject side. For instance, for a *form-submit* event, this field contains the value to be filled and submitted, so that the media event can be reproduced precisely.

Table 5.6: Summary of elements in a media event message.

Element	Field	Example
media type identifier	media-type	image
media identifier	media-id	image-block
event type identifier	event-type	mouse-scroll
sequence identifier	seq-id	8
timestamp	timestamp	5000
semantic	description	zoom out an image
optional data	data	delta: -1.5

Based on the basic structure we have explained above, we demonstrate two message examples in Figure 5.15. In Figure 5.15a, there is a message representing a *play* event of a video. It contains all the fields we have described. In the *data* field, a piece of current time information is included to indicate the time to play or pause. In Figure 5.15b, there is a message for zooming out an image. Here, the *data* field holds a value of delta, standing for how much a user has zoomed out the image.

Our message structure is open to extension, and a developer can adhere new fields to the message for other synchronization and collaboration purposes, as long as the following rules are obeyed.

Self-descriptive. Each message should carry complete information to describe itself. This indicates that a *media event replayer* can obtain sufficient information to repeat the action, without any knowledge of prior events.

No duplicate fields. For example, when there has been a *media-type* field, it is not necessary to have another field like *media-category* that serves the same purpose.

The fields except *data* should exist in all messages. If a field can only exist in some of the messages, it should be moved to the *data* field. Take the *id* field in Figure 5.15a as an example. In a button click event, an *id* refers to the identifier of the clicked button element. Considering that not all the messages need a unique identifier to point to a web element, this value should be placed in the *data* field.

Immutable. After a message is created, it shall not be modified on the client or server side. To modify the message, we have to create a new message and discard the useless one.

Deterministic. All the values contained in the message for execution should be deterministic. That is, no randomness could exist. The execution of the message on every client should produce the same result.

Control Event Message

Besides media event messages, control event messages are also necessary to represent controlling information, such as change of authorities, resynchronization, and reorganization of panels, in a collaboration session. The major skeleton of a control event message, illustrated in Table 5.7, is quite similar to that of a media event message, except:

- **Control type identifier (control-type)**. This value identifies the type of control event, such as *resync*. This field, together with *media type identifier*, is used to judge whether a message is a media event or a control event.

- **No indispensable media-related fields**. Unlike media events, the *media type identifier* and *media identifier* are not essential any more in control events. However, some control events may still be necessary to be included in the *data* field. For example, in Table 5.7, we put *media-id* in the *data* field to point to the specific media block to resynchronize.

A developer can also add new fields to the structure of a control event message, as long as he/she follows the same rules as those of media event messages.

Table 5.7: Summary of elements in a control event message.

Element	Field	Example
control type identifier	control-type	resync
sequence identifier	seq-id	5
timestamp	timestamp	10000
semantic	description	Resync a media block
optional data	data	media-id: video-block
		media-state: state string

5.3.6 Message Serializer/Deserializer

The responsibilities of a *message serializer* and *message deserializer* are to wrap a media message into a standardized format of string and to read the message from such a string, respectively. To maintain consistency, the two components share the same message format.

Currently, there are two widely used API formats: XML and JSON. Extensible Markup Language(XML) is a markup language which defines a very flexible text format, and it plays a crucial role in the communication of web services [15]. XML uses elements, tags, and optional attributes to describe data exchanged. It is especially conducive to represent data in hierarchical structures, and during parsing, it is usually converted into a tree structure, XML Document Object Model (DOM). By contrast, JSON, short for JavaScript Object Notation, is another human-readable data exchange format consisting of name-value pairs and ordered list of values [25]. Although there is some previous work, including [107], [133], and [64], applying XML based messages, especially Extensible Messaging and Presence Protocol (XMPP) to collaboration systems, there is some other work such as [159], [88], and [79] claiming

that JSON is much faster than XML in web applications. In Figure 5.16, we demonstrate the comparison of messages of a playing event of a video in JSON and XML formats. Because of the requirement of brackets in XML messages, they may need more bytes to represent messages when compared with JSON.

```
{
  "media-type": "video",
  "media-id": "video-block",
  "event-type": "button-click",
  "seq-id": 5,
  "timestamp": 2000,
  "description": "play video",
  "data": {
    "id": "video-play",
    "current-time": 0
  }
}
```

(a) Message representing play/pause event of a video.

```
<message>
  <media-type>video</media-type>
  <media-id>video-block</media-id>
  <event-type>button-click</event-type>
  <seq-id>5</seq-id>
  <timestamp>2000</timestamp>
  <description>play video</description>
  <data>
    <id>video-play</id>
    <current-time>0</current-time>
  </data>
</message>
```

(b) A message in XML representing *play* event of a video.

Fig. 5.16.: Messages of a video event in JSON and XML.

Furthermore, binary message formats such as Protocol Buffers [147] and Apache Thrift [116] are also used in some collaboration platforms such as [52] and [11]. We do not restrict our application to a specific format, and it is easy to extend the message structure to other formats. For example, in Figure 5.17, we show how to define media events and control events in the Protocol Buffers, version proto3. Because the serialized information structure needs to be specified clearly beforehand, we put the type of optional *data* field to be a string, as we do not have prior knowledge of the precise key-value pairs in a message. This also indicates a limitation of the

binary message format. That is, the structure of every message transferred should be well-defined, leading to a weakening of flexibility.

```
message MediaEvent {
    string media_type = 1;
    string media_id = 2;
    string event_type = 3;
    uint32 seq_id = 4;
    uint32 timestamp = 5;
    string description = 6;
    string data = 7;
}
```

(a) A structure of media event messages in Protocol Buffers.

```
message ControlEvent {
    string control_type = 1;
    uint32 seq_id = 2;
    uint32 timestamp = 3;
    string description = 4;
    string data = 5;
}
```

(b) A structure of control event messages in Protocol Buffers.

Fig. 5.17.: Structures of media control event messages in Protocol Buffers.

5.4 Evaluations

5.4.1 Web Application

Based on the design principles in Section 5.2 and architecture components in Section 5.3, we created a web application to provide a collaborative presentation service, including account management, materials preparation, presentation synchronization, and replay. The graphical user interfaces are totally implemented on web browsers and have been tested on Chrome 73 and Firefox 66. The backend service, whose architecture is exactly based on the design described in Figure 5.4, is implemented in Node.js [26], which is an event-driven, non-blocking, and cross-platform JavaScript run-time environment. We use MongoDB [96], a document-oriented NoSQL database

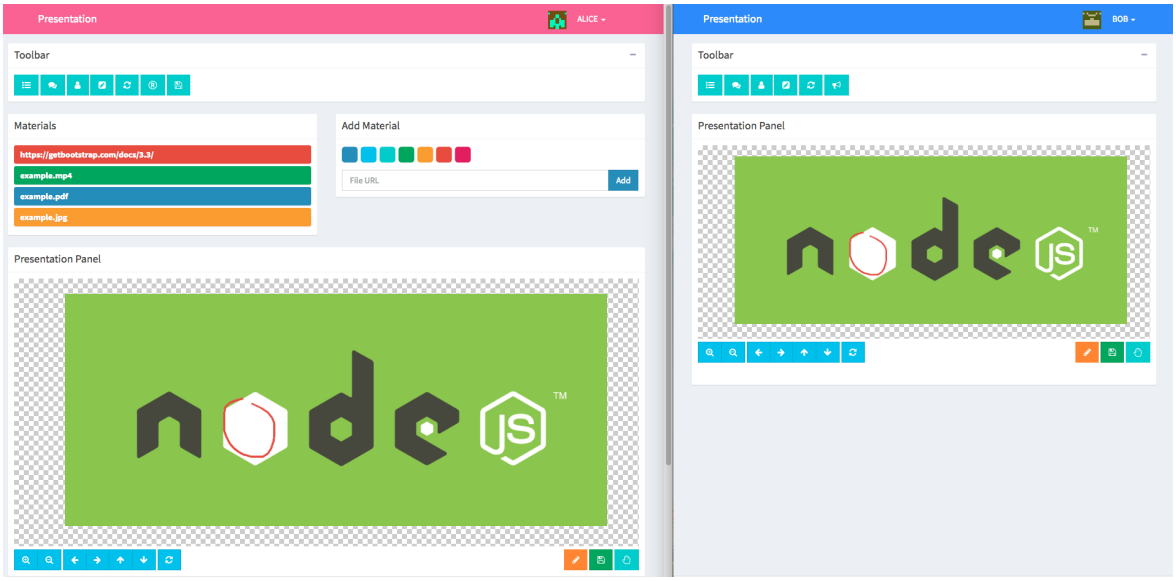


Fig. 5.18.: Two users' displays of presenting an image in one collaboration session. The presenter (also the administrator)'s windows is on the left, and a listener's is on the right. There are four major panels in a presentation web window: *Toolbar*, *Materials*, *Add Material*, and *Presentation Panel*.

to store presentation data, especially considering that MongoDB has a built-in JSON schema for documents. For the message broker, we choose Redis [121], which is an in-memory key-value database, famous for high performance. It provides a Publish/-Subscribe message paradigm.

In Figure 5.18, we show presentation windows of two users of different roles. The left presentation window belongs to the presenter (also the administrator), whose name is Alice. On the right side, there is a listener, Bob's presentation window. As illustrated in the figure, there are four major components in a web presentation window: *Toolbar*, *Materials*, *Add Material*, and *Presentation Panel*.

In the *Toolbar* block, there are useful functionalities for a presentation. For example, a user can show/hide the chatbox component for chatting, as well as the annotation bar to add annotations on a media block. A listener is allowed to send

resync request from the toolbar, and the presenter can receive the request and resync the current media state. As a result, users from different roles have various tools here. For example, in Bob's *Toolbar* block, there are no recording or saving presentation mechanisms. And on the Alice side, she cannot send a resync request, because she has already been the presenter. The *Materials* panel contains essential multimedia resources for the session. In the figure, there is a webpage (https://getbootstrap.com/docs/3.3/), a PDF document (example.pdf), a video (example.mp4), and an image (example.jpg). We also allow a presenter to add a material dynamically via the *Add Material* component. The inserted material will also be pushed into the list in the *Materials* panel. Finally, the *Presentation Panel* displays the current presenting media. The operations on the media are broadcast to other clients. The manipulations of an image include moving left/right/up/down, zooming in/out, as well as free drawing. In the example, if Alice would like to present another media material, for instance, the example.mp4 video, she just needs to drag the material from the *Materials* panel and drop it onto the *Presentation Panel*. The specific material insertion event will be automatically broadcast to all the listeners in the session, and the new material will be shown on their panels.

As an administrator, one is allowed to record the events in a presentation session into simple messages and store them in a database instance. Afterward, attendees can replay the whole session for a review. The replay is based on re-execution of events, without the help of any screen recording techniques.

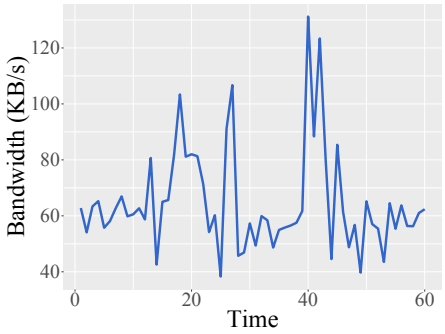
Moreover, based on the system we created, to study the performance of collaboration, we set up an AWS EC2 t2.micro instance, deployed in Oregon. Two web

browsers, Chrome and Firefox, are used on one MacBook Pro to act as the active and passive collaboration subject, respectively. We tested the time elapsed between the active collaboration subject triggering the *play* event of a video and the passive collaboration subject replaying the event. The delay is only 48 ms, which illustrates the high performance of our system.

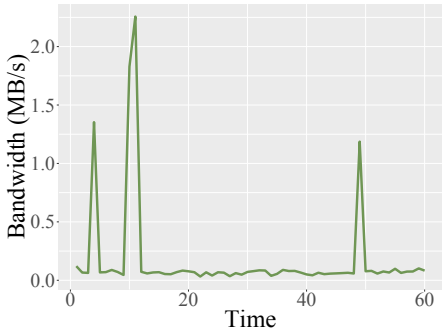
5.4.2 Comparison with Screen Sharing Tools

Currently, screen sharing products such as Microsoft Skype and Google Hangouts are widely used to provide basic collaboration support in video conferencing for presentation. In this section, we compare these screen sharing tools with our design from multiple perspectives.

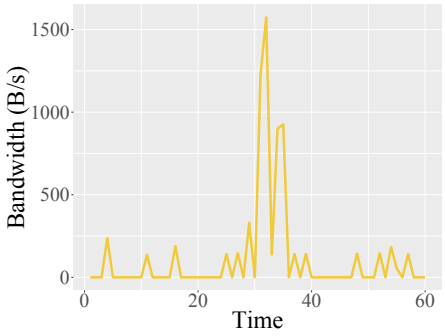
We perform a series of experiments on multimedia, including videos, PDF documents, images, and web pages, to measure the network bandwidth usages in one minute of Google Hangouts and our tool, on the presenter side. Instead of transferring video chunks during a presentation, the static materials can be loaded beforehand, and only minimal events information transferred in a session. Thus, we find that our tool requires a much lower bandwidth usage, as illustrated in Figure 5.19, 5.20, 5.21, and 5.22. Four situations are considered in these figures: Google Hangouts/our tool with preloaded materials and Google Hangouts/our tool with materials loaded dynamically after the start of the presentation. The bandwidth usages in the four cases, as well as the combination of the four scenarios, are shown in these figures. Note that



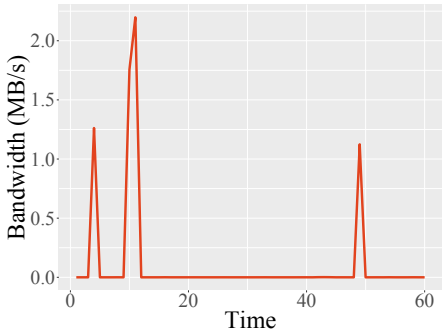
(a) Google Hangouts - preload.



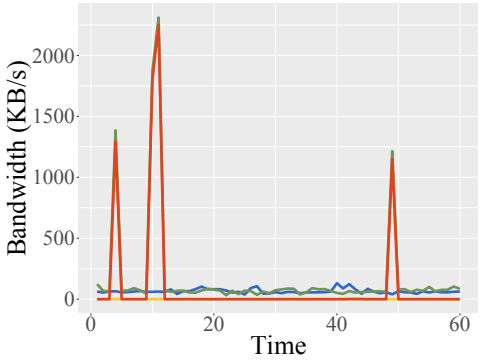
(b) Google Hangouts - afterload.



(c) Our work - preload.



(d) Our work - afterload.

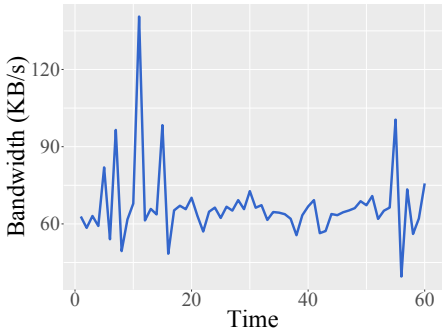


(e) Combination of previous four scenarios.

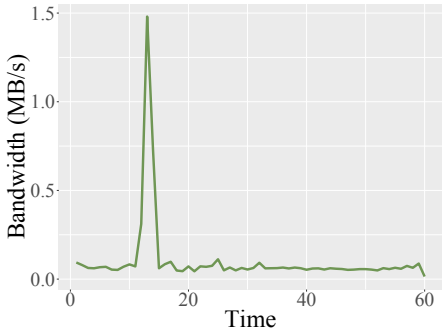
Fig. 5.19.: Comparison of networking usages when presenting a **video**, using our platform and Google Hangouts, from the presenter's perspective.

for the presentation of a web page, we do not provide a preload mode because a web page has to be loaded dynamically in a presentation.

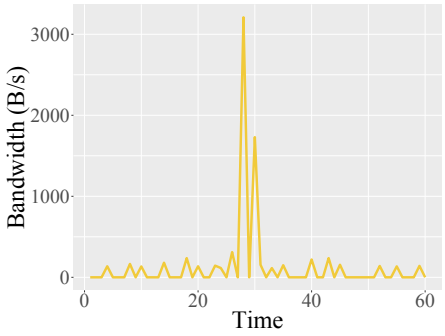
From the figures, we can see that our system has a much lower network requirement than Google Hangouts, especially in the preload mode. In the measurement in one



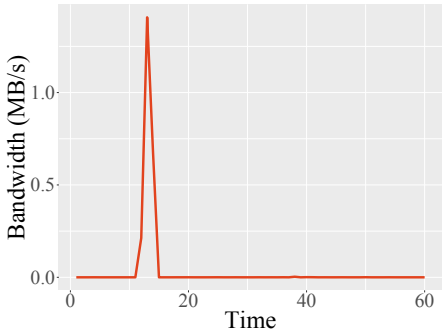
(a) Google Hangouts - preload.



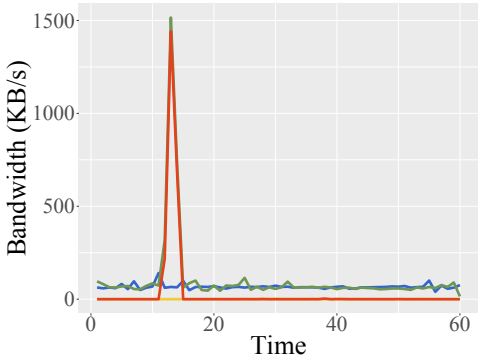
(b) Google Hangouts - afterload.



(c) Our work - preload.



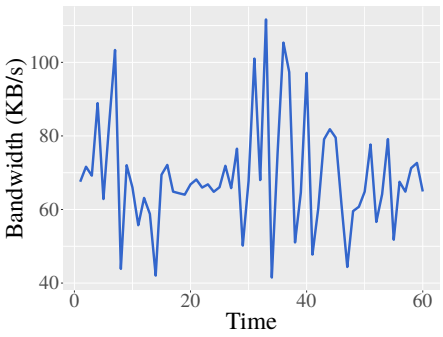
(d) Our work - afterload.



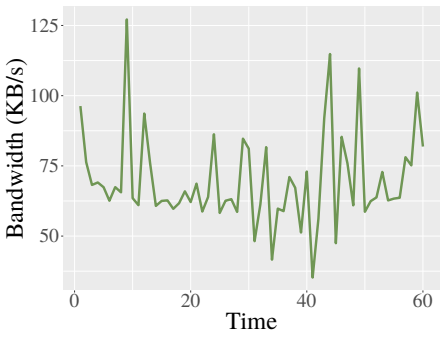
(e) Combination of previous four scenarios.

Fig. 5.20.: Comparison of networking usages when presenting a **PDF document**, using our platform and Google Hangouts, from the presenter's perspective.

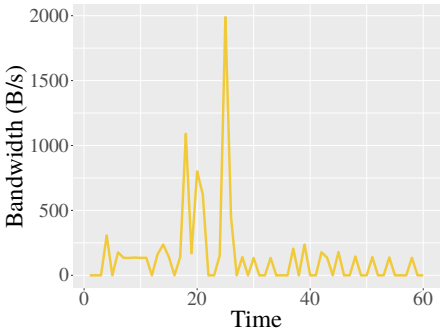
minute, for most of the events, only hundreds of bytes per second are required. Only for the free drawing events can the bandwidth usages be as high as around 3000 bytes per second (shown as some peaks in these figures), which are still much lower than those of Google Hangouts. In the afterload mode, because the target materials are



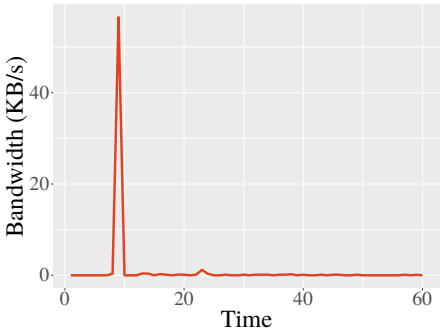
(a) Google Hangouts - preload.



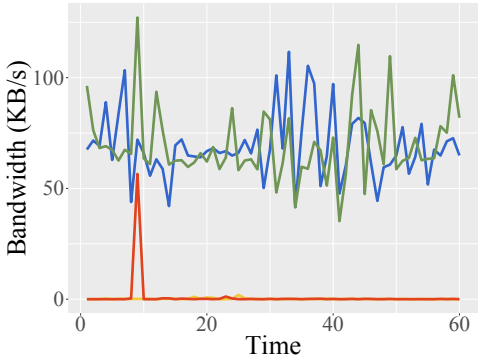
(b) Google Hangouts - afterload.



(c) Our work - preload.



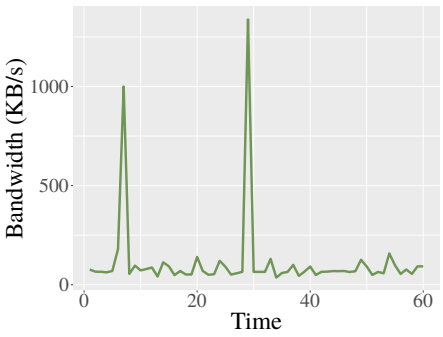
(d) Our work - afterload.



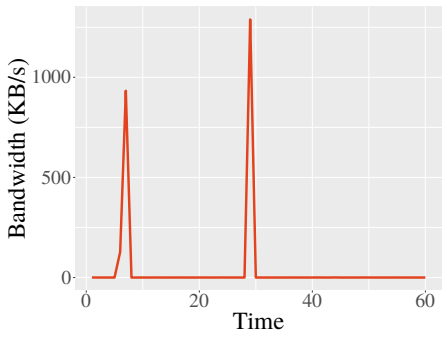
(e) Combination of previous four scenarios.

Fig. 5.21.: Comparison of networking usages when presenting an **image**, using our platform and Google Hangouts, from the presenter's perspective.

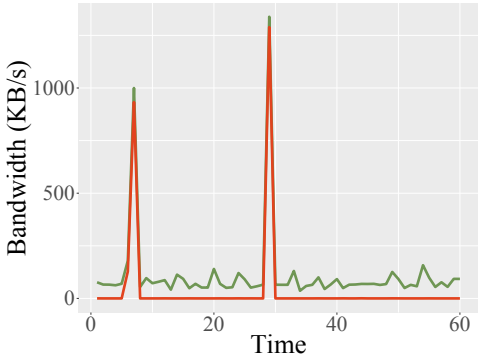
downloaded on the fly, more bandwidth usage is required in our tool, as shown in the figures on afterload mode in our tool. Because the network usages are measured on the presenter side, the bandwidth usages of loading materials are also collected in the statistics of Google Hangouts in the afterload mode. On a listener side, the usages



(a) Google Hangouts - afterload.



(b) Our work - afterload.



(c) Combination of previous two scenarios.

Fig. 5.22.: Comparison of networking usages when presenting a **web page**, using our platform and Google Hangouts, from the presenter's perspective.

should be similar to those shown in the preload mode. Still, in the afterload mode, our tool only requires higher bandwidth usage when loading necessary presentation resources. And after the loading, the usage will drop to a very low level as only limited messages are transmitted.

We also sum up the bytes transmitted in this one minute and show the result in Table 5.8. Our tool transmits very few bytes in a presentation, especially noticing that in the preload mode, in one minute, the total bytes transmitted in our tool is less than **10 KB**.

Table 5.8: Total bytes transmitted in one minute of four types of media, under the four situations.

	Video	PDF	Image	Webpage
Google Hangouts - Preload	3.80MB	4.01MB	4.04MB	-
Google Hangouts - Afterload	10.40MB	6.13MB	4.09MB	6.58MB
Our work - Preload	6.73KB	7.76KB	8.80KB	-
Our work - Afterload	6.35MB	2.30MB	62.07KB	2.30MB

Besides bandwidth usages, there are some other differences between screen sharing tools and our platform. Because the mechanism of screen sharing tools is to capture and broadcast the display, it cares nothing about what application the user is synchronizing. As a result, it is straightforward for these tools to support multimedia in web browsers, as well as a desktop environment. By contrast, our platform is totally web-based. And to support another type of media, we need to instrument necessary control to achieve the collaboration. In summary, we expect our design to be a supplement to current screen sharing applications, instead of a substitute, considering that it provides an efficient platform to support message-based collaboration on multimedia.

5.5 Related Work

Real-Time Collaboration is usually considered as a subdomain of Computer Supported Cooperative Work (CSCW). Previously, people achieved basic remote collaboration by means of video/audio calls. While it is obvious that this form of communication can only allow very preliminary cooperation. Afterward, researchers developed more complicated collaborative applications. Some of the pioneering work includes

GroupSketch [50], VideoWhiteboard [139], and Liveboard [31]. In these preliminary systems, users and their drawings are captured by cameras, transmitted and projected to remote screens. Another example is the Jazz project, developed by Cheng et at. [22]. It brings collaboration into programming bounded with Eclipse, via screen sharing. An issue of these systems is that only screenshots are transmitted, and thus they lack the flexibility to provide interaction among people geographically dispersed. Subsequently, researchers developed more elaborate native desktop collaborative applications. For instance, Booth et al. [13] proposed a "mighty mouse" multi-screen collaboration tool, which provides a smooth mouse movement cross-platform, via VNC protocol. Wu et al. [168] created the SDB (Software Design Board), which is a prototype collaborative software design tool. A crucial issue of this kind of systems is that they usually require complicated setups on different platforms because they need to have distinct implements on these platforms.

Recently, with the significant enhancement of modern web browsers, especially with the advent of technologies brought from HTML5 [151] standard, such as WebSocket [154], researchers began to consider web browsers as the backbone for real-time collaboration applications. Bentley et al. [7, 8] gave pioneering work in this direction. They produced a basic shared workspace system across web browsers. Gutwin et al. [54] performed elaborate experiments on collaboration with three approaches: XMLHttpRequest (XHR), WebSocket and Java Applet. They claimed that WebSocket has relatively high performance on various platforms including mobile devices. Similarly, Mogan et al. [94] performed a comparison study on three group-aware tools: desktop-based Java tool XCHIPS [161], browser-based Adobe Flash applica-

tion ThinkTank, and browser-based AJAX system PowerMeeting. They claimed that the last one has an advantage in features, user interface, usefulness, etc. over the others. Moreover, Pimentel et al. [114] studied WebSocket, polling, and long polling, and claimed that WebSocket has an obvious lower latency than the others.

Web-based collaboration systems can be categorized into two types, depending on every user shares one screen or each has his/her own display. Some tools fall in the former category. For example, Han et al. [56] proposed a unified XML framework for multi-device collaborative web-browsing, and Schmid et al. [125] employed this XML format to develop a web-based interactive collaborative environment. To our knowledge, most of the current web-based collaboration tools are in the latter category, and this type of collaboration has been applied to various domains. For example, Google Docs [45] and Etherpad [33] are both rather mature products which enable real-time document editing collaboration among multiple users. Fetter et al. [37] created a collaborative web browsing tool distilling the concept of lightweight interference, transitions, and adoptions. Goldman et al. [44] created a Web IDE, Collabode, for real-time collaborative coding. It can also synchronize errors during programming. Yang et al. [170] developed a collaborative photo sharing system, which received photo streams from different sources. Their system can also produce a summary from the photo streams. Recently, Binda et al. [12] developed a photo sharing system to stay aware of family members' health. Mozilla Foundation developed BrowserQuest [97], an online collaborative game which supports multiple users in one game. Chen et al. [18] also proposed a framework for multiplayer online games with the help of We-

bGL and WebSocket. Lee et al. [69] developed a UI design tool to explore real-time collaboration in crowd-powered systems.

Meanwhile, by utilizing modern HTML5 technologies, some researchers are devoted to integrating collaboration into traditional video conferencing products. For instance, Knuz et al. [66] created a device named CollaBoard which supports remote collaborative whiteboard based on video conferencing. To enhance the convenience of setup and usage, recently, Wenzel et al. [165] developed Tele-Board to provide web-based real-time collaboration combined with WebRTC-based video conferencing. They embedded the video in an iframe element on web browsers and adhered a drawing layer on top of the video to hold shared artifacts in the workspace. Chang et al. [17] proposed AlphaRead to support collaborative annotation in video-based objects.

5.6 Conclusion

In this chapter, we proposed a context-aware collaborative framework, which supports collaboration on multimedia and uses simple messages to represent media controls. We demonstrated the design methodologies of a distributed collaboration framework and discussed the implementation of architecture components. We compared our system with screen sharing tools such as Google Hangouts. Our evaluation results showed that our tool has a lower bandwidth usage than that of screen sharing tools.

The next chapter describes the details of collaborative web browsing, which is very useful in a real multimedia collaboration application. The collaboration is im-

plemented through a Chrome extension to inject additional JavaScript code to target web pages.

6. COLLABORATIVE WEB BROWSING

6.1 Introduction

With the advent of modern web browsers, more and more materials can be accessed directly online, and the demand for collaborative web browsing is quite high in our daily lives. For example, family members may want to browse commercial products on Amazon from various machines collaboratively, including the same page, scrolling position, and even annotations. To achieve the collaboration, some prior work such as WebSplitter [56] creates a proxy middleware with a proxy server. The server serves the target web resources to clients in the same session.

The proxy-based approach provides a circumvent for collaborative web browsing, although there are some drawbacks in this approach. With the wide usage of HTTPS (Hyper Text Transfer Secure), the secure version of HTTP, many websites are not allowed to be intercepted by a proxy server because of concern of security. Moreover, a server is required for the collaboration, leading to additional software setup.

To our knowledge, there is also a lack of annotation tools in prior work. For example, in a collaborative web browsing session, users may want to highlight some text for additional attention. However, we do not find any prior work can provide such precise annotation mechanisms.

To fill the research gaps in prior work mentioned above, we propose an object-based collaboration web browsing mechanism with the help of a Chrome extension. All manipulations including clicks, scrolls, and annotations on a web page are captured as object-based DOM events, explained in Chapter 5. These manipulations are serialized to simple messages and broadcast to other participants. The synchronization is accurate and efficient.

6.2 Challenges

To our study, there are two major challenges in creating a comprehensive web-based collaboration platform which supports web pages: same-origin policy and responsive web design.

Same-origin policy. The same-origin policy is a crucial concept in web browser security protection. According to this policy, a web browser can permit a script from one page to directly access data such as DOM of the document in another page, only if the two web pages are of the same origin [4, 155]. An origin is defined as the combination of scheme, host, and port of a URL [153]. Because we provide a totally browser-based uniform multimedia collaborative presentation system, the web pages have to be embedded in our tool as iframe objects. However, because of the same-origin policy, with the scripts on our web page, we cannot directly capture the DOM events in the embedded pages, and we also cannot replay the events there. For example, in a normal web page, we can monitor the *onclick* event of a button by registering an event handler in JavaScript. While, for a *onclick* event on a button in

the web page embedded in an iframe object, no matter where we register the event listener, we cannot capture it via the outside world.

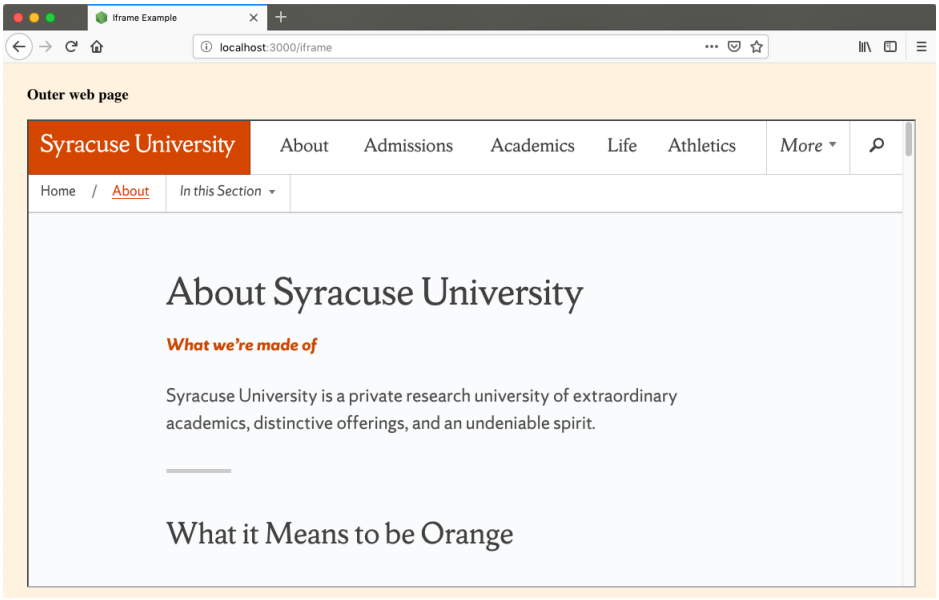


Fig. 6.1.: An example of embedding the Syracuse University website in an iframe.

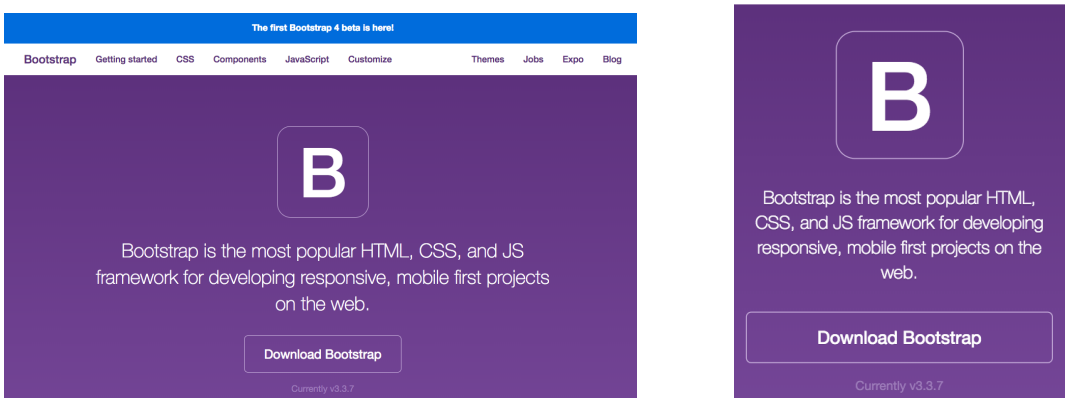
Responsive web design. With the advent of HTML5 technologies and the increasing popularity of mobile devices, many websites have applied a new design paradigm called responsive web design. In this design, a web page's layout and content can be automatically adapted across various digital devices including desktops, tablets, and mobile phones, with different screen sizes, to enrich user experience [41]. In this design, a meta viewport tag is usually necessary in the head part of a web page to control dimensions and scaling. Additionally, media queries are also applied to adjust CSS rules on various devices. An example of using media query in the Twitter Bootstrap framework is shown in Figure 6.2. Here, it assigns the max width of elements of the class container to 540 px when the minimum width of the viewport is 576 px.

```
@media (min-width: 576px) {
  .container {
    max-width: 540px;
  }
}
```

Fig. 6.2.: An example of using media queries.

According to [157], because of the flexibility of CSS media queries, there are two design approaches in responsive web design: mobile-first and desktop-first, depending on the platform the default styles are targeted. An example of a popular mobile-first UI library, Twitter Bootstrap is shown in Figure 6.3. Here, for one web page, *getbootstrap.com/docs/3.3*, it has different layouts and contents on a MacBook Air and an iPhone 8 device. The website context is adapted by fluid grid-system to concrete screen sizes. For example, on MacBook Air, the navigation bar contains various links such as "Getting started" and "CSS". By contrast, on iPhone 8, the links are collapsed into a single button. The menu of links will not be shown until the toggle is clicked.

With the modern responsive web design, it is difficult to apply traditional position-based or proportion-based synchronization techniques, considering that the layouts on different machines can be distinct. Still take the web page layouts in Figure 6.3 as an example; if the presenter clicks the "Components" link on the navigation bar on the MacBook Air, we can record the position of the operation, but at the same position or percentage of height/width on the web page on the iPhone 8, there is no such link to click.



(a) Web page on 13" MacBook Air. **(b)** Web page on iPhone 8.

Fig. 6.3.: Different web page layouts of website *getbootstrap.com/docs/3.3/* on MacBook Air and iPhone 8.

6.3 Synchronization of Web Browsing Actions

The major issue caused by the same-origin policy is that the scripts in the external page do not have access to DOM trees of the pages embedded in an iframe object. While, a Chrome extension provides us the capability to inject additional controls into an embedded page from the outside world, as explained in Figure 6.4. The execution of an embedded webpage and that of our extension are isolated, and will not affect each other. Finally, the two modified DOM trees are merged for further rendering. In that case, in our code, we can create and append a new node to the embedded page.

By using this approach, we capture the events triggered in the embedded pages and replay them on other client sides. Although the events can be captured in the embedded pages, we still need to transmit them out to the outer web page in messages for further synchronization and collaboration. This involves in cross-origin commu-

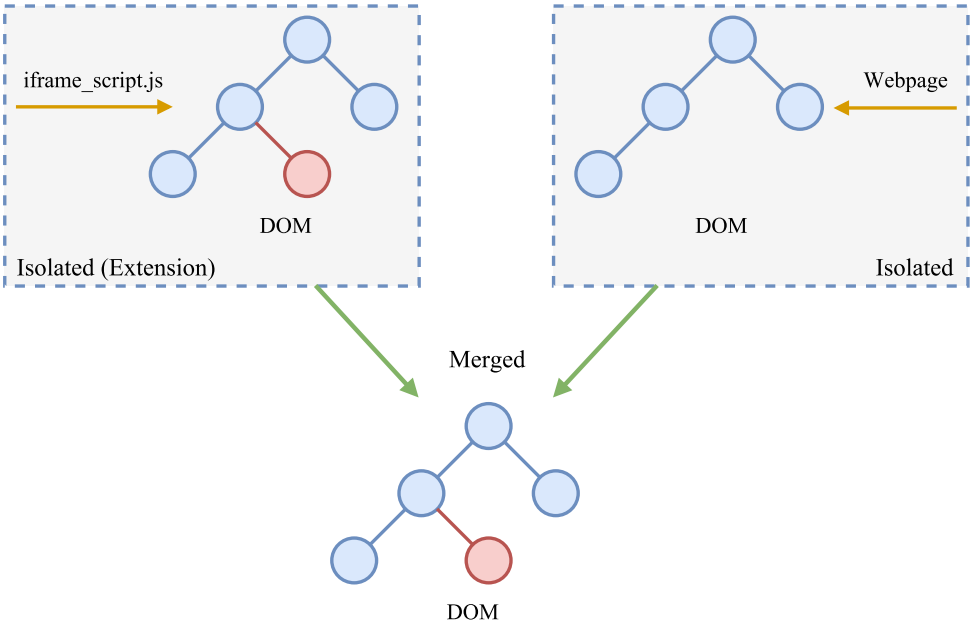


Fig. 6.4.: Execution of a Chrome extension content script.

nication, and we use *postMessage API* [149] to ensure the communication between window objects. The concrete implementation is shown in Figure 6.5.

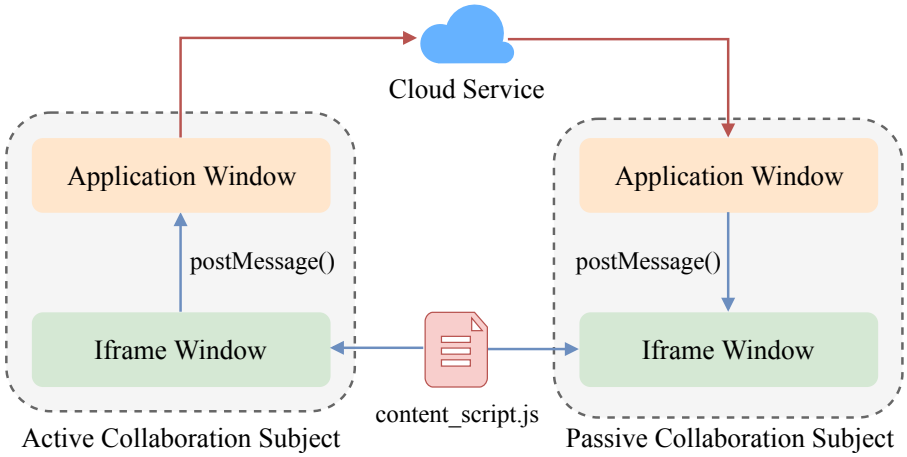


Fig. 6.5.: Communication among windows in a web page synchronization.

Suppose that an event is triggered in the iframe window. The event is first captured by our extension, assembled into a message as what we have described. Af-

terward, the message is sent to the presentation window via *postMessage API*. We register essential event listeners there to receive the message and broadcast it to other collaboration subjects, via the central cloud service. For a passive collaboration subject, the message is received in the application window and then sent to its iframe window, also via *postMessage API*. Finally, the message is handled and the event is replayed to realize the synchronization. Note that the messages exchanged between the application window and iframe window are the same as the messages transferred among collaboration subjects. In that case, specific media event replayers are necessary to be deployed in iframe windows to precisely replay the events on web pages. Note that the client side needs to check the message before executing the actions contained in the message [131].

Another challenge is accurate web page synchronization, especially in responsive web design. According to the *object-prioritized* principle introduced in Chapter 5, we capture the events on web pages by monitoring the changes on objects. For example, suppose the active collaboration subject clicks a button to view another link, the *click* event of the button can be captured and replayed on passive collaboration subjects' sides. An issue here is how to pinpoint the button object. Unlike the message defined in Figure 5.16a, where an id attribute is used to identify the button element, we have no prior knowledge of the web pages collaborated on.

Here, we introduce a path-based approach to uniquely specify the object where the target event fired. Additionally, it provides an easy way to serialize/deserialize the representation of the object. For serialization, shown in Algorithm 1, starting from the node where the captured event fired, we find the index of the node among its siblings

and push this index into a *path* variable. We repeat this step until we arrive at the root element, aka the document node, of a web page. Finally, we serialize the list of indexes to a string for broadcasting. For deserialization, explained in Algorithm 2, we first convert the representation string to a list of indexes. Afterward, traversing the list, and for each index, we find the child node with this index and set current node to this specific child node. Finally, the current node is the object we need.

Algorithm 1 Serialization of Representation of Synchronized Objects.

```
1: var path;
2: var curr_node;
3: while curr_node is not root element do
4:     set index := curr_node's index in all siblings;
5:     push index to path;
6:     set curr_node := parent node;
7: end while
8: reverse path;
9: serialize path to string;
10: return path
```

Algorithm 2 Deserialization of Representation of Synchronized Objects.

```
1: Initialize path with the passed representation string;
2: Deserialize path string to a list of indexes;
3: var curr_node := root element;
4: for each n ∈ path do
5:     set curr_node := (n + 1)th child of curr_node;
6: end for
7: return curr_node
```

An example is demonstrated in Figure 6.6 for better illustration. In this diagram, a DOM is drawn, and the button element is the object where an event is triggered. At first, the *path* variable is empty. We start from the button node, noticing that it is the only child of the parent node, so we push 0 to *path* variable. In the next step, the div element is the third child of the body element, so we append 2 to *path*

variable. Afterward, the body element is the second child of the html element, so we add 1 to *path* variable. Finally, we arrive at the root element, and *path* contains [0, 2, 1]. We reverse the path, convert it into a string, and send it to passive collaboration subjects. For deserialization, we first convert the string to a list of indexes. In the example, the list is [1, 2, 0]. Afterward, starting from the root element, according to each index in the list, we visit the child node with the associated index. Here, the first index is 1, and the second child of html element is the body element, so we visit the body element. Then, with the index of 2, we visit the div element. And finally, we arrive at the button element, which is the accurate object to synchronize.

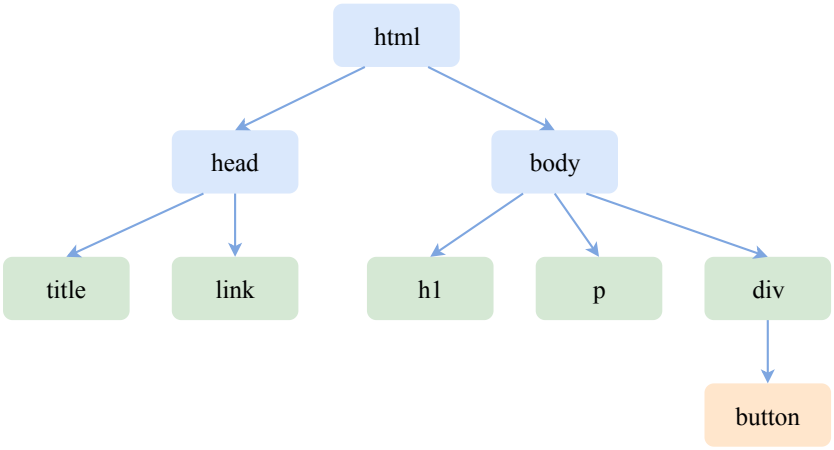


Fig. 6.6.: Tree structure related to the node where an event takes place.

6.4 Related Work

Some pioneering work in collaborative web browsing includes GroupWeb [51], CoBrow [130], Silhouettell [105], and Letizia [77]. These tools were developed when the web was just boosting, and they only provide very basic collaboration mechanisms

via various approaches. For example, in GroupWeb [51], every participant keeps a replica, and each synchronization is based on the replica. The authors designed a document slaving mechanism to synchronize webpages after the presenter navigates a link. Because visual contents might be different depending on display sizes, they relaxed the "what-you-see-is-what-I-see" (WYSIWIS) so that people may not always see the same things on their displays. By contrast, Silhouettell [105] used a large display screen to project participants' locations together for collaboration purpose.

Further work aims to add more beneficial functionalities for collaboration. For example, Wiltse [166] et al. created a web-browsing tool to interpret users' browsing actions into natural languages and synchronize the messages to other sides. For another instance, Maekawa et al. [89] set up a collaborative web browsing system for multiple users. And for every web page, it is segmented to components, which are sent to different devices. Because the displays of a web page can be variant on different platforms, researchers proposed to use proxy(agent)-based approach to serve the collaboration. In this approach, there is a server to serve the web pages, and participants connect to this server to fetch materials. For example, WebSplitter [56] employs a unified XML framework to enable collaborative web browsing. There is a proxy middleware holding web resources and document-splitting functionalities, and delivering partial views to different users. One issue of this kind of approach is that a proxy server is necessary, and the original web page contents need to be modified at the server, which may lead to possible security issues.

6.5 Conclusion

In this chapter, we demonstrated a Chrome extension-based approach to achieve collaborative web browsing. The extension injects additional JavaScript code into target web pages, monitors events on the elements, synchronizes the events to other clients, and repeats them through the injected control.

The next chapter concludes this dissertation and shows some interesting suggestion for further research.

7. CONCLUSION AND FUTURE WORK

In this dissertation, to fill the research gaps in prior work of multimedia systems and real-time collaboration, we proposed a web-based collaborative multimedia presentation document system with rich functionalities including preparation of temporal and spatial models, real-time interactive controls these models, extraction and loading of external dynamic web resources, storage of presentations in simple messages, and precise replay of presentations. Additionally, we discussed some interesting topics in developing the system, including the approaches to achieve continuous updates of external dynamic web elements and collaborative web browsing. After a detailed evaluation, we claimed that not only can our system provide fine-grained interactive controls of presentations, but also it is very efficient and requires very low bandwidth usages.

As our hybrid multimedia model and collaboration protocol are flexible and easy to be extended, we posit our work can serve as a direction for interactive and collaborative presentation platforms.

There are some interesting aspects to develop based on our research work. Some ideas are given below.

As mentioned in Chapter 5, to support collaboration on a new type of media, the developer needs to add essential handlers in the *media event capturer*, *media state recorder*, and *media event replayer*. It could be even more convenient if there exists

an automatic framework to complete the task. A possible idea is to allow a developer to provide basic media information in JSON style messages, and the framework can generate a code skeleton for the developer to start.

Another idea falls into the domain of Human-Computer Interaction (HCI). A possible further research direction is to study the influence of the user interfaces and interaction styles on participants. The results can be used to improve the current document system.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Aww app, 2019. URL `https://awwapp.com/`.

[2] J. F. Adam and D. L. Tennenhouse. The vidboard: A video capture and processing peripheral for a distributed multimedia system. *Multimedia Systems*, 2(4):150–156, 1994.

[3] A. Adler, J. C. Nash, and S. Noël. Evaluating and implementing a collaborative office document system. *Interacting with computers*, 18(4):665–682, 2006.

[4] A. Barth. The web origin concept. Technical report, 2011.

[5] S. Barua. An interactive multimedia system on computer architecture, organization, and design. *IEEE Transactions on Education*, 44(1):41–46, 2001.

[6] R. M. Bastos and D. D. A. Ruiz. Extending uml activity diagram for workflow modeling in production systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 3786–3795. IEEE, 2002.

[7] R. Bentley and W. Appelt. Designing a system for cooperative work on the world-wide web: Experiences with the bscw system. In *System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on*, volume 4, pages 297–306. IEEE, 1997.

[8] R. Bentley, T. Horstmann, and J. Trevor. The world wide web as enabling technology for cscw: The case of bscw. *Computer Supported Cooperative Work (CSCW)*, 6(2-3):111–134, 1997.

[9] A. Bergkvist, D. C. Burnett, C. Jennings, A. Narayanan, and B. Aboba. Webrtc 1.0: Real-time communication between browsers. *Working draft, W3C*, 91, 2012.

[10] G. Berhe, L. Brunie, and J.-M. Pierson. Content adaptation in distributed multimedia system. *Journal of Digital Information Management*, 3(2):95, 2005.

[11] A. Bhardwaj, A. Deshpande, A. J. Elmore, D. Karger, S. Madden, A. Parameswaran, H. Subramanyam, E. Wu, and R. Zhang. Collaborative data analytics with datahub. *Proceedings of the VLDB Endowment*, 8(12): 1916–1919, 2015.

[12] J. Binda, E. Georgieva, Y. Yang, F. Gui, J. Beck, and J. M. Carroll. Phamilyhealth: A photo sharing system for intergenerational family collaboration on health. In *Companion of the 2018 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 337–340. ACM, 2018.

[13] K. S. Booth, B. D. Fisher, C. J. R. Lin, and R. Argue. The mighty mouse multi-screen collaboration tool. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 209–212. ACM, 2002.

[14] P. Branch, G. Egan, and B. Tonkin. Modeling interactive behaviour of a video based multimedia system. In *Communications, 1999. ICC'99. 1999 IEEE International Conference on*, volume 2, pages 978–982. IEEE, 1999.

[15] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66, 1997.

[16] B. E. Brewington and G. Cybenko. How dynamic is the web? 1. *Computer Networks*, 33(1-6):257–276, 2000.

[17] Y.-C. Chang, H.-C. Wang, H.-k. Chu, S.-Y. Lin, and S.-P. Wang. Alpharead: Support unambiguous referencing in remote collaboration with readable object annotation. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 2246–2259. ACM, 2017.

[18] B. Chen and Z. Xu. A framework for browser-based multiplayer online games using webgl and websocket. In *Multimedia Technology (ICMT), 2011 International Conference on*, pages 471–474. IEEE, 2011.

[19] F. Chen. Cropper.js, 2019. URL `https://github.com/fengyuanchen/cropperjs`.

[20] M. Chen, X. Qiu, W. Xu, L. Wang, J. Zhao, and X. Li. Uml activity diagram-based automatic test case generation for java programs. *The Computer Journal*, 52(5):545–556, 2007.

[21] M.-S. Chen, Z.-Y. Shae, D. D. Kandlur, T. P. Barzilai, and H. M. Vin. A multimedia desktop collaboration system. In *Global Telecommunications Conference, 1992. Conference Record., GLOBECOM'92. Communication for Global Users., IEEE*, pages 739–746. IEEE, 1992.

[22] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up eclipse with collaborative tools. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 45–49. ACM, 2003.

[23] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. Technical report, Stanford, 1999.

[24] G. Ciocca, I. Gagliardi, and R. Schettini. Quicklook2: An integrated multimedia system. *Journal of Visual Languages and Computing*, 12(1):81–103, 2001.

[25] D. Crockford. The application/json media type for javascript object notation (json). 2006.

[26] R. Dahl. Nodejs. 2009. URL `https://nodejs.org/en/`.

[27] M. Diaz and P. Sénac. Time stream petri nets a model for timed multimedia information. In *International Conference on Application and Theory of Petri Nets*, pages 219–238. Springer, 1994.

[28] J. D. N. Dionisio and A. F. Cárdenas. A unified data model for representing multimedia, timeline, and simulation data. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):746–767, 1998.

[29] M. Douneva, R. Jaron, and M. T. Thielsch. Effects of different website designs on first impressions, aesthetic judgements and memory performance after short presentation. *Interacting with Computers*, 28(4):552–567, 2016.

[30] E. Ebeid, D. Quaglia, and F. Fummi. Generation of systemc/tlm code from uml/marte sequence diagrams for verification. In *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 187–190. IEEE, 2012.

[31] S. Elrod, R. Bruce, R. Gold, D. Goldberg, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, et al. Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 599–607. ACM, 1992.

[32] R. Eshuis and R. Wieringa. Comparing petri net and activity diagram variants for workflow modelling–a quest for reactive petri nets. In *Petri Net Technology for Communication-Based Systems*, pages 321–351. Springer, 2003.

[33] Etherpad. Etherpad, 2008. URL `http://etherpad.org`.

[34] J. Ezpeleta, J. M. Colom, and J. Martinez. A petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE transactions on robotics and automation*, 11(2):173–184, 1995.

[35] Fabric.js. Fabric.js, 2014. URL `http://fabricjs.com/`.

[36] I. Fette and A. Melnikov. WebSocket protocol, 2019. URL `https://tools.ietf.org/html/rfc6455`.

[37] M. Fetter, R. Strobel, and T. Gross. Lightweight support for collaborative web browsing through spreadvector. In *Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing systems*, pages 1339–1344. ACM, 2014.

[38] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the 12th international conference on World Wide Web*, pages 669–678. ACM, 2003.

[39] G. Firebase. Firebase. *Firebase Realtime Database,[Online]. Available: https://firebase. google. com/docs/database*.

[40] M. Flower. Event Sourcing, 2005. URL `https://martinfowler.com/eaaDev/EventSourcing.html`.

[41] B. S. Gardner. Responsive web design: Enriching the user experience. *Sigma Journal: Inside the Digital Ecosystem*, 11(1):13–19, 2011.

[42] C. V. Geambaşu. Bpmn vs uml activity diagram for business process modeling. *Accounting and Management Information Systems*, 11(4):637–651, 2012.

[43] S. Gibbs, L. Dami, and D. Tsichritzis. An object-oriented framework for multimedia composition and synchronisation. In *Multimedia*, pages 101–111. Springer, 1992.

[44] M. Goldman, G. Little, and R. C. Miller. Real-time collaborative coding in a web ide. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 155–164. ACM, 2011.

[45] Google. Google Docs, 2006. URL `https://docs.google.com`.

[46] Google. Google Slides, 2006. URL `https://www.google.com/slides/about/`.

[47] Google. Google Hangouts, 2013. URL `https://hangouts.google.com`.

[48] Google. Blink, 2019. URL `https://www.chromium.org/blink`.

[49] Google. Headless Chrome, 2019. URL `https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md`.

[50] S. Greenberg and R. Bohnet. Group sketch: a multi-user sketchpad for geographically-distributed small groups. 1990.

[51] S. Greenberg and M. Roseman. Groupweb: A www browser as real time groupware. In *Conference Companion on Human Factors in Computing Systems*, pages 271–272. ACM, 1996.

[52] T.-M. Gronli, J. Hansen, and G. Ghinea. A context-aware meeting room: Mobile interaction and collaboration using android, java me and windows mobile. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*, pages 311–316. IEEE, 2010.

[53] N. Guelfi and A. Mammar. A formal semantics of timed activity diagrams and its promela translation. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 8–pp. IEEE, 2005.

[54] C. A. Gutwin, M. Lippold, and T. Graham. Real-time groupware in the browser: testing the performance of web-based networking. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 167–176. ACM, 2011.

[55] R. Hamadi and B. Benatallah. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference-Volume 17*, pages 191–200. Australian Computer Society, Inc., 2003.

[56] R. Han, V. Perret, and M. Naghshineh. Websplitter: a unified xml framework for multi-device collaborative web browsing. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 221–230. ACM, 2000.

[57] J. H. Hausmann, R. Heckel, and S. Sauer. Dynamic meta modeling with time: Specifying the semantics of multimedia sequence diagrams. *Software & Systems Modeling*, 3(3):181–193, 2004.

[58] N. Hirzalla, B. Falchuk, and A. Karmouch. A temporal model for interactive multimedia scenarios. *IEEE MultiMedia*, 2(3):24–31, 1995.

[59] P.-Y. Hsu, Y.-B. Chang, and Y.-L. Chen. Strpn: a petri-net approach for modeling spatial-temporal relations between moving multimedia objects. *IEEE Transactions on Software Engineering*, 29(1):63–76, 2003.

[60] A. K. Jena, S. K. Swain, and D. P. Mohapatra. A novel approach for test case generation from uml activity diagram. In *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pages 621–629. IEEE, 2014.

[61] Z. Kemp. Multimedia and spatial information systems. *IEEE MultiMedia*, 2 (4):68–76, 1995.

[62] K. Kim, K. Kim, K. Lee, T. Kim, and W. Cho. Design and implementation of web crawler based on dynamic web collection cycle. In *Information Networking (ICOIN), 2012 International Conference on*, pages 562–566. IEEE, 2012.

[63] K. Kim, W. Ha, O. Choi, H. Yeh, J.-H. Kim, M. Hong, and T. Shon. An interactive pervasive whiteboard based on mvc architecture for ubiquitous collaboration. *Multimedia Tools and Applications*, 74(5):1557–1576, 2015.

[64] R. Klauck, J. Gaebler, M. Kirsche, and S. Schoepke. Mobile xmpp and cloud service collaboration: An alliance for flexible disaster management. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 201–210. IEEE, 2011.

[65] A. Kolber. audio.js, 2019. URL `https://kolber.github.io/audiojs/`.

[66] A. Kunz, T. Nescher, and M. Kuchler. Collaboard: a novel interactive electronic whiteboard for remote collaboration with people on content. In *Cyberworlds (CW), 2010 International Conference on*, pages 430–437. IEEE, 2010.

[67] Y.-M. Kwon, E. Ferrari, and E. Bertino. Modeling spatio-temporal constraints for multimedia objects. *Data & Knowledge Engineering*, 30(3):217–238, 1999.

[68] M. Lai and D. Wong. Slicing tree is a complete floorplan representation. In *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, pages 228–232. IEEE, 2001.

[69] S. W. Lee, R. Krosnick, S. Y. Park, B. Keelean, S. Vaidya, S. D. O'Keefe, and W. S. Lasecki. Exploring real-time collaboration in crowd-powered systems through a ui design tool. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):104, 2018.

[70] Y. Lee, S. Oh, and W. Woo. A context-based storytelling with a responsive multimedia system (rms). In *International Conference on Virtual Storytelling*, pages 12–21. Springer, 2005.

[71] R. Lefevre. caman.js, 2016. URL `https://github.com/meltingice/CamanJS/`.

[72] B.-L. Li, Z.-s. Li, L. Qing, and Y.-H. Chen. Test case automate generation from uml sequence diagram and ocl expression. In *2007 international conference on computational intelligence and security (cis 2007)*, pages 1048–1052. IEEE, 2007.

[73] L. Li, A. Karmouch, and N. D. Georganas. Multimedia teleorchestra with independent sources: Part 1—temporal modeling of collaborative multimedia scenarios. *Multimedia Systems*, 1(4):143–153, 1994.

[74] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 114–130. IEEE, 2002.

[75] N. Li, Ł. Kidziński, P. Jermann, and P. Dillenbourg. Mooc video interaction patterns: What do they tell us? In *Design for teaching and learning in a networked world*, pages 197–210. Springer, 2015.

[76] Z. Li and M. Zhou. Elementary siphons of petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 34(1):38–51, 2004.

[77] H. Lieberman, N. Van Dyke, and A. Vivacqua. Let's browse: a collaborative browsing agent. *Knowledge-Based Systems*, 12(8):427–431, 1999.

[78] V. Lima, C. Talhi, D. Mouheb, M. Debbabi, L. Wang, and M. Pourzandi. Formal verification and validation of uml 2.0 sequence diagrams using source and destination of messages. *Electronic Notes in Theoretical Computer Science*, 254:143–160, 2009.

[79] B. Lin, Y. Chen, X. Chen, and Y. Yu. Comparison between json and xml in applications based on ajax. In *Computer science & service system (csss), 2012 international conference on*, pages 1174–1177. IEEE, 2012.

[80] C.-C. Lin, H.-H. Chin, and D.-J. Deng. Dynamic multiservice load balancing in cloud-based multimedia system. *IEEE systems journal*, 8(1):225–234, 2014.

[81] G. Lindblad, D. J. Wetherall, W. F. Stasior, J. F. Adam, H. H. Houh, M. Ismert, D. R. Bacher, B. M. Phillips, and D. L. Tennenhouse. Viewstation applications: implications for network traffic. *IEEE Journal on Selected Areas in Communications*, 13(5):768–778, 1995.

[82] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang. Generating test cases from uml activity diagram based on gray-box method. In *11th Asia-Pacific software engineering conference*, pages 284–291. IEEE, 2004.

[83] T. D. C. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *IEEE journal on selected areas in communications*, 8(3):413–427, 1990.

[84] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A continual query system for update monitoring in the www. *Computer Systems Science and Engineering*, 14(2):99–112, 1999.

[85] L. Liu, C. Pu, and W. Tang. Webcq-detecting and delivering information changes on the web. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 512–519. ACM, 2000.

[86] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 401–416. ACM, 2011.

[87] A. Lombardi. *WebSocket: Lightweight Client-server Communications.* " O'Reilly Media, Inc.", 2014.

[88] K. Maeda. Performance evaluation of object serialization libraries in xml, json and binary formats. In *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, pages 177–182. IEEE, 2012.

[89] T. Maekawa, T. Hara, and S. Nishio. A collaborative web browsing system for multiple mobile users. In *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, pages 12–pp. IEEE, 2006.

[90] A. Mesbah, E. Bozdag, and A. Van Deursen. Crawling ajax by inferring user interface state changes. In *Web Engineering, 2008. ICWE'08. Eighth International Conference on*, pages 122–134. IEEE, 2008.

[91] A. Mesbah, A. Van Deursen, and S. Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web (TWEB)*, 6(1):3, 2012.

[92] Microsoft. Microsoft Skype, 2003. URL `https://www.skype.com`.

[93] Microsoft. Microsoft Powerpoint, 2018. URL `https://products.office.com/en-us/powerpoint`.

[94] S. Mogan and W. Wang. The impact of web 2.0 developments on real-time groupware. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 534–539. IEEE, 2010.

[95] M. K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on computers*, (9):913–917, 1982.

[96] MongoDB. Mongodb, 2009. URL `https://www.mongodb.com/`.

[97] Mozilla. BrowserQuest, 2014. URL `http://browserquest.mozilla.org/`.

[98] Mozilla. HTML Canvas API, 2019. URL `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API`.

[99] Mozilla. HTML5 Range API, 2019. URL `https://developer.mozilla.org/en-US/docs/Web/API/Range`.

[100] Mozilla. Mutation Observer, 2019. URL `https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver`.

[101] Mozilla. pdf.js, 2019. URL `https://mozilla.github.io/pdf.js/`.

[102] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference on World Wide Web*, pages 77–88. ACM, 2002.

[103] Northwoods. GoJS, 2019. URL `https://gojs.net/latest/index.html`.

[104] A. Ntoulas, J. Cho, and C. Olston. What's new on the web?: the evolution of the web from a search engine perspective. In *Proceedings of the 13th international conference on World Wide Web*, pages 1–12. ACM, 2004.

[105] M. Okamoto, H. Nakanishi, T. Nishimura, and T. Ishida. Silhouettell: Awareness support for real-world encounter. In *Community computing and support systems*, pages 316–329. Springer, 1998.

[106] R. H. Otten. Automatic floorplan design. In *Proceedings of the 19th Design Automation Conference*, pages 261–267. IEEE Press, 1982.

[107] O. Ozturk. Introduction to xmpp protocol and developing online collaboration applications using open source software and libraries. In *Collaborative Technologies and Systems (CTS), 2010 International Symposium on*, pages 21–25. IEEE, 2010.

[108] S. Pandey, K. Ramamritham, and S. Chakrabarti. Monitoring the dynamic web to respond to continuous queries. In *Proceedings of the 12th international conference on World Wide Web*, pages 659–668. ACM, 2003.

[109] Parse and Facebook. Parse server. `https://parseplatform.org/`, 2019.

[110] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pages 50–59. IEEE, 2002.

[111] D. Perkins, N. Agrawal, A. Aranya, C. Yu, Y. Go, H. V. Madhyastha, and C. Ungureanu. Simba: Tunable end-to-end data consistency for mobile apps. In *Proceedings of the Tenth European Conference on Computer Systems*, page 7. ACM, 2015.

[112] C. A. Petri. Kommunikation mit automaten. 1962.

[113] PhantomJS. PhantomJS. `http://phantomjs.org/`, 2018.

[114] V. Pimentel and B. G. Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, 16(4):45–53, 2012.

[115] N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. In *USENIX Security Symposium*, 2003.

[116] A. Prunicki. Apache thrift. Technical report, Technical report, Object Computing, Inc, 2009.

[117] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed petri nets.* PhD thesis, Massachusetts Institute of Technology, 1973.

[118] J. Resig, B. Bibeault, and J. Maras. *Secrets of the JavaScript ninja*. Manning Publications Co., 2016.

[119] K. Salimifard and M. Wright. Petri net-based modelling of workflow systems: An overview. *European journal of operational research*, 134(3):664–676, 2001.

[120] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[121] S. Sanfilippo. Redis, 2009. URL `https://redis.io/`.

[122] M. Sarma, D. Kundu, and R. Mall. Automatic test case generation from uml sequence diagram. In *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*, pages 60–67. IEEE, 2007.

[123] S. Sauer and G. Engels. Extending uml for modeling of multimedia applications. In *Proceedings 1999 IEEE Symposium on Visual Languages*, pages 80–87. IEEE, 1999.

[124] S. Sauer and G. Engels. Uml-based behavior specification of interactive multimedia applications. In *Proceedings IEEE Symposia on Human-Centric Computing Languages and Environments (Cat. No. 01TH8587)*, pages 248–255. IEEE, 2001.

[125] O. Schmid, A. L. Masson, and B. Hirsbrunner. Real-time collaboration through web applications: an introduction to the toolkit for web-based interactive collaborative environments (twice). *Personal and Ubiquitous Computing*, 18(5): 1201–1211, 2014.

[126] A. Sharma and A. Dixit. Self adjusting refresh time based architecture for incremental web crawler. *International Journal of Computer Science and Network Security*, 8(12):349–354, 2008.

[127] H. Shen. A semantic-aware attribute-based access control model for web services. *Algorithms and Architectures for Parallel Processing*, pages 693–703, 2009.

[128] H.-b. Shen and F. Hong. An attribute-based access control model for web services. In *Parallel and Distributed Computing, Applications and Technologies, 2006. PDCAT'06. Seventh International Conference on*, pages 74–79. IEEE, 2006.

[129] A. Shraer, A. Aybes, B. Davis, C. Chrysafis, D. Browning, E. Krugler, E. Stone, H. Chandler, J. Farkas, J. Quinn, et al. Cloudkit: structured storage for mobile applications. *Proceedings of the VLDB Endowment*, 11(5):540–552, 2018.

[130] G. Sidler, A. Scott, and H. Wolf. Collaborative browsing in the world wide web. In *Proceedings of the 8th Joint European Networking Conference*, pages 122–1. Citeseer, 1997.

[131] S. Son and V. Shmatikov. The postman always rings twice: Attacking and defending postmessage in html5 websites. In *NDSS*, 2013.

[132] S. Souders. Velocity and the Bottom Line, 2009. URL `http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html`.

[133] T. Springer, D. Schuster, I. Braun, J. Janeiro, M. Endler, and A. A. Loureiro. A flexible architecture for mobile collaboration services. In *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*, pages 118–120. ACM, 2008.

[134] A. Sreeramaneni, H. Im, W. M. Kang, C. Koh, and J. H. Park. Cims: a context-based intelligent multimedia system for ubiquitous cloud computing. *Information*, 6(2):228–245, 2015.

[135] W. Stallings and L. Brown. *Computer Security: Principles and Practice, 3rd Edition*. Pearson, 2015.

[136] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and control*, 57(2-3):91–101, 1983.

[137] S. K. Swain, D. P. Mohapatra, and R. Mall. Test case generation based on use case and sequence diagram. *International Journal of Software Engineering*, 3 (2):21–52, 2010.

[138] W. Tan, Y. Fan, and M. Zhou. A petri net-based method for compatibility analysis and composition of web services in business process execution language. *IEEE Transactions on Automation Science and Engineering*, 6(1):94–106, 2009.

[139] J. C. Tang and S. Minneman. Videowhiteboard: video shadows to support remote collaboration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 315–322. ACM, 1991.

[140] D. L. Tennenhouse, J. Adam, D. Carver, H. Houh, M. Ismert, C. Lindblad, B. Stasior, D. Wetherall, D. Bacher, and T. Chang. The viewstation: a software-intensive approach to media processing and distribution. *Multimedia Systems*, 3(3):104–115, 1995.

[141] TextHighlighter. TextHighlighter, 2016. URL `https://github.com/mir3z/texthighlighter`.

[142] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Computing Surveys (CSUR)*, 37(1):29–41, 2005.

[143] A. Vakali, E. Terzi, E. Bertino, and A. Elmagarmid. Hierarchical data placement for navigational multimedia applications. *Data & Knowledge Engineering*, 44 (1):49–80, 2003.

[144] W. M. Van Der Aalst. Three good reasons for using a petri-net-based workflow management system. In *Information and Process Integration in Enterprises*, pages 161–182. Springer, 1998.

[145] W. M. Van Der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management*, pages 161–183. Springer, 2000.

[146] A. van der Linde, P. Fouto, J. Leitão, N. Preguiça, S. Castiñeira, and A. Bieniusa. Legion: Enriching internet services with peer-to-peer interactions. In *Proceedings of the 26th International Conference on World Wide Web*, pages 283–292. International World Wide Web Conferences Steering Committee, 2017.

[147] K. Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul*, 72, 2008.

[148] video.js, 2019. URL `https://videojs.com/`.

[149] W3C. W3C postmessage, 2009. URL `https://www.w3.org/TR/2009/WD-html5-20090423/comms.html#dom-window-postmessage-2`.

[150] W3C. W3C web annotation protocol, 2017. URL `https://www.w3.org/TR/2017/REC-annotation-protocol-20170223/`.

[151] W3C. W3C html 5.2 recommendation, 2017. URL `https://www.w3.org/TR/html5/`.

[152] W3C. W3C document object model (dom) level 2 events specification, 2017. URL `https://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/`.

[153] W3C. W3C definition of origin, 2017. URL `https://www.w3.org/TR/html5/browsers.html#concept-origin`.

[154] W3C. W3C websocket api, 2017. URL `https://www.w3.org/TR/websockets/`.

[155] W3C. W3C same-origin policy, 2019. URL `https://www.w3.org/Security/wiki/Same_Origin_Policy`.

[156] W3C. W3C server-sent events, 2019. URL `https://html.spec.whatwg.org/multipage/server-sent-events.html#server-sent-events`.

[157] J. Wagner. *Web Performance in Action: Building Faster Web Pages*. Manning Publications Co., 2017.

[158] T. Wahl, S. Wirag, and K. Rothermel. Tiempo: Temporal modeling and authoring of interactive multimedia. In *proceedings of the international conference on multimedia computing and systems*, pages 274–277. IEEE, 1995.

[159] G. Wang. Improving data transmission in web applications via the translation between xml and json. In *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, pages 182–185. IEEE, 2011.

[160] J. Wang. Petri nets for dynamic event-driven system modeling. *Handbook of Dynamic System Modeling*, 1, 2007.

[161] W. Wang, K. Finch, J. Rubart, and J. M. Haake. A cooperative hypermedia approach to flexible process support for managing distributed projects. *International Journal of Cooperative Information Systems*, 18(03n04):481–512, 2009.

[162] Webex and Cisco. Cisco WebEx, 2019. URL `https://www.webex.com`.

[163] WebKit. WebKit, 2019. URL `https://webkit.org/`.

[164] H. Wen, L. Chuang, Z. Hai-ying, and Y. Yang. Effective load balancing for cloud-based multimedia system. In *Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology*, volume 1, pages 165–168. IEEE, 2011.

[165] M. Wenzel and C. Meinel. Full-body webrtc video conferencing in a web-based real-time collaboration system. In *Computer Supported Cooperative Work in Design (CSCWD), 2016 IEEE 20th International Conference on*, pages 334–339. IEEE, 2016.

[166] H. Wiltse and J. Nichols. Playbyplay: collaborative web browsing for desktop and mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1781–1790. ACM, 2009.

[167] D. Wong and C. Liu. A new algorithm for floorplan design. In *23rd ACM/IEEE Design Automation Conference*, pages 101–107. IEEE, 1986.

[168] J. Wu and T. N. Graham. The software design board: a tool supporting workstyle transitions in collaborative software design. In *International Workshop on Design, Specification, and Verification of Interactive Systems*, pages 363–382. Springer, 2004.

[169] T. Yamanouchi, K. Tamakashi, and T. Kambe. Hybrid floorplanning based on partial clustering and module restructuring. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 478–483. IEEE Computer Society, 1997.

[170] J. Yang, J. Luo, J. Yu, and T. S. Huang. Photo stream alignment and summarization for collaborative photo collection and sharing. *IEEE Transactions on Multimedia*, 14(6):1642–1651, 2012.

[171] J. Yang, S. He, Y. Lin, and Z. Lv. Multimedia cloud transmission and storage system based on internet of things. *Multimedia Tools and Applications*, 76(17): 17735–17750, 2017.

[172] D. Yoon, N. Chen, B. Randles, A. Cheatle, C. E. Löckenhoff, S. J. Jackson, A. Sellen, and F. Guimbretière. Richreview++: Deployment of a collaborative multi-modal annotation system for instructor feedback and peer discussion. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, pages 195–205. ACM, 2016.

[173] F. Y. Young and D. Wong. Slicing floorplans with pre-placed modules. In *1998 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (IEEE Cat. No. 98CB36287)*, pages 252–258. IEEE, 1998.

[174] J. Yu, T. Li, and Q. Tan. The use of uml sequence diagram for system-on-chip system level transaction-based functional verification. In *2006 6th World Congress on Intelligent Control and Automation*, volume 2, pages 6173–6177. IEEE, 2006.

[175] E. Yuan and J. Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[176] B. Zhao. Collaborative access control. In *T-110.501 seminar on Network Security*, 2001.

[177] M. Zhou and F. DiCesare. *Petri net synthesis for discrete event control of manufacturing systems*, volume 204. Springer Science & Business Media, 2012.

[178] H. Zhu and M. Zhou. Role-based collaboration and its kernel mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 36(4):578–589, 2006.

[179] L. Zhu, J.-w. Yin, G. Chen, and J.-x. Dong. Complicated product oriented collaborative document management system. In *Computer Supported Cooperative Work in Design, 2005. Proceedings of the Ninth International Conference on*, volume 2, pages 1065–1070. IEEE, 2005.

VITA

VITA

Chunxu Tang was born in Harbin, Heilongjiang province, China. He received his Bachelor of Science degree in Engineering at Xiamen University (Xiamen, Fujian, China). He received his Master of Science degree in Computer Engineering and PhD in Electrical and Computer Engineering from Syracuse University (Syracuse, New York, USA).