CloudMediate

Showcase Implementation with Google Firebase

Raimund K. Ege
Computer Science
Northern Illinois University
DeKalb, IL, USA
email: ege@niu.edu

***Abstract*—CloudMediate is a peer-to-peer multimedia stream sharing platform. It provides a media aggregation framework that is flexible, powerful and scalable to identify, establish and manage connections to input media stream sources. It enables the mediation of input streams into consumable output streams, which become part of the shared content pool. Google Firebase is a cloud service. It is used to implement CloudMediate in conjunction with the Angular JavaScript web-application framework. The implementation serves as a showcase for modern, efficient and powerful realization of cloud-based distributed applications. Details of the use of hosting, user authentication and real-time database can serve as recipe for many similar efforts.**

Keywords-peer to peer; multimedia; stream sharing; stream mediation; cloud implementation.

I. INTRODUCTION

CloudMediate is a multimedia stream sharing and processing framework. Its conceptual approach and architecture is described in [1]. CloudMediate allows users to become members of a peer-to-peer (p2p) network where streams can be posted, mediated and consumed, i.e. viewed. This paper describes an implementation of CloudMediate using a modern cloud implementation framework: Google Firebase [2]. All aspects of the p2p content sharing network are handled in the cloud and are accessible everywhere.

Mobile and wearable devices are common place today and have allowed access to a multitude of disparate but often related media streams, while scaling geographical barriers. These multimedia streams are produced, stored on and accessed from various kinds of heterogeneous devices. CloudMediate allows to register and then select suitable input streams, correlate and combine them into output streams. The output streams are then made available to peers, again rendered onto suitable mobile and wearable devices. The correlation and combination of input streams into consumable output streams is achieved by active intermediary compute nodes. CloudMediate uses the term "mediator" to describe these intermediaries. We chose this term in analogy to the "Mediator" behavioral pattern that address the responsibilities of objects in an application and how they communicate [3].

Since the streams are meant to be consumed from the original source, which can be anywhere in the Internet, we choose a cloud-based implementation of stream management. We selected Google Firebase as the implementation vehicle, since it offers all the services we needed: flexible authentication, real-time database and a JavaScript based computation engine. It provides an Angular compatible API to access its features. Google Firebase also handles world-wide hosting with exceptional scaling capabilities (if we ever need them).

The rest of this paper is organized as follows. Section II gives background information on media streaming and media mediation, plus describes the features of Google Firebase and the Angular JavaScript framework. Section III describes the CloudMediate multimedia stream sharing and mediation framework. Section IV gives implementation detail and might serve as a recipe for other cloud-based implementations of similar nature. Section V summarizes our approach and effort and closes with an outlook to our future work.

II. BACKGROUND

Devices that handle multimedia are commonplace. Every smartphone has camera(s) and high-resolution screens. Every major vendor of systems and hardware has introduced mobile and wearable gadgets to support virtual and augmented reality. From simple holders for smart phones, to optical head-mounted displays from market leaders - such as Microsoft's Windows Mixed Reality headsets, Oculus Rift, HTC Vive or even the older Google Glass - software to enable these devices is becoming more commonplace. The application developer kits are becoming ever more powerful to harness the dynamic features of these devices.

Streaming of multimedia data requires significant throughput and quality of service (QoS) factors, such as latency, jitter, order of delivery, etc. Any architectures must deal with buffering and the intermittent connection associated with mobility. Connectivity capabilities are typically wireless and include high-bandwidth cellular (4G, LTE) and WLAN (IEEE 802.11) connections, plus lower-bandwidth near field connections (Bluetooth, NFC, etc.). Transmission rates in the multi megabits per second range and latency rates in the sub millisecond range are currently quite standard.

In peer-sourced augmented reality systems, the management of the multimedia source and establishment of trust is essential [4]. In our prior work [5] [6], we investigated the authentication of participants in peer-to-peer networks, the establishment and management of trust, and the use of such

 ISBN: 978-1-61208-607-1

media sources in building content management systems. An important lesson was that while modern mobile devices are compute-capable, cloud-based components add additional heft and authority to a seamless and smooth creation of a truly immersing virtual and augmented reality experience [7][8][9].

Cloud platforms that offer integrated services are becoming more widely available. The Heroku [10] platform is probably most widely known. These platforms allow developers to compose their applications locally and then upload and deploy into cloud containers, which makes them available Internet-wide. We choose Google Firebase for the bundled services it provides, such as authentication, real-time database and compute engine, plus its comprehensive integration with the Angular [11] JavaScript web programming interface. The Angular framework offers a modern component- and object-based implementation which does not rely on server-based functionality but rather focusses on client-provided JavaScript execution. Features can be built quickly with simple, declarative templates. New components can extend a wide array of existing components.

III. CLOUDMEDIATE

A. General Framework

CloudMediate is a peer-to-peer (p2p) content sharing network and aims to aggregate multimedia streams and make them available to peers. Figure 1 shows the general approach of how CloudMediate operates. The left side of the figure symbolizes the multitude of potential input streams. While audio and video streams are most common, our approach allows arbitrary streams of data from any sensor. The right side of the figure shows consumption of streams by mobile devices. The common smartphone might be one example of such a display device. Wearable devices, such as headsets and virtual glasses are the target of our approach. Both sides are connected by a cloud-hosted network of intermediaries that normalize, correlate and combine input streams into consumable output streams.

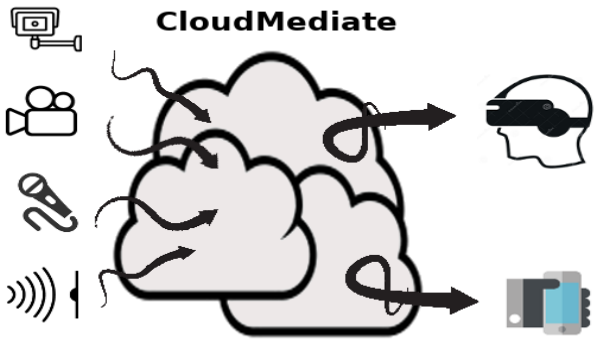

Figure 1. CloudMediate Structure

B. Mediators

CloudMediate uses the term “mediator” to describe these intermediaries. Each mediator has an input and an output side and transfers and negotiates on three kinds of information; the schema of the data stream, the data stream itself, and some QoS information specific to the stream. The nature of mediation varies from a simple combination of input streams, to correlation of virtual and augmented streams, and up to reformatting of a stream based on attitude information. Mediators can be categorized as “Combiner,” “Transformer” and “Splitter.” A Combiner is able to correlate multiple input streams into a single output stream; Transformer accepts one input stream and transforms it into one output stream based on a transformation schema which can be a set of static or dynamic parameters; Splitter is able to correlate an input stream into a number of output streams. Each mediator follows a schema that defines the relationship between the input and output streams. The schema can be parameterized via static values or a dynamic stream of input values. These general mediator categories are further specialized, examples of “Transformer” are the “Scaler” mediator, which changes a video streams resolution, or the “Securer” mediator, which takes an unencrypted input stream and produces an encrypted output stream. The “Securer” schema parameters determine the encryption algorithm, mode, parameters, and key.

In addition to the actual data sensed, each stream must be packaged with the exact time of recording. Multiple streams of sensor data are combined into a multimedia stream which interleaves its content streams plus provides meta-data to ensure their proper sequencing and correlation. It is important that the container format used to wrap the content streams is flexible enough to accommodate not only the stream data but also extensive amounts of reference information used to combine the streams. We are using an extension of the WebM project [12] format. The WebM container format is an open standard and allows us to collate an unlimited number of video, audio, pictures and subtitle tracks into one stream. We add the capability of identifying reference elements at identified points in time and at locations.

C. Peers

In a P2P communications model, peers participate on an equal basis. Each peer must register and gets a unique identity, which is made known to other peers. Limited information about each peer is collected and maintained. Currently, the only identifying information maintained is the peer’s email address. Each peer must have a confirmed email address. The most relevant information about a peer is detailed information on a peer’s participation in the network. The peer’s history of relevant transactions is maintained in a container we call “trust nugget”. This nugget contains detailed information on a peer’s participation, such as length and quality of stream transmission, ratio of seed vs. leech behavior, judgments of other stream participants, etc. It is a matter of trust whether and to what degree the peer is allowed to partake in the shared media content.

All peers have the same capabilities: any peer can contribute a stream, any peer can consume a stream, and any peer can offer a new mediator or even a new mediator category. Peers can also preconfigure existing mediators with existing input streams and parameters to create new mediated output streams.

 ISBN: 978-1-61208-607-1

IV. IMPLEMENTATION

There are several components that make up our CloudMediate implementation.

In this paper, we focus on the web-based peer access portal. This portal is implemented using the Google Firebase cloud service framework. The web application is constructed from html, css, and JavaScript pieces. All processing is done in the web client using the Angular JavaScript framework. All back-end processing is provided by Firebase.

Other CloudMediate components, such as the Android and iOS application for mobile devices are not subject of this paper. They are currently under development and might be discussed in a future paper.

A. Angular

Angular is an open-source front-end web application platform. It combines declarative templates, dependency injection with end to end tooling. Angular is a framework for building client applications in HTML and either JavaScript or a language like TypeScript that compiles to JavaScript. It empowers developers to quickly and efficiently build applications that live on the web, mobile, or the desktop.

Angular web applications are constructed from components. Each component can be viewed as a class in the object-oriented sense, with instance variables and methods, even super classes. Each component has an html template and a css style file which govern its appearance. Variables and methods are accessible from within the html template. Angular also enables 2-way data binding which reduces the amount of JavaScript code necessary to keep screen data updated. Angular components that only provide services, but have no screen appearance are configured as providers. Angular providers are made available to Angular components via dependency injection.

For example, consider the Angular code in Figure 2. It declares the "AddNewMediator" component.

```
@Component({
  selector: 'addnew-mediator',
  templateUrl: './addnew.mediator.html',
  styleUrls: ['./mediator.component.css']
})
export class AddNewMediator extends MediatorComponent {

  name: string;
  url: string;
  type: string = "please select";
  streamUrl: string;
  schemaFile: File = null;

  setType(t: string) {···
  }

  setFile(event: any) {···
  }

  onSubmit() {···
  }
}
```

Figure 2. Angular Component

The "AddNewMediator" component is declared with selector "addnew-mediator", which enables this component to be inserted into html elsewhere with the <addnew-mediator> tag. The component's appearance is specified in separate css and html files. The component also defines class "AddNewMediator" as a subclass of "MediatorComponent". It inherits all instance variables and methods from its superclass. It defines additional variables to hold the name, URL, type and schema file information. These fields are filled in the html portion of the component from within a html form. The shown methods help process the input from the form, as well govern what occurs when the form is submitted, i.e., "onSubmit()".

With Angular we use module "angularfire2", which is the official library for Firebase and Angular to directly access Firebase features.

B. Firebase Hosting

The first feature of Firebase that we use has the purpose to host our portal website within Google Firebase. Firebase features a console that allows the creation of hosting space. All that it required is user authentication with a Gmail address. While Firebase provides a URL address for the hosted website, it is also possible to redirect any domain to it. Figure 3 shows the portal's home screen at mediate.ege.com:

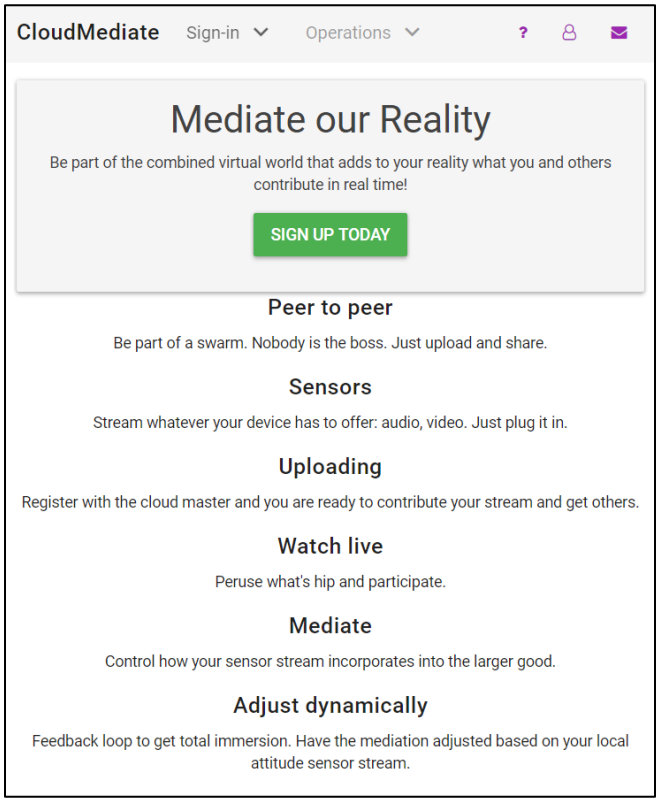

Figure 3. CloudMediate Home Screen

Firebase provides multiple levels of service. We used the "Spark" level, which is free of charge. It has limits on use and capacity. Additional levels are available at cost.

C. Firebase Authentication

Firebase allows websites to authenticate their users. Since we needed to authenticate our peers, we opted to use this feature. Among the multitude of options for authentication providers and features we selected "Google authentication" which allows Gmail users, and "Email authentication" which requires a peer to provide an email address. The confirmation of the email is handled automatically by Firebase. In both modes of authentication, each peer will have a confirmed email address. The management of user information is completely handled by Firebase. Once a peer has logged in CloudMediate's operation become available (see Figure 4):

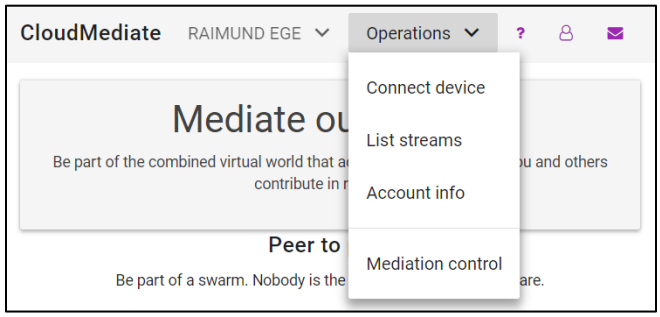

Figure 4. Peer Operations

Here, a peer can connect a stream from a device, list all available media streams, manage the account, or use the mediation features of CloudMediate.

Figure 5 shows the "Connect device" Angular component which is pulled up via menu selection:

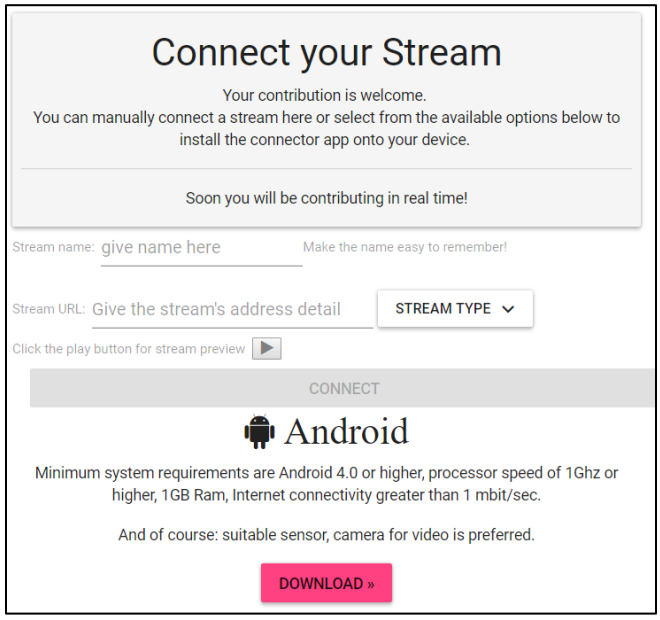

Figure 5. Stream Detail

A peer has several choices to register multimedia streams. CloudMediate calls for Android or iOS apps that can be downloaded from the presented screen. Another choice is to provide all stream detail manually here: stream name, URL, and type. Before the stream is submitted the peer can call up a preview to check the availability and suitability of the contribution. Once all fields as filled and verified, the "CONNECT" button becomes available: it submits the stream information.

D. Firebase Real-time Database

The central feature of Firebase is its database. Once it is enabled in the Firebase console, it is available via the Angular – Firebase interface. All database data is stored as JSON objects. The database is not constructed using tables or records. Simple JavaScript operations allow to manipulate the data. The database is a "real-time" database in the sense that any data modification done anywhere is immediately reflected in all places the data is used or displayed. All streams that were connected to CloudMediate are stored in the Firebase real-time database.

Figure 6 shows "List streams" Angular component which is pulled up via menu selection:

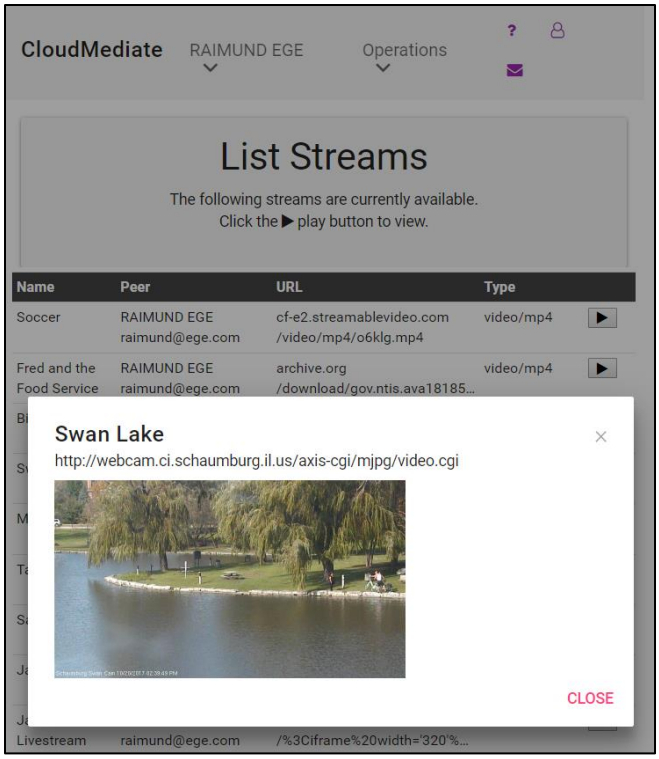

Figure 6. List streams with stream preview

All available streams are shown with their name, URL and type information, as well which peer is contributing the stream. Each stream can be viewed directly in a modal sub dialog.

The "Mediation control" selection from the operations menu opens up the central feature of CloudMediate where

peers can select and configure mediators. Figure 7 shows the initial screen:

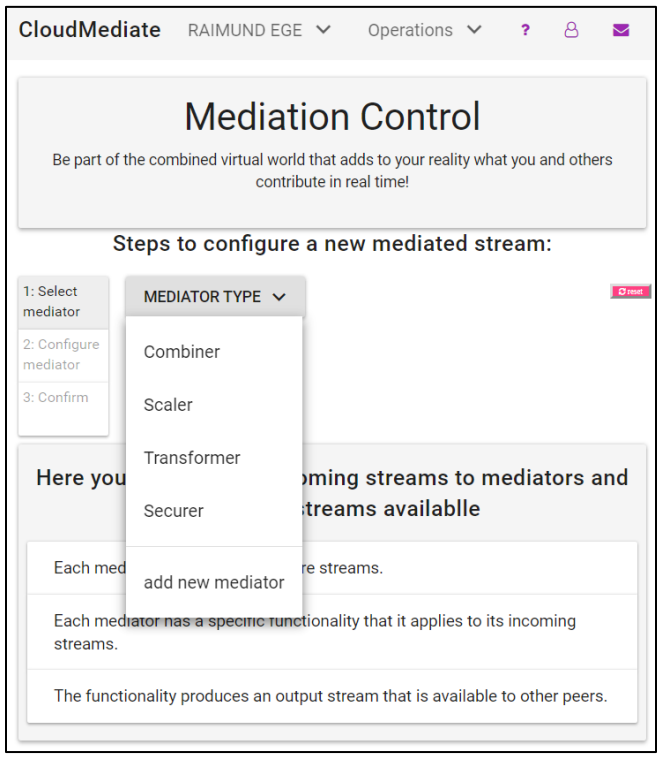

Figure 7. CloudMediate Mediation Control Component

Three steps are necessary to configure a mediator:

- Select an existing mediator: the "MEDIATOR TYPE" dropdown shows which types of mediators are currently available (the dropdown also allows to add new mediators); upon selection of the type, all available mediators are shown; the peer can select a specific mediator, which pulls up the mediator configuration component (see Figure 8).
- Configure the mediator: depending on the type it will require the specification of one or more input streams. The "Overlap" mediator shown in Figure 8 requires 2 input streams: each can be selected from the list of available streams. The parameters and the exact URL for the out stream also need to be specified.
- Confirm the submission of the configured mediator into the list of available streams in CloudMediate. The mediator's output streams are stored alongside other multimedia streams in the Firebase real-time database and appear immediately in the list of available streams for all peers connected to CloudMediate.

The dropdown menu also featured the "add new mediator" choice. It allows a peer to register a brand-new mediator type. The new mediator is configured via the "AddNewMediator" Angular component.

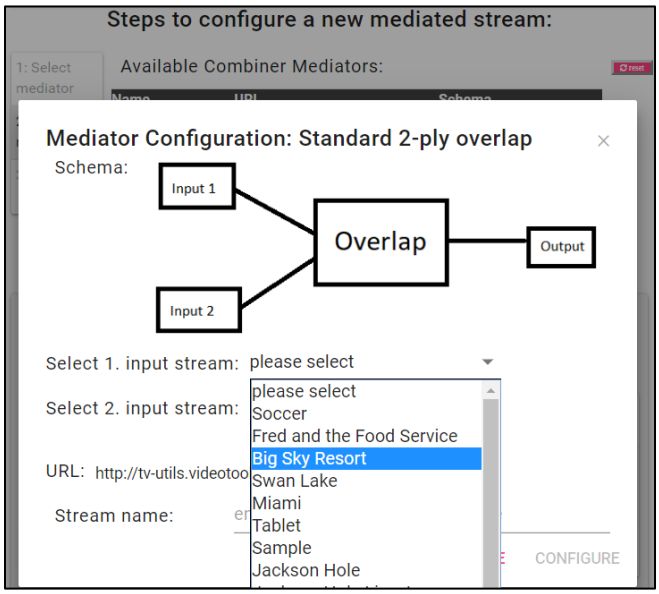

Figure 8. Configure Mediator

A new mediator is specified by giving it an expressive name, specifying the URL of the host and port where the mediation service is provided. Each new mediator needs to fall into one of the top-level mediator categories: "Combiner," "Transformer" or "Splitter." The mediation capabilities of the new mediator type are given via a mediator schema file.

Figure 2 showed the Angular component "AddNewMediator." Figure 9 shows the screen appearance of the component:

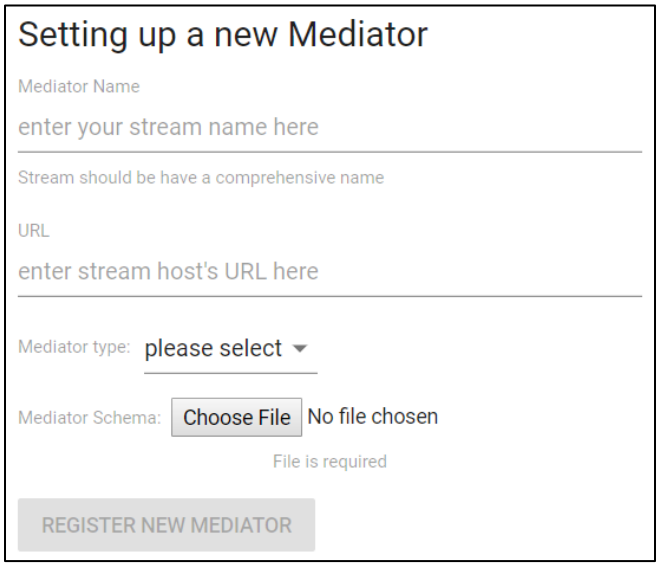

Figure 9. Add New Mediator Component

The new mediator that is configured via this component is again added to the Firebase JSON database stream list, and is instantly available to other peers.

 ISBN: 978-1-61208-607-1

V. CONCLUSION

CloudMediate is a peer-to-peer content sharing framework. In addition to sharing multimedia streams, it also allows to modify streams via mediators. Streams can be combined, split and transformed. Peers log in to CloudMediate add or select streams, configure mediators and add them to the sharing network, or just consume, i.e., view streams on their wearable devices.

In this paper, we highlighted our implementation of the CloudMediate framework using the Google Firebase cloud service. Firebase hosting, authentication and real-time database were used and allowed an efficient and scalable web portal that is available everywhere. The current CloudMediate incarnation is available at htttp//mediate.ege.com

One feature of Google Firebase that we did not use is the ability to program webservices that run within the Firebase back end. We plan to investigate the "Firebase Functions" feature (which is currently in beta status) to provide basic stream handling that would enable easier provision of more sophisticated mediator types.

REFERENCES

[1] R. Ege, "CloudMediate: Peer-to-peer Media Aggregation for Augmented Reality," The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2017), Athens, Greece, pp. 88-92, 2017.

[2] Google Firebase: Mobile and Web Application development Platform, https://firebase.google.com. [retrieved October 20, 2017]

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[4] S. Aukstakalnis, Practical Augmented Reality, Addison-Wesley Professional, 2017.

[5] R. Ege, "Secure Trust Management for the Android Platform," International Conference on Systems (ICONS 2013), Seville, Spain, pp. 98-103, 2013.

[6] R. Ege, "Peer to Peer Media Management for Augmented Reality," International Conference on Networking and Services (ICNS 2015), Rome, Italy, pp. 95-100, 2015.

[7] R. Azuma et al., "Recent Advances in Augmented Reality," IEEE Computer Graphics and Applications (CGA) 21(6), pp. 34-47, 2001.

[8] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-Time Detection and Tracking for Augmented Reality on Mobile Phones," IEEE Transactions on Visualization and Computer Graphics, 16(3), pp. 355-368, 2010.

[9] A. Morrison et al., "Collaborative use of mobile augmented reality with paper maps," Journal on Computers & Graphics (Elsevier), 35(4), pp. 789-799, 2011.

[10] Heroku: Platform as a Service, https://www.heroku.com. [retrieved October 20, 2017]

[11] Angular: Web Application Framework, http://angular.io/. [retrieved October 20, 2017]

[12] WebM: an open web media project, http://www.webmproject.org. [retrieved October 20, 2017]

 ISBN: 978-1-61208-607-1