Multimedia Conferencing at the Network Edge

Ivaylo I. Atanasov[1] and Evelina N. Pencheva[2]
[1]Technogrid Ltd, Sofia, Bulgaria
[2]Todor Kableshkov University of Transport, Sofia, Bulgaria
Email: i.i.a@abv.bg; evelina.nik.pencheva@gmail.com

Abstract —The fifth generation (5G) mobile system provides enhanced data rates, ultra-low latency, and better network resource usage for Internet of Things (IoT) devices in a dense environment. The integration of 5G IoT and multimedia conferencing enables collaboration between humans and devices by launching new capabilities. In this paper, an approach to define Application Programming Interfaces (API) for multimedia conference control is presented. The API design follows the principles of Multi-access Edge Computing (MEC) and features the inherent advantages of MEC. The API is described by resource structure, data types, and sequence diagrams that illustrate the API functionality. Some implementation issues are discussed, and the injected latency is evaluated.

Index Terms—Fifth generation, internet of things, multi-access edge computing, application programming interfaces

I. Introduction

The fifth generation (5G) Internet of Things (IoT) technology promises to improve all areas of our lives facing the growing needs of the networked society. The new benefits to the IoT brought by 5G include the ability to support a big number of IoT devices, both static and mobile, with a diverse range of quality of service requirements [1]-[3]. The 5G provides powerful combination of high speed, low latency, ubiquitous coverage, high reliability, and increased energy efficiency, which enables distance control in use cases, where the network efficiency is crucial. Edge computing and artificial intelligence may help to manage the data volumes generated by IoT devices, increasing network capacity. These data can be analyzed in real time to fasten decision making.

The 5G stimulates innovations in IoT and creates new ways for engagement of end users. Key factors for the success are the capabilities to create new services and to shorten time to market [4]-[6]. The integration of multimedia conferencing and IoT is one of the exciting new 5G use cases. Multimedia conferencing has been used for collaboration between humans for a long time, but the integration with IoT launches new capabilities. In office environment, it is possible to look for information during active conference, to issue voice commands, to ask questions and to receive instant answers. To improve efficiency and performance, IoT enables interconnection between databases, conference rooms, white boards, video conferencing devices and other tools for collaboration. For example, digital screens with multimedia conference connection may be connected to any object or product, making it source of information and platform for video communications. The integration of IoT and conferencing, combined with edge computing and artificial intelligence, can be beneficial in emergency services to improve health care and public safety [7]-[9]. Artificial Intelligence (AI) applications may be used to trigger conference calls with the interested parties automatically and to support multiparty communication sessions e.g., with end users and health service providers.

There are a lot of proprietary conferencing solutions, such as Click Meeting, Zoom Meetings, GoToMeetings, Skype for Business, Google hangouts, etc. [10]. Telecom operators may provide multimedia conferencing as an Internet protocol Multimedia Subsystem (IMS) service [11]. In this paper, we propose a new RESTful service for multimedia conferencing that may be deployed in the vicinity of end users exploiting the benefits of Multi-access Edge Computing (MEC) technology. MEC makes possible the development of IoT applications with requirements of location and context awareness, mobility support, deployment independence and high quality of service [12], [13]. The proposed service allows applications in IoT devices and human controlled terminals to initiate a multimedia conference, to add and remove conference participants, and to receive information about conference participant status. The service does not require IMS deployment and exposes low latency, more efficient bandwidth management and high reliability characteristics of MEC.

Next sections focus on the research motivation, describe the multimedia conference service functionality by use cases, define the service Application Programming Interfaces and related data types, and discuss some implementation aspects.

II. Research Motivation

Multimedia conferencing platforms have been used for years for communication and collaboration between people, but the integration with IoT devices offers huge potential for innovative services and applications. Some examples include healthcare management, improvement

Manuscript received August 20, 2021; revised January 12, 2022.
The research is part of project grant KP-06-H37/33, funded by Bulgarian National Science Fund.
Corresponding author email: i.i.a@abv.bg.
doi:10.12720/jcm.17.2.117-124

of quality of service, inventory management, security and data generation. Virtually, any device and anyone connected to the Internet generates data. These data can be analyzed in near real time and thus can improve the decision-making process by integrating with AI technologies.

An example is video conferencing between people and the need for access to information. The use of devices with embedded AI capabilities, which are activated by voice commands, provides immediate access to the information without interrupting the conversation. AI for visualization of emotional sensation can be used to analyze the emotions of conference participants. With the help of multimedia conferencing and IoT, health monitoring of patients, elderly and people with disabilities can be improved. When diagnosing an emergency medical situation, multimedia conferencing can help saving lives.

To provide unified interoperable platform for multimedia conferencing, telecom operators deploy IMS Multimedia Telephony standard which offers converged, mobile and fixed, real-time multimedia communications. IMS simplifies the integration of cloud services, but it is a complex and heavy platform, normally located in the core network.

In many use cases, the latency is a critical factor. The latency is affected by the speed and available bandwidth, the number of hops in the data path, the interconnections, and the distance between the client device and the server. Sending data for analysis to the cloud and returning a response may take time, which is not acceptable for mission critical applications. MEC provides a distributed environment for running cloud applications in close proximity to end users and thus reduces delays. MEC enables localized and faster processing, and optimizes network bandwidth usage. Integration of multimedia conferencing with IoT, AI and Edge computing may improve the efficiency and effectiveness of communications between connected cars and industrial automation which require highly reliable and low latency network connections. MEC based multimedia conferencing may improve the quality of experience for end users of high-bandwidth applications of augmented and virtual realities.

The research motivation is to exploit the MEC benefits to provide distributed multimedia conferencing functionality close to end users without the use of IMS.

III. Service Information Structure

A. Resource Structure

The proposed service is named Multimedia Conference Control (MCC) service. The MCC service enables mobile edge applications to create a uniquely identified “context” to connect participants. A participant is a party involved in the conference and may be any object with conferencing capabilities. There may be a participant with special privileges, named hereafter conference owner, who can end the conference call or be the charged party. Using the MCC Application Programming Interface (API), a mobile edge application can add or remove conference participants.

The MCC API follows the REST (REpresentational State Transfer) style for developing distributed applications. REST is suitable for IoT applications as it is simple, based on resources, whose status may be managed using HTTP.

Fig. 1 shows the resources supported by the MCC service.

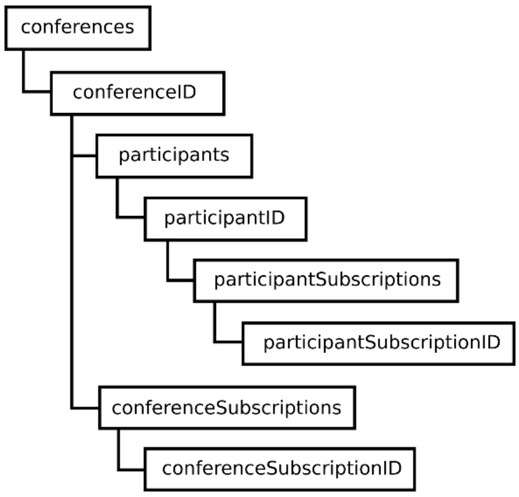

Fig. 1. MCC service resources.

The *conferences* resource represents all conferences created by the MEC applications. To create a new conference the application invokes HTTP POST method on the *conferences* resource where the method body contains conference information details. The MCC service creates a new multimedia conference with no participants initially and answers with an HTTP 201 Created response. The response body contains the URI (Uniform Resource Identifier) of the *conferenceID* resource representing the just created conference. An HTTP GET method invoked on *conferences* resource retrieves a list of all application created conferences or information about existing conference, respectively. The application can update an existing conference by sending an HTTP PUT method on the respective *conferenceID* resource.

The *participants* resource represents conference participants. To add a conference participant, the application invokes an HTTP POST method on the *participants* resource providing the participant URI and media to be used. In case the participant joined the conference successfully, the MCC service returns an HTTP 201 Created response with the identification of the added participant stored at the *participantID* resource. The application uses an HTTP GET method on the *participants* resource or on the *participantID* resource to get the conference participant list or information about existing conference participant, respectively. The application can update the media used by the participant by sending an HTTP PATCH method on the corresponding *participantID* resource. The application can remove the participant from the conference by sending an HTTP DELETE method on the resource representing the participant.

The conference termination may be due to expiry i. e. reaching the maximum duration included as an optional parameter in the request for conference creation. The application uses an HTTP GET request on the *conferenceID* resource to retrieve the conference status (initial, active, or terminated). The conference termination may occur due to leaving the conference by all the participants. To be notified about such events, the application needs a subscription. The *conferenceSubscriptions* resource represents all subscriptions to notification regarding termination of existing conference, while the *conferenceSubscriptionID* resource identifies an existing subscription. The application can create a subscription using an HTTP POST method on the subscription resource. It can also terminate given subscription by invoking an HTTP DELETE method on the corresponding *conferenceSubscriptionID* resource. Similarly, the *participantSubscriptions* resource represents all subscriptions for notifications about the status of the conference participant, and the *participantSubscriptionID* resource represents such subscription. The HTTP POST method used to create a subscription includes the callback address where the application wants to receive the notifications. In case of change of the participant status (e. g. participant media is on hold or the participant leaves the conference), the MCC service notifies the application.

Table I summarizes the MCC service resources and supported HTTP methods. All MCC service resources follow the service URI which can be published to a service directory.

B. Service Data Model

In this subsection, we describe the data types used in data structures that are exchanged over the MCC API.

TABLE I: CONFERENCING RESOURCES AND APPLICABLE HTTP METHODS

Resource URI	HTTP method	Description
/conferences	GET	Retrieves list of all multimedia conferences created by applications.
	POST	Creates a new conference.
/conferences/conferenceID	GET	Retrieves information about existing multimedia conference.
	PUT	Modifies existing conference.
	DELETE	Terminates existing conference.
/conferences/conferenceID /participants	GET	Retrieves list of all conference participants.
	POST	Adds a new participant to the conference.
/conferences/conferenceID /participants/participantID	GET	Retrieves information about existing conference participant.
	PATCH	Updates existing information about existing conference participant.
	DELETE	Removes conference participant.
/conferences/conferenceID /conferenceSubscriptions	GET	Retrieves list of all conference subscriptions.
	POST	Creates a new conference subscription.
/conferences/conferenceID /conferenceSubscriptions/ conferenceSubscriptionID	GET	Retrieves information about existing conference subscription.
	PUT	Updates existing conference subscription.
	DELETE	Deletes existing conference subscription.
/conferences/conferenceID /participants/participantID/ participantSubscriptions	GET	Retrieves list of all participant subscriptions.
	POST	Creates a new participant subscription.
/conferences/conferenceID /participants/participantID/ participantSubscriptions/ participantSubscriptionID	GET	Retrieves information about existing participant subscription.
	PUT	Updates existing participant subscription.
	DELETE	Deletes existing participant subscription.

The *ConferenceData* data type contains information about the conference, and it is a structure of the conference status (initial, active, terminated), the time at which conference is created, time at which the conference has at least one participant, the conference duration, the conference owner, the current number of participants connected to the conference, the maximum number of allowed participants, the call session identifier, and conference description.

The *participantData* data type contains information about existing conference participant and it includes participants address, media information which is a structure of used media and its status, the time at which the participant has joined the conference and the participant status (invited, connected, and disconnected).

The *confSubscription* data type represents a subscription to conference status changes. It is a structure of the callback address provided by the application where the application wants to receive the notifications about conference status, and the application identification.

The partSubscription data type represents a subscription to conference participant status and contains the callback address and the applicationID.

An example of simplified MEC application request to retrieve the conference status and the respective response with JSON description is as follows:

```
GET /exampleAPI/mcc/v1/conferences/DABEDABEDA HTTP/1.1
Host: example.com
Accept: application/json
Content-length: 0

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 279

{"conferenceData":{
"status":"active",
"startTime":"2021-03-23T12:15:03.521Z",
"activeTime":"2021-03-23T12:16:05.321Z",
"duration":1200,
"owner":"ivan.petrov@example.com",
"maxParticipants":5,
"currParticipants":3,
"callSessionID":"AABBCCDD",
"conferenceDescription":"climate"}
}
```

An example of simplified notification about the conference participant status change with JSON description is as follows:

```
POST /notificationApp/BADDAD HTTP/1.1
Host: example.com
Cache-Control: no-cache
Content-Type: application/json
Content-Length: 196

{"participantData":{
"participant":"alex.dimov@example.com",
"mediaInfo":["data":"active","video":"onhold"],
"timestamp":"2021-03-23T12:16:06.447Z",
"status":"connected",
"prevStatus":"invited"}

HTTP/1.1 204 No content
```

IV. Discussion on Service Implementation Issues

A. *Mapping API onto Network Operations*

The MCC service may be deployed at a MEC server that is co-located with distributed, virtualized core network functions. As a mediator between MEC applications and the core network, the MCC service must translate MCC API methods onto service operations of Network Exposure Function (NEF) which provides secure and robust access to exposed network capabilities and services. The NEF services and operations are defined in [14].

Fig. 2 shows the message flow for creating a conference by MEC application. The MCC service may reserve resources for the conference in the network (not shown in the figure). The application subscribes to notifications about conference status.

Fig. 3 shows the flow of adding of conference owner to existing conference by MEC application.

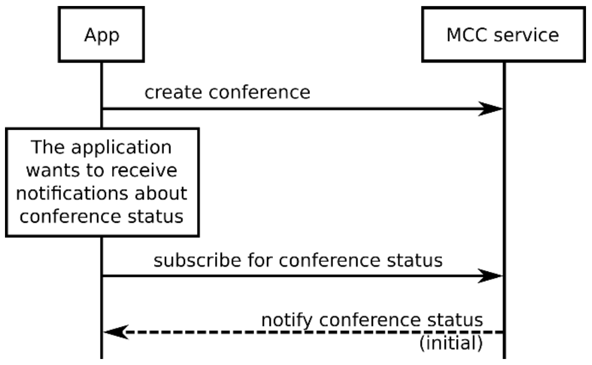
App
MCC service
create conference
The application wants to receive notifications about conference status
subscribe for conference status
notify conference status (initial)

Fig. 2. Flow of conference creation initiated by an application.

When the application requests to add a participant to existing conference as a conference owner, the MCC service invokes Nnef_TrafficInfluence_Create operation of the NEF to trigger the session initiation with the participant. The NEF invokes the Nnef_TrafficInfluence_Notify operation to notify the MCC service that the participant is invited to the conference session. The MCC service invokes the Nnef_ChargeableParty_Create operation to indicate the conference chargeable party. The application subscribes to receive notifications about participant status. When the participant is connected to the conference, the NEF notifies the MCC service, which in turn sends a notification of the participant status to the application.

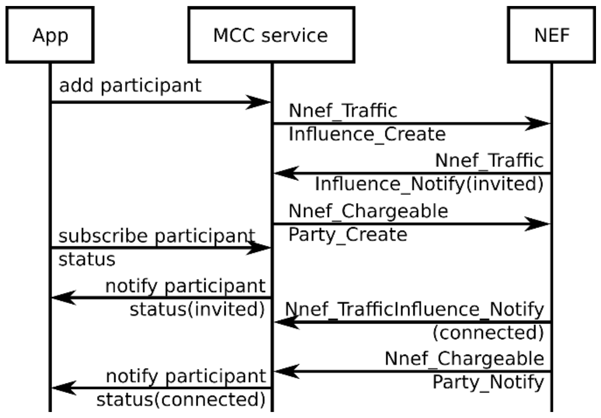
App
MCC service
NEF
add participant
Nnef_Traffic Influence_Create
Nnef_Traffic Influence_Notify(invited)
Nnef_Chargeable Party_Create
subscribe participant status
notify participant status(invited)
Nnef_TrafficInfluence_Notify (connected)
Nnef_Chargeable Party_Notify
notify participant status(connected)

Fig. 3. Flow of adding a conference participant by an application.

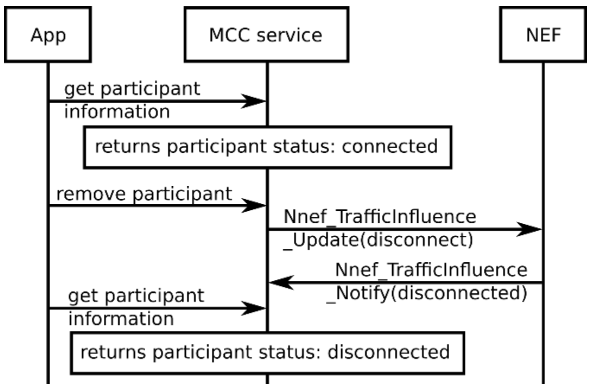
App
MCC service
NEF
get participant information
returns participant status: connected
remove participant
Nnef_TrafficInfluence _Update(disconnect)
Nnef_TrafficInfluence _Notify(disconnected)
get participant information
returns participant status: disconnected

Fig. 4. Flow of removing a conference participant by an application.

Fig. 4 shows the flow of participant disconnection initiated by the MEC application. The application requests for the conference participant status and decides to remove the participant. The MCC service invokes Nnef_TrafficInfluence_Update operation to trigger the participant disconnection in the network and then NEF notifies the MCC service when the participant is disconnected.

When a participant, who is not the owner of the conference, leaves, the NEF notifies the MCC service that the respective participant is disconnected, and the application, having an active subscription for participant status changes, is also notified by the MCC service.

Fig. 5 shows the flow of conference termination in case the conference owner leaves. When the participant, which is the conference owner, leaves, the NEF invokes Nnef_TrafficInfluence_Notify operation to notify the MCC service and invokes Nnef_ChargeableParty_Notify operation also to report the chargeable event. The MCC service in turn initiates conference session termination with the other participants (Nnef_TrafficInfluence_Update, Nnef_TrafficInfluence_Notify operations). If the application has an active subscription for conference state changes, it is notified by the MCC service, when all the participants are disconnected.

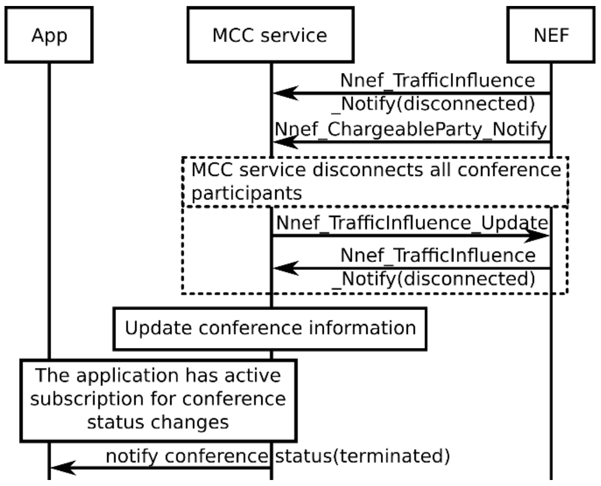
App
MCC service
NEF
Nnef_TrafficInfluence _Notify(disconnected)
Nnef_ChargeableParty_Notify
MCC service disconnects all conference participants
Nnef_TrafficInfluence_Update
Nnef_TrafficInfluence _Notify(disconnected)
Update conference information
The application has active subscription for conference status changes
notify conference status(terminated)

Fig. 5. Flow of conference termination in case the conference owner leaves.

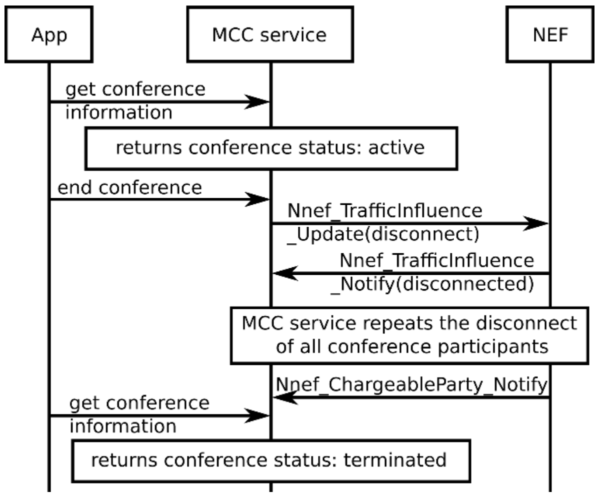
App
MCC service
NEF
get conference information
returns conference status: active
end conference
Nnef_TrafficInfluence _Update(disconnect)
Nnef_TrafficInfluence _Notify(disconnected)
MCC service repeats the disconnect of all conference participants
Nnef_ChargeableParty_Notify
get conference information
returns conference status: terminated

Fig. 6. Flow of application initiated conference termination.

Fig. 6 shows the flow of application initiated conference termination. The application retrieves information about the conference and requests for the conference termination. The MCC service initiates the conference disconnection. When all the participants are disconnected, the NEF notifies the MCC service about the chargeable event. If the application requests for some conference information, the MCC service returns terminated status.

B. Service State Models

The application view on the conference state and on the conference participant status must be synchronized with the MCC service view on both of them.

Fig. 7 shows the simplified conference state model as seen from the application point of view. The *Disconnected* state is the state in which the conference resource does not exist. The application can initiate a conference using the MCC service API. In *Initial* state, the conference resource is created but has no participant resources associated to it. The application can request a participant owner to be invited to the conference. When the conference owner joins the conference, the MCC service notifies the application, and the conference state becomes *Active*. In *Active* state, the application can invite another conference participant or remove given conference participant. The application can terminate the conference, or it can be notified about conference participant disconnection. The model is simplified as it does not show the subscription to notifications as well as the subscription termination.

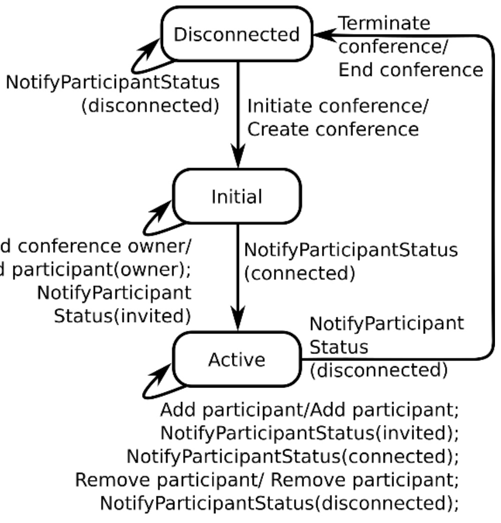
Disconnected
Terminate conference/ End conference
NotifyParticipantStatus (disconnected)
Initiate conference/ Create conference
Initial
Add conference owner/ Add participant(owner); NotifyParticipant Status(invited)
NotifyParticipantStatus (connected)
Active
NotifyParticipant Status (disconnected)
Add participant/Add participant; NotifyParticipantStatus(invited); NotifyParticipantStatus(connected); Remove participant/ Remove participant; NotifyParticipantStatus(disconnected);

Fig. 7. The application view on the conference state.

Fig. 8 shows the simplified conference state model as seen from the MCC service point of view. In *Idle* state, the conference is not created. In *Created* state, the conference is created with no participants. The application may request adding the conference owner as participant, which causes the MCC service to request session initiation and setting a chargeable party. In *InvitingConferenceOwner* state, the MCC service waits for notification from the network that the conference owner is connected. In *SettingChargeableParty*, the MCC service waits for notification from the network that the chargeable party is set. In *Active* state, the conference has at least one participant connected. In *InvitingParticipant* state, the MCC waits for notification from the network that the participant invited by the application is connected to the conference. In *RemovingParticipant* state, the MCC service waits for notification from the network that a conference participant, removed by the application, is disconnected.

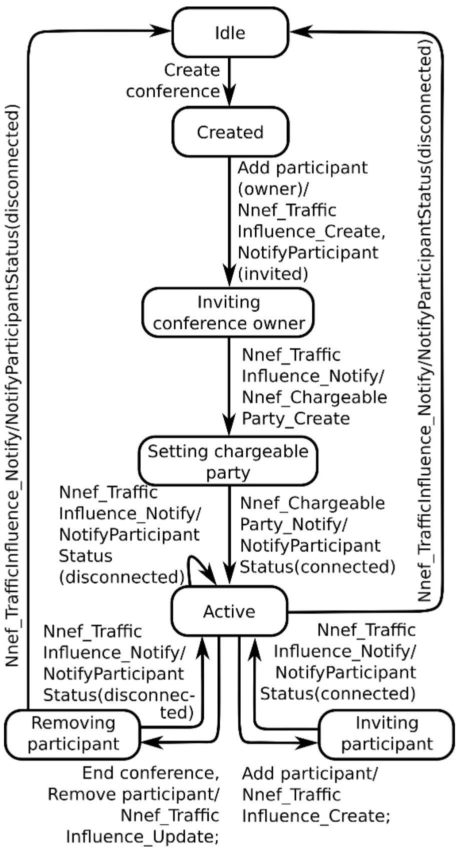
Idle
Create conference
Created
Add participant (owner)/ Nnef_Traffic Influence_Create, NotifyParticipant (invited)
Inviting conference owner
Nnef_Traffic Influence_Notify/ Nnef_Chargeable Party_Create
Setting chargeable party
Nnef_Traffic Influence_Notify/ NotifyParticipant Status (disconnected)
Nnef_Chargeable Party_Notify/ NotifyParticipant Status(connected)
Active
Nnef_Traffic Influence_Notify/ NotifyParticipant Status(disconnec-ted)
Nnef_Traffic Influence_Notify/ NotifyParticipant Status(connected)
Removing participant
Inviting participant
End conference, Remove participant/ Nnef_Traffic Influence_Update;
Add participant/ Nnef_Traffic Influence_Create;
Nnef_TrafficInfluence_Notify/NotifyParticipantStatus(disconnected)
Nnef_TrafficInfluence_Notify/NotifyParticipantStatus(disconnected)

Fig. 8. The MCC service view on the conference state.

The synchronization between conference state models may be proved by formal mathematical description and using the concept of weak bi-simulation.

To formalize mathematically the model description, the concept of Finite State Machine (FSM) is used. An FSM is formally described as a quadruple of a set of states, a set of events initiating transitions, a set of transitions between states and an initial state.

By $FSM_{app} = (S_{app}, E_{app}, T_{app}, s^0_{app})$ it is denoted an FSM representing the application view on the conference state, where:

S_{app} = {Disconnected [s_A^1], Initial [s_A^2], Active [s_A^3]};

E_{app} = {InitiateConference [e_A^1], AddConference-Owner [e_A^2], NotifyParticipantStatus (invited) [e_A^3], NotifyParticipantStatus(connected) [e_A^4], AddParticipant [e_A^5], RemoveParticipant [e_A^6], NotifyParticipantStatus (disconnected) [e_A^7], TerminateConference [e_A^8]};

$T_{app} = \{(s_A^1\ e_A^1\ s_A^2), (s_A^2\ e_A^2\ s_A^2), (s_A^2\ e_A^3\ s_A^2), (s_A^2\ e_A^4\ s_A^3), (s_A^3\ e_A^5\ s_A^3), (s_A^3\ e_A^3\ s_A^3), (s_A^3\ e_A^4\ s_A^3), (s_A^3\ e_A^6\ s_A^3), (s_A^3\ e_A^7\ s_A^3), (s_A^3\ e_A^8\ s_A^1), (s_A^3\ e_A^7\ s_A^1), (s_A^1\ e_A^7\ s_A^1)\}$;

$s^0_{app} = s_A^1$.

We use short notations for the names put in brackets.

By $FSM_{mcc} = (S_{mcc}, E_{mcc}, T_{mcc}, s^0_{mcc})$ it is denoted an FSM representing the MCC service view on the conference state, where:

S_{mcc} = {Idle [s_S^1], Created [s_S^2], InvitingConference-Owner [s_S^3], SettingChargeableParty [s_S^4], Active [s_S^5], InvitingParticipant [s_S^6], RemovingParticipant [s_S^7]};

E_{mcc} = {createConference [e_S^1], addParticipant(owner) [e_S^2], Nnef_TrafficInfluence_Notify [e_S^3], Nnef_Charge-ableParty_Notify [e_S^4], addParticipant [e_S^5], endConfe-rence [e_S^6], removeParticipant [e_S^7]};

$T_{mcc} = \{(s_S^1\ e_S^1\ s_S^2), (s_S^2\ e_S^2\ s_S^3), (s_S^3\ e_S^3\ s_S^4), (s_S^4\ e_S^4\ s_S^5), (s_S^5\ e_S^5\ s_S^6), (s_S^6\ e_S^3\ s_S^5), (s_S^5\ e_S^3\ s_S^5), (s_S^5\ e_S^6\ s_S^7), (s_S^5\ e_S^7\ s_S^7), (s_S^7\ e_S^3\ s_S^5), (s_S^5\ e_S^3\ s_S^1), (s_S^7\ e_S^3\ s_S^1)\}$;

$s^0_{mcc} = s_S^1$.

To prove that the application and MCC service views on the conference state are synchronized, we show that both models expose equivalent behavior using the concept of weak bi-simulation. The weak bi-simulation requires identification of tuples of states (s_A^i, s_S^j), i=1..3, j=1..7, such that for each transition sequence from a state in a tuple to a state in another tuple within the one FSM there exists a transition sequence from the corresponding state within the tuple to the corresponding state in the other tuple in the other FSM.

Proposition: FSM_{app} and FSM_{mcc} expose equivalent behavior, i. e. they have a bi-simulation relationship.

Proof: By $R = \{(s_A^1, s_S^1), (s_A^2, s_S^2), (s_A^3, s_S^5)\}$ it is denoted a relationship between the states of FSM_{app} and FSM_{mcc}.

Then the following transition sequences can be identified:

1. The application creates a conference: $\forall(s_A^1\ e_A^1\ s_A^2)\ \exists\ (s_S^1\ e_S^1\ s_S^2)$.
2. The application invites the conference owner to connect to the conference: $\forall\ (s_A^2\ e_A^2\ s_A^2) \sqcap (s_A^2\ e_A^3\ s_A^2) \sqcap (s_A^2\ e_A^4\ s_A^3)\ \exists\ (s_S^2\ e_S^2\ s_S^3) \sqcap (s_S^3\ e_S^3\ s_S^4) \sqcap (s_S^4\ e_S^4\ s_S^5)$.
3. The application invites a participant to connect to the conference: $\forall\ (s_A^3\ e_A^5\ s_A^3) \sqcap (s_A^3\ e_A^3\ s_A^3) \sqcap (s_A^3\ e_A^4\ s_A^3)\ \exists\ (s_S^5\ e_S^5\ s_S^6) \sqcap (s_S^6\ e_S^3\ s_S^5)$.
4. The application is notified that a conference participant (not the last one), leaves: $\forall\ (s_A^3\ e_A^7\ s_A^3)\ \exists\ (s_S^5\ e_S^3\ s_S^5)$.
5. The application is notified that the last conference participant leaves: $\forall\ (s_A^3\ e_A^7\ s_A^1)\ \exists\ (s_S^5\ e_S^3\ s_S^1)$.
6. The application removes a conference participant (not the last one): $\forall\ (s_A^3\ e_A^6\ s_A^3) \sqcap (s_A^3\ e_A^7\ s_A^3)\ \exists\ (s_S^5\ e_S^7\ s_S^7) \sqcap (s_S^7\ e_S^3\ s_S^5)$.
7. The application terminates the conference: $\forall\ (s_A^3\ e_A^8\ s_A^1) \sqcap (s_A^1\ e_A^7\ s_A^1)\ \exists\ (s_S^5\ e_S^6\ s_S^7) \sqcap (s_S^7\ e_S^3\ s_S^1)$.

Therefore, the R is a bi-simulation relationship, which means that FSM_{app} and FSM_{mcc} expose equivalent behavior.

The formal model verification is useful in the implementation process for testing the specified behavior against the observed one.

C. Assessment of Injected Latency

In order to estimate the latency, which might be added by the MCC API to the latency budget, we make an emulation, that is based on matured tools like Apache Cassandra and Eclipse Vert.x. As far as the API is RESTful, its configuration of the experiment follows the well-known client-server model, where the implementation of the client, that is supposed to generate the traffic, is based on Java. The front-end of the implementation is built on the Vert.x verticles, exchanging messages over the event bus, and the back-end is trusted on Cassandra, working in a single node mode.

The injected latency is assessed by offering traffic load, that is consisted of twenty thousand HTTP requests of POST type, corresponding to create operations.

Fig. 9 depicts the numerical results, shown in ascending order, where about 95% of the load is with latency under 6ms and averages to 2ms approximately, however the top 5% go beyond 1s. It's likely to conclude that the latency issue still remains an open question.

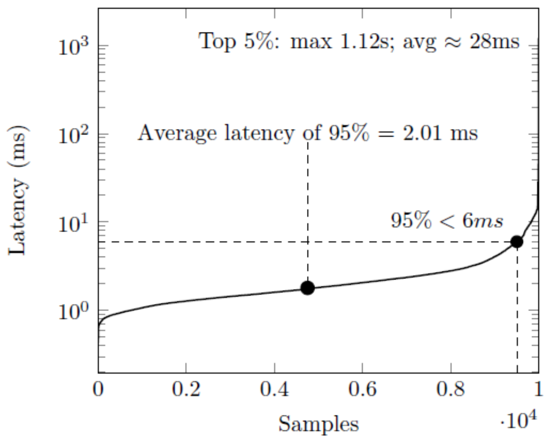

Fig. 9. Latency of the traffic as an ordered sequence.

V. Conclusion

The paper presents an approach to define a RESTful API for conference control by third party applications. The proposed MCC service enables applications deployed at the edge of the mobile network to create a conference, to add and/or remove conference participants and to subscribe to and receive notifications about conference state and conference participant status. The deployment of MCC API in the vicinity of end users benefits from low latency and more efficient network resource usage featured by edge computing. In addition to the conference control with human participants, the MCC API may be used to manage IoT devices as conference participants which creates opportunities for exciting new use cases.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

E. Pencheva contributed to API definition and the I. Atanasov conducted the API performance assessment.

References

[1] L. Chettri and R. Bera, "A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 16-32, Jan. 2020.

[2] B. Rong, *et al*., "Integration of 5G networks and internet of things for Future Smart City," *Wireless Communications and Mobile Computing*, 2020.

[3] P. Vagra, *et al*., "5G support for industrial iot applications – challenges, solutions and research gaps," *Sensors*, vol. 20 no. 828, pp. 1-43, 2020.

[4] S. Liu, L. Liu, H. Yang, K. Yue, and T. Guo, "Research on 5G technology based on Internet of things," in *Proc. IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Chongqing, China, 2020, pp. 1821-1823.

[5] Q. Wang, *et al.*, "Multimedia IoT systems and applications," in *Proc. Global Internet of Things Summit (GIoTS)*, Geneva, 2017, pp. 1-6.

[6] A. Karaadi, L. Sun, and I. Mkwawa, "Multimedia Communications in Internet of Things QoT or QoE?" in *Proc. IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Exeter, UK, 2017, pp. 23-29.

[7] F. Andriopoulou, T. Orphanoudakis, and T. Dagiuklas, "IoTA: IoT automated SIP-based emergency call triggering system for general eHealth purposes," in *Proc. IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Italy, 2017, pp. 362-369.

[8] J. Ferreira, J. N. Soares, R. Jardim-Goncalves, and C. Agostinho, "Management of IoT devices in a physical network," in *Proc. 21st International Conference on Control Systems and Computer Science (CSCS)*, Romania, 2017, pp. 485-492.

[9] X. Huang and N. Ansari, "Secure multi-party data communications in cloud augmented IoT environment," in *Proc. IEEE International Conference on Communications (ICC)*, France, 2017, pp. 1-6.

[10] Wondershare PDFelement. (2020). Top 5 Video Conferencing Solutions. [Online]. Available: https://pdf.wondershare.com/reseller/video-conferencing-solutions.html

[11] G. Mishra, S. Dharmaraja, and S. Kar, "Performance analysis of multi-party conferencing in IMS using vacation queues," in *Proc. IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, New Delhi, India, 2014, pp. 1-6.
[12] R. Zhu, L. Liu, H. Song, and M. Ma, "Multi-access edge computing enabled internet of things: Advances and novel applications," *Neural Computing and Applications, Springer*, vol. 32, 15313–15316, 2020.
[13] I. Atanasov, E. Pencheva, D. Velkova, and I. Asenov, "Multiparty call control at the network edge," *Elektronika Ir Elektrotechnika*, vol. 26, no. 5, pp. 39-49, 2020.
[14] 3GPP TS 23.502 Procedures for the 5G System (5GS); Stage 2; Release 16, v16.6.0, 2020.

Ivaylo Atanasov is born in Sofia, Bulgaria. He has received his MSc degree in Electronics and PhD degree in Communication networks from Technical University of Sofia (TU-Sofia). He has been awarded DSc degree in Communication networks in 2016, TU-Sofia. Since 2013, he is full professor and his scientific research area covers mobile networks, internet communications and protocols, and mobile applications.

Evelina Pencheva is with the Todor Kableshkov University of Transport, Sofia. She is born in Sofia. She has received her MSc degree in Mathematics from Sofia University "St. Kliment Ohridski" and PhD degree in Communication networks from TU-Sofia. She has defended her DSc thesis in 2014. Since 2010, she is full professor and her scientific research area covers multimedia networks, telecommunication protocols, and service platforms.