17th International Conference on Space Operations, Dubai, United Arab Emirates – 6-10 May 2023.

SpaceOps-2023, ID # 204

Openvocs, a light-weight Voice Communication System for Space Mission Control

Falk R. Schiffner[a]*, Anja Bertard[a], Michael Beer[b], Markus Töpfer[a]

[a] *German Aerospace Center (DLR e.V.), Space Operations & Astronaut Training, Berlin, Germany*, [falk.schiffner, anja.bertard, markus.toepfer]@dlr.de

[a] *German Aerospace Center (DLR e.V.), Space Operations & Astronaut Training, German Space Operation Center (GSOC), Oberpfaffenhofen, Germany*, michael.beer@dlr.de
* Corresponding Author

Abstract

Flawless voice communication is crucial for mission control. It is the most practical and precise way to transmit information, and further, it is the most natural way to communicate between humans. Today's voice communication systems are highly specialized solutions but inflexible in terms of adjustability. Furthermore, hard-wired solutions are often limited in terms of accessibility and very costly. Nowadays, space operations are usually international endeavors, with different space agencies and private enterprises involved. Distributed operation teams are working together within different missions. Therefore, voice communication interoperability between space mission control centers must be guaranteed. Our approach reduces the entry barriers, especially for smaller participants, simplifies distributed operations, and helps the space flight community to grow.

This paper describes an approach of a lightweight multi-party multi-conferencing voice communication system for space mission operations. The paper reports on the component-based implementation of the system and describe the user interface as well as core technologies. Our solution will be a pure software-based system. The developed system provides all necessary functionalities over a website and is usable via any device capable of running an off-the-shelf internet browser. There is no need for special devices, special client software, dedicated keysets, or separate networks. After successful authentication at the web server, the user has control over his roles and the role-dependent voice loop setup. In short, a potential user only needs a laptop, a headset, and an internet browser to connect to the system. This approach makes the system much more accessible to a broad range of users while enhancing maintainability and simplifying system deployment.

Next to the system description, we outline basic ideas behind individual components to showcase used design principles and patterns, especially in connection to operational and maintainability topics.

Moreover, we firmly believe that in today's globally connected world, where technologies of various domains melt together, that voice communication for mission control can largely benefit from technologies and methods of related domains, like speech telephony and video conferencing, and internet infrastructure.

Keywords: Voice Communication System, Space Mission Control, Speech Communication, Software Development

Acronyms/Abbreviations

ADAM	=	Audio Distribution Alternative Methodologies.
API	=	Application Programming Interface
DLR	=	Deutsches Zentrum für Luft- und Raumfahrt e.V. (German Aerospace Center)
ESA	=	European Space Agency
GSOC	=	German Space Operation Center
IP	=	Internet Protocol
LDAP	=	Lightweight Directory Access Protocol
MORABA	=	Mobile Raketenbasis (Mobile Rocket Base)
POC	=	Proof of Concept
PTT	=	Push To Talk
RBAC	=	Role-Based-Access-Control
SIP	=	Session Initiation Protocol
TCP	=	Transmission Control Protocol
TURN	=	Traversal Using Relays around NAT (network address translators)
VoCS	=	Voice Communication System

VoIP = Voice over Internet Protocols
WebRTC = Web Real-Time Communication

1. Introduction

Voice communication in space mission control is a strict, formal, and highly regulated way to communicate only domain-specific context to assigned groups. The speech signals are transmitted in specific audio channels comparable with widely known audio conferences. Because conducting a space flight mission is a tremendously complex task, several audio conferences must be held simultaneously. Unlike in classical telephony conferences, the audio channel is available 24/7 and called a voice loop [1]. Users with the correct allowance can enter or leave the voice loop as needed for their task. Typically voice loops are organized as communication groups with a dedicated purpose. For example, a voice loop to lead the whole team is named Flight Director, whereas a voice loop may be set up for the payload team to collaborate specifically within their topic area. Operators choose a voice loop to talk only in the context of the chosen channel. This way, the operator can reach out to other team members for specific issues, and the communication channel is not cluttered with information not relevant to the issue at that moment.

In addition, what separates voice communication from standard conferencing systems in a mission control environment is that users can have different sets of voice loops available and participate in parallel in multiple voice loops. Moreover, depending on their tasks, varying permission to talk or only listen to a voice loop sets this approach apart. Finally, this also reflects an abstract perspective on the organizational structure. Here, a multi-party multi-conferencing environment is created with users' adaptable participation, state selection, and predefined entry permission, connecting users, teams, and agencies over multiple locations.

Telecommunication systems, such as classical telephony, were used to transmit voice over distances. With ongoing advancements in technological development, not only voice but all kinds of data are transmitted over IP-based networks. Many communication providers have already changed their core network technology from circuit-switched to IP-based networks. Even if classical network infrastructure still exists and is kept in mind when developing a new system, future communications for mission control rooms need to consider a packet-oriented core design.

In recent years, space missions are increasingly multi-agencies and multi-companies endeavors. This unveils the urgent need to have a voice communication system that is highly adaptable and flexible to use. This starkly contrasts traditional systems, which are immovable and hard-wired solutions. Moreover, in today's interconnected world, where the internet penetrates and reaches all areas of daily life, new ways and concepts must be developed to reflect this trend.

When designing a future VoCS, one has to decompose the core functionalities of the system and virtually abstract them. This approach leads away from dedicated, highly redundant, and expansive hardware infrastructure, and in the end, a pure software-based VoCS emerges. In this paper, we present our result of this process, namely *openvocs*.

1.1 Past Work

This subsection provides a rough outline of past development circles. This is done to appreciate the efforts and unremitting adherence to the vision of a purely software-based voice communication system for space mission operations.

In 2014 the idea was announced at SpaceOps [2] and started as project ADAM (audio distribution alternative methodologies) at the German Aerospace Center (DLR e.V.). The first draft implementations were developed and tested (e.g., ESA workshop). This approach failed in 2016 because the open-source elements used could not provide the required performance. Three complete rewrites were necessary to tackle unforeseeable technical obstacles. At this point, the project was renamed *openvocs*. After years of development, the first complete functional system was rolled out in 2020. Two years later, in 2022, the first *openvocs v.1.0.0* (Beta) was internally released. Beginning in 2023, the integration process into the infrastructure of the GSOC and MORABA was started.

1.2 Structure of the paper

The following section presents a general overview of the system and its capabilities. Here, it will be explained how the architecture of the system is set. Further, insight into the system's scalability will be provided. Finally, the section closes with a subsection on the role-based access management implemented in the system.

The paper continues with a section on the implementation of the system. More precisely, it will start with a description of the front-end and present the user interfaces for the voice client and the RBAC client

(Role-Based-Access-Control). This is followed by a subsection on gateways and data transmission. Here, details on connection establishment and data transmission handling are provided. Further, the distinction between the signaling and media channels is made. This section closes with a subsection on the backend and audio signal processing. Here, the management of the micro-processes is explained, and insights into the audio signal processing are given.

The paper closes with a conclusion and an outlook on future work. Finally, we will present a superficial roadmap of our goals for 2023 and present an insight into our vision of future communication systems for space mission control.

2. General Overview of openvocs

This section explains the general structure of *openvocs* and presents the key features and capabilities of the system.

2.1 General Architecture

The system's general architecture can be subdivided into three parts: frontend, gateways, and backend. The core concept is based on the server – client structure. The core system runs on a Linux server, and all clients connect to the server. From there, the user interface is provided to the user. This means that no additional openvocs specific software needs to be installed on the user side (besides a standard web browser)

The frontend is mainly the access point for the end user to the system. Here the voice client and the RBAC client are placed. The voice client is the actual interface a mission control operator uses to communicate.

All interaction at the frontend side is further transmitted to the gateways. Here, two transmission channels have to be distinguished. The first transmission channel is the signaling channel. The data transmitted over this channel are switching operations and state changes. This information is later needed at the backend to orchestrate the processes. The signaling protocol in this system part uses web sockets for transmission. The second channel is the media channel. In this transmission channel, the actual voice signal is sent. Therefore WebRTC [3] and a TURN server [4] are used.

At the backend, the two transmission channels arrive. The information sent over the signaling channel is now used to acquire new users, audio mixers, and other services. The needed microservices are set up and configured according to state changes of the different voice loops. The voice data arriving via the media connection is further rooted to the respective voice loops transcoded and mixed. After this process, the media data is sent back to the user's frontend, as well as the information about the successful state change. With that, the user hears the correct mixed audio signal from the voice loops. The user interface is updated with the correct new state information about the voice loop configuration.

2.2 System Scalability and Redundancy

As said in the previous subsection, the backend orchestrates microservices. These services are, e.g., voice loop mixers or user mixers. The backend only runs the number of mixers currently needed and does not waste resources in holding microservices on reserve. For example, if a new user logs into the system, the backend receives a request for a new internal user and, with that, a new user mixer. This approach allows scaling up the system on demand and reduces the need to keep a vast amount of computational resources ready.

A second important aspect of scalability is the ability to interconnect several instances of openvocs (see Fig. 1). For example, two mission control centers run an instance of openvocs on their own but can share configurations and voice loops to connect distributed teams or even agencies. This is an addition to the core server–client structure a single openvocs instance has. In the interconnect structure, a user still connects to his “home server,” but via the connection to the second server, he is able to communicate in voice loops hosted on the far-end server.

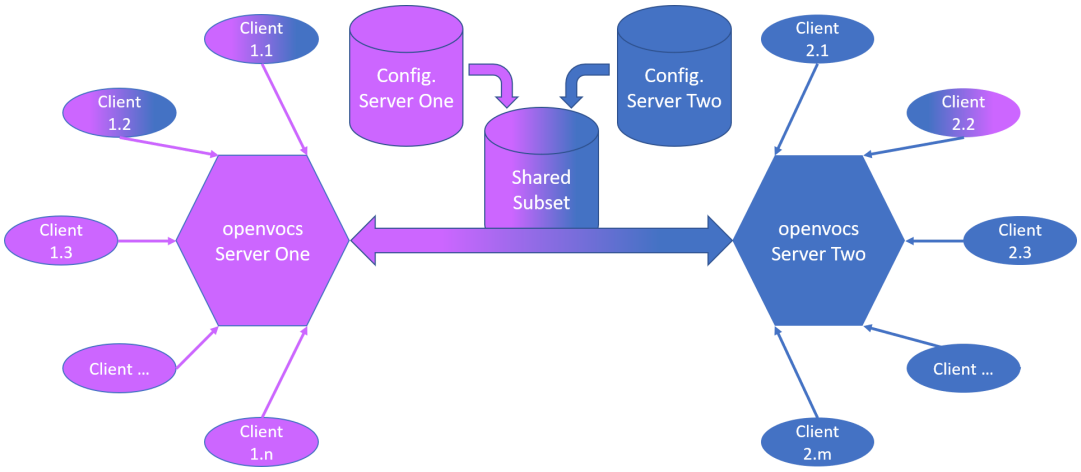

Fig. 1. General scheme of the interconnection of two openvocs instances. Several clients are connected to their server and in this case, Client 1.1 and Client 1.2 are able to communicate with Client 2.2.

System redundancy is a crucial aspect of any system used in high-risk areas. Therefore, we implemented a dual server setup. The system runs on a main server and mirrors all activity to a backup server. If a problem occurs on the main server, the system switches to the backup server to ensure interruption-free operation. When the main server connection is reestablished, the system switches back again. If a dual setup is not sufficient enough to ensure failure-free operation, more backup servers are thinkable.

Besides that, the system allows a user to login in with more than one device, e.g., the operator is logged in at a terminal and in addition via a tablet. If the user switches a state of a voice loop (e.g., switch off), this is automatically replicated on all devices the user is simultaneously logged in. This system feature allows an operator more freedom because he can leave this terminal, enter a meeting, and still be able to participate in all voice loops.

2.3 Role-based Access Management

The access management begins with the authentication at the user login. The system provides its own login client, where users can enter their credentials. These user accounts must be created by a system administrator. Further, openvocs provides a module that allows exchanging user information via LDAP [5] authentication if needed. This is necessary to reduce the need for additional user management at GSOC because all users already have an LDAP account. Nevertheless, if no LDAP user management is applied, the system uses its own authentication process.

The second part of access management is the assignment of different "roles." A role is independent of a specific person and can be assigned to multiple individuals. For example, the role "Flight Director" can be assigned to John Doe and Jane Doe because they work on different shifts or substitute each other in case of illnesses or similar.

No fixed numbers of roles are predefined because the number of roles in a space mission depends on the mission's needs. Each role has a predefined voice loop setup with various specialized voice loops. Further, some roles may have restrictions on some voice loops, meaning that they are only allowed to monitor this specific voice loop. So, the mission communication structure is defined on the role level and not on the individual. If a new operator enters a mission, he is assigned to a role and, with that, inherits the voice loop setup and talk/monitoring permissions. This drastically reduces administrative efforts.

3. Insights into the Implementation

3.1 Front End and User Interfaces

The frontend of openvocs lives in the browser and is written with classical web languages like HTML, CSS, and JavaScript. Modern browsers and current standard JavaScript APIs provide all tools necessary to implement a full-fledged VoCS user interface [6]. Our system, and especially the clients, are designed in a redundant way to,

amongst other things, account for the possibility that browser-based clients might not be as reliable as native client implementations.

Our system has both client redundancy as well as server redundancy. The client redundancy is achieved thanks to the server forwarding signaling messages to other clients of the same user. On the other hand, server redundancy is achieved because each client is commanding two or more servers simultaneously. The client sends each signaling request and audio stream to all connected servers. In this way, we create the same state on several independent servers.

Because we use role-based access control, users need not only to authenticate but also to authorize themself. Therefore after a successful login, the user is guided to a selection screen (comp. Fig 2).

Here, the user can select the mission or the role he wants. For example, it could be possible that an individual is part of multiple missions and have different roles and, with that, different permissions (e.g., flight director in mission A and engine engineering in mission B).

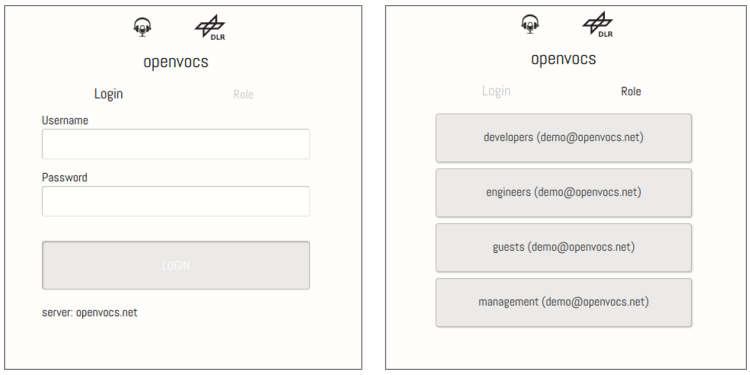

Fig. 2. Screenshots of authentication page (left) and mission / role selection page (right)

We designed the login and the voice interface with touch interfaces in mind. All buttons are big enough to use on touch interfaces easily. The interface of current VoCS keysets is heavily influenced by the design of the original VoCS keysets, which were operated with actual buttons. We also lean on the current well-known interfaces with our interface design but condense the design to save space. We, therefore, have no special command buttons but have integrated all commands in the loop buttons themselves (see Fig. 3). This also results in fewer clicks necessary to, for example, leave a voice loop.

Fig. 3. Voice client user interface, tile layout for voice loop selection, blue tile (monitoring) / green tile (talking)

The voice client in Fig. 3 contains a software PTT button (the long button on the bottom). The interface can be configured. The size of the loops, the number of loops per column and row, but also the method of PTT input can be configured. The best method is a secure PTT hardware button, which only allows speech input to reach the voice client when the PTT button is pressed. The significant advantage is that, in this case, the browser or browser tab does not need to be in focus to accept PTT input. The VoCS interface can be minimized and can still be usable. With all other methods, the VoCS interface needs to be in focus. If such specialized hardware is not available, it is also possible to configure the client to use a key on the keyboard or the software PTT button on the website to activate the speech transmission.

Next to the voice client interface, we also created an interface to manage the RBAC configuration (see Fig. 4) of openvocs. The RBAC interface is only accessible to users with system or project admin rights. It is a point-and-click interface to configure which user is in which role and which role has access to which voice loop. Our main intention in developing this interface was to allow authorized mission personnel to directly and instantly change the configuration of their mission VoCS configuration without relying on the system admin or needing any programming or administrative skills. Our interface uses a Sigiyama-styled graph visualization [7] to decipher the connections between users, roles, and voice loops. The user can select users, roles, and voice loops and edit their settings or via point and click change their connection to each other. In [8], we further described our RBAC interface and conducted a study to prove the usability of our visualization.

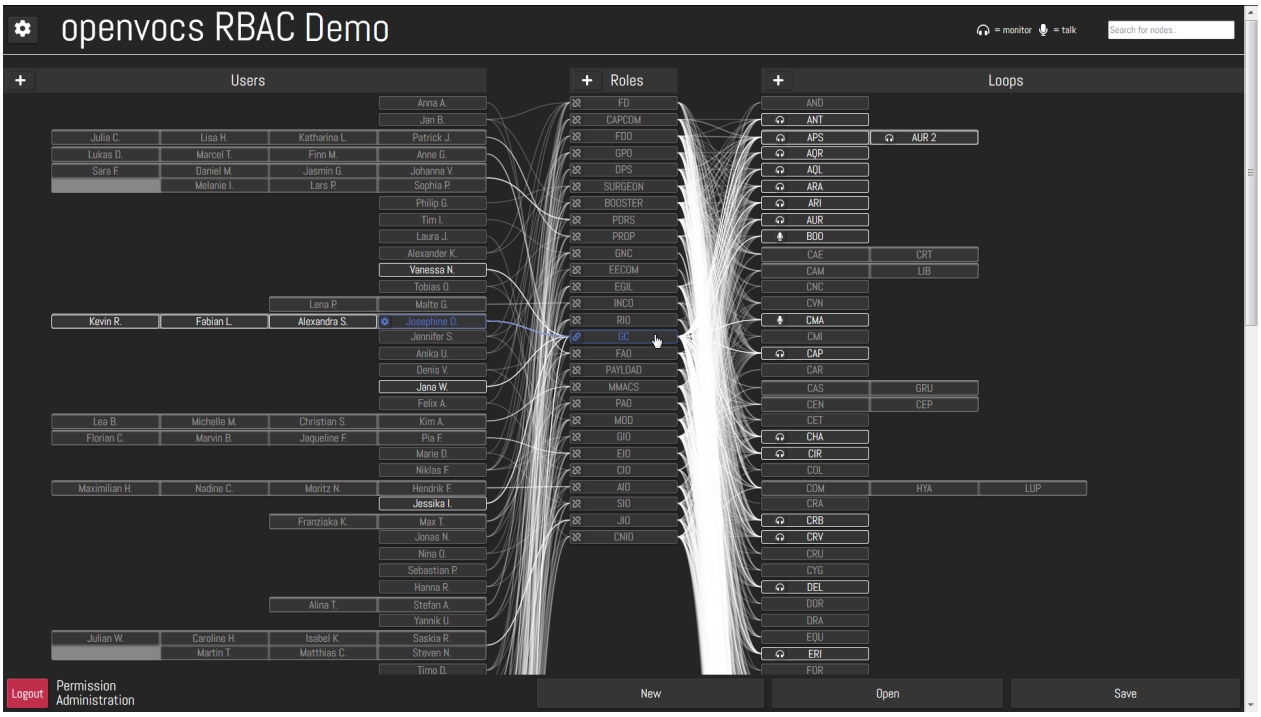

Fig. 4. RBAC client user interface, point and click interface with Sugiyama-styled graph layout
(The data we used is based on an older version of the Columbus Mission VoCS configuration, but is completely anonymized.)

3.2 Gateways and Data Transmission

Service provision for user endpoints is provided via gateways. A web gateway is used to provide the web pages for the user interface as well as the signaling channel for state changes. Data is transmitted as HTTP [9] for webpage content and JSON over WebSockets for signaling content. We use the signaling protocol defined in [10] with messages for authentication-related data exchange, e.g., SWITCH_ROLE to switch to some specific role or LOGIN to login some user, as well as loop switching messages to switch the voice loops between the states NONE, MONITOR and TALK. Next to state switches for loops, Volume switching events are used to adapt the loudness level. The web gateway with its signaling channel is used to enforce permissions for Voiceloop access, whereas media transmission is done over some media gateway.

Actual voice transmission is implemented using ICE (Internet Connectivity Establishment [11]) support within the media gateway. This gateway provides a secure RTP channel for media transmission. SRTP credentials are transmitted over the signaling protocol. Media gateway and web gateway are tightly coupled. A dedicated media session will be created for each signaling connection. The connection parameters for the ICE protocol are exchanged additionally to switching and authentication messages via the signaling protocol.

In each of these sessions, the signaling, as well as the media, are bound to some user mixer. The media path between a web browser and a user mixer is a point-to-point connection via the media gateway. The user mixer is a dedicated mixing instance for the actual signaling connection.

The web gateway is connected to the audio backend, which is responsible for audio mixing. Each login to the web gateway acquires some user mixer for the connection and prepares a voice path between a web browser and a user mixer via the media gateway. The process is as follows. The user login will trigger the media path setup at the media gateway and exchange external media connection parameters between the webpage and the media gateway using WebRTC [3] based Session Descriptions. In addition, the web gateway acquires a new user mixer for this specific connection. Media path parameters will be exchanged between the webpage and media gateway for the external connection and the media gateway and audio backend for the internal path between the user mixer and media gateway. Once this process finishes, the client will be informed with MEDIA_READY, and the whole transmission chain will be established.

Events to switch loop states are forwarded between the web gateway and audio backend to switch the loops according to the user input. Loop switch events are processed internally within the audio backend to switch the transmission paths of the incoming audio path from the media gateway to the respective loops. The same applies to volume switches. When a client sends a SWITCH_LOOP request, the web gateway checks the client's permission for the voice loop. (see role-based access control Section 2.3) If the client is allowed to participate within the voice loop, the web gateway forwards the switching request to the audio backend to switch the voice loop matrix.

A third type of gateway is the interconnection gateway, which connects two openvocs instances. A predefined set of loops will be loaded, and according to this configuration, an internal user session will be created for each Voiceloop. Externally the voice loops connect via ICE protocol the same way a client connects to the system. The interconnection gateway includes a media gateway that may be configured as a client or server. The client connects to the server and requests voice loops via a CONNECT message. Messaging between client and server interconnect is done using JSON messaging over TCP [12].

Signaling connections between the client and server side require a reachable port. For Webclients, the port is 443 HTTPS for HTTP as well as WebSocket messages. Both protocols are multiplexed over a single port for simplified Firewall configurations. Interconnection gateways require one port per interconnection side. This port is configurable within the system.

Gateways and protocol design are the core components defining the lightweightness of the system. Above and beyond, lightweightness gateways define the security of the systems. All communication channels are encrypted, signaling via TLS [13], and media via DTLS-SRTP [14]. Next to transmission security, gateways enforce the RBAC mechanism for Voiceloop access.

Gateway and messaging design are strongly aligned with WebRTC and web-based messaging protocols. Nonetheless, the design is open to implementing any client instances capable of WebRTC and JSON over WebSocket messaging. Furthermore, the lightweight approach over a webpage eliminates all dedicated client requirements.

3.3 Backend and Audio Signal Processing

Each user sends their audio and expects to receive back a combined stream containing all audio streams he wants to listen to.

The backend’s core functionality is to accept all those incoming audio streams, mix them and redistribute them appropriately (usually not directly, but via gateways). Those audio streams are RTP [15] streams with the actual audio payload expected to be encoded with Opus [16] sampled at 48kHz and sent in 20ms chunks (i.e., the payload of one RTP frame is expected to contain 960 samples).

For controlling the stream mixing, the backend provides a signaling interface that provides 3 basic commands:

1. `acquire_user` to create a new user and allocate appropriate resources in the backend
2. `listen_on` to start mixing a loop into the user’s outbound audio stream
3. `talk_on` to start mixing a user’s inbound audio stream into a loop

The commands to reverse actions are: `release_user` , `listen_off` and `talk_off`.

The backend provides a TCP port to connect to and send these commands as openvocs JSON signaling messages. Internally, the backend generally comprises many relatively simple, specialized processes interacting via the network rather than heavy monolithic programs.

Key components are mixers: Each mixer accepts several audio input streams, mixes them into one single stream, and sends this stream to several destinations. Another component that comes in handy is a multiplexer, which is a component that accepts an audio stream, duplicates it, and sends it to several destinations.

A user speaks in several loops and listens in several (other) loops.

The backend models this in a three-tiered way:

1. Each loop has an associated mixer (A “loop mixer”) - users who speak on this loop will have their audio streams sent into this mixer.
2. Each user has an associated mixer as well (A “user mixer”)- If the user listens to a loop, the appropriate loop mixer will send its output stream into this user’s mixer. The user mixer will send its output back to the user’s client (probably via a gateway of some sort).
3. As each user can talk in several loops at once, its input stream must be sent to several loop mixers. Hence the incoming user’s audio stream is sent to a dedicated multiplexer first, which duplicates the stream sending it to all the loop mixers required. Here, it should be added that the backend allows this operation, but the formalized radio etiquette does not allow talking in several voice loops simultaneously. So, the Talk state allows only one voice loop per user simultaneously. Nevertheless, circumstances can arise where it is necessary to talk in more than one voice loop (e.g., in an emergency, the attention of all users may be required instantly). Therefore, we implement this optional functionality.

Another component, the resource manager, orchestrates all those components themselves. First, each component connects to the resources manager and registers itself. The resource manager will then perform some basic configuration of said components and command them as needed and provide the TCP port for external commands. Finally, if such a command is received, the resource manager translates that command into a sequence of steps, which are executed sequentially and will involve reconfiguring mixers and multiplexers.

The user/loop stream can be facilitated to process any audio stream by creating a virtual user for each stream and using this virtual user as a kind of “handle” to control the stream’s processing in the backend.

Aside from that core functionality, the backend also should be capable of

- archiving/replaying archived audio
- exchange audio streams with external systems

For archiving, recorders are provided. Currently, those are still in development, but they can receive one audio stream and write its data into a file or read audio data from a file and send it to a destination.

They also register with the resource manager and are controlled exclusively by the resource manager.

External audio can be exchanged via SIP [17]. The SIP - gateway provides a SIP trunk, which can be used to call into the openvocs or receive calls from the openvocs. Each call is tied to a loop, which it will listen to and be able to talk on.

Unlike other components, the SIP gateway is not controlled by the resource manager but uses the resource manager to configure the backend to its needs. A new backend user is created and switched onto the preconfigured loop for every call. The SIP gateway provides a command interface via a TCP port where, e.g., a management client can connect to it to configure the SIP gateway.

Internally, all audio is exchanged as RTP streams, encoded with Opus at a 48kHz sample rate and a length of 20ms per RTP frame. Using Opus introduces a substantial performance hit, especially for the senders of an audio stream, but is unavoidable because of restrictions of UDP [18], which RTP uses on the transport layer.

The echo cancellation is achieved by subtracting the users input audio stream from their output stream before sending it to a user mixer at the loop mixers.

4. Conclusion and Outlook

This paper reports on the development of a new lightweight voice communication system for space mission control. The presented system stands apart from other solutions as a mere software solution. It uses a Linux distribution as an underlying operations system. Further, it is built upon open and standardized web technologies and

protocols. The browser-based user interfaces (for the voice client and administration) make the system easy to use and independent from expensive specialized hardware. The system operates on a role-based access control scheme to reflect the necessary hierarchical command structure indispensable for space mission control. The system provides a high audio quality standard using state-of-the-art audio coding. Because *openvocs* is rooted in internet technology, the system operates in IP networks, and access over the internet is possible. This leads to easy and broad access to space mission control communication, especially if enterprises, universities, and private companies work together with space agencies on space flight missions.

4.1 Road Map 2023

The reported openvocs v.1.0.0 version lacks some components necessary for operational usage. These components are currently under development. More precisely, a module to connect openvocs with traditional landline telephony via SIP will be developed to allow incoming telephone calls to be patched into predefined voice loops and further enable operators to call specific telephony numbers if needed for participation. This will extend the usage of openvocs far beyond the first envisioned use cases. A second core component currently under development is the audio recorder. The system will be able to record predefined voice loops and enable an operator to replay recording directly in the voice client user interface. The completion of these two features is expected in the first half of 2023.

As said at the beginning, the integration process had already started in 2023 at the MORABA (DLR), and it is foreseen to equip entire control rooms at GSOC end of 2023.

4.2 Future Work and Long Term Vision

As said, the system development still needs to be completed. Further, the development of new features and long-term maintenance of the code base is planned.

As described in [19], a prototype chat system was developed at DLR, and a potential combination is envisioned. In a future iteration of a multimodal communication system for space mission control, a fusion of the two modalities, namely the voice channel and the text channel, could be targeted. More precisely, the vision enables the conversion of the written text in a chat loop to be played out on a voice loop via state-of-the-art Text-To-Speech algorithms and vice versa. So, spoken utterances unanswered in a voice loop can be transferred via Speech-To-Text algorithms into text and sent to the respective chat loop. Even though that is a future vision, the technologies are already used in different domains and applications. In today's world, where technologies from different areas are already starting to fuse, tweaking them and using them for space mission control seems promising.

Moreover, as video conferencing is regarded as the logical next step from telephony, adding a video communication channel on demand for specific events could be a useful and promising additional feature in a space mission control environment.

References

[1] E. S. Patterson, J. Watts-Perotti and D. D.Woods, *"Voice Loops as Coordination Aids in Space Shuttle Mission Control,"* Computer Supported Cooperative Work, Vol. 8, Kluwer Academic Publishers,12/1999, pp. 353-371.

[2] M. Töpfer, R. Kozlowski, *"An Implicit Voice Conferencing System,"* AIAA, SpaceOps 2014 Conferences, Pasadena, USA, May 2016.

[3] C. Jennings et al., *"WebRTC 1.0: Real-Time Communication Between Browsers,"* W3C Editor's Draft, Jan 2023.

[4] T. Reddy et al., *"Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN),"* Internet Engineering Task Force (IETF), RFC 8656, ISSN: 2070-1721, Feb 2020.

[5] J. Sermersheim, Ed. *"Lightweight Directory Access Protocol (LDAP) ,"* Network Working Group, RFC 4511, Jun 2006.

[6] A. Bertard, M. Töpfer, *"How web-based voice communication system clients can change operation in mission control,"* in SpaceOps 2021 Conference, Cape Town, South Africa, Mar. 2021.

[7] K. Sugiyama, S. Tagawa, M. Toda, *"Methods for visual understanding of hierarchical system structures,"* IEEE Trans. Syst. Man Cybern. 11(2), 109–125, URL: https://doi.org/10.1109/TSMC.1981.4308636, 1981.

[8] A. Bertard, J.-K. Kopp, *"Using Sugiyama-Styled Graphs to Directly Manipulate Role-Based Access Control Configurations,"*HCI International 2020 - Posters, CCIS 1224, pp. 1–8, Springer Nature, https://doi.org/10.1007/978-3-030-50726-8_53, Switzerland, 2020.

[9] M. Thomson, Ed.(Mozilla), C. Benfield, Ed. (Apple Inc.) *" HTTP/2 ,"* Internet Engineering Task Force (IETF), RFC 9113, Jun 2022.

[10] M. Töpfer, A. Sonnenberg and R. Kozlowski, *"OpenSource based Voice Communication for Mission Control,"* AIAA, SpaceOps 2016 Conferences, South Korea, 2016.

[11] A. Keranen, C. Holmberg and J. Rosenberg, *"Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal,"* Internet Engineering Task Force (IETF), RFC 8445, ISSN: 2070-172, Jul 2018.

[12] W. Eddy, Ed., *"Transmission Control Protocol (TCP),"* Internet Engineering Task Force (IETF), RFC 9293, Aug 2022.

[13] T. Dierks and E. Rescorla, *"The Transport Layer Security (TLS) Protocol Version 1.2,"* Network Working Group, RFC 5246, Aug 2008.

[14] D. McGrew and E. Rescorla, *"Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP),"* Internet Engineering Task Force (IETF), RFC 5764, ISSN: 2070-1721, May 2010.

[15] H. Schulzrinne et al. *"RTP: A Transport Protocol for Real-Time Applications,"* Network Working Group, RFC 3550, Jul 2003.

[16] J.-M. Valin (Mozilla), K. Vos (Skype Technologies) and T. Terriberry (Mozilla), *"Definition of the Opus Audio Codec,"* Internet Engineering Task Force (IETF), RFC 6716, ISSN: 2070-1721, Sept 2012.

[17] J. Rosenberg et al.,*"SIP: Session Initiation Protocol,"* Network Working Group, RFC 2543, Jun 2002.

[18] J. Postel, *"User Datagram Protocol,"* RFC 768, Aug 1980.

[19] F. Schiffner, M. Burr, *"Role-based Multi-Chat System for Space Mission Control,"* in SpaceOps 2023 Conference, Dubai, UAE, Mar. 2023.