Immersive Audio Application Coding Proposal to the SBTVD TV 3.0 Call for Proposals

Oliver Major
Ziad Shaban
Bernd Czelhan
Adrian Murtaza

CITE THIS ARTICLE
Major, Oliver; Shaban, Ziad; Czelhan, Bernd and Murtaza, Adrian; 2021. Immersive Audio Application Coding Proposal to the SBTVD TV 3.0 Call for Proposals. SET INTERNATIONAL JOURNAL OF BROADCAST ENGINEERING. ISSN Print: 2446-9246 ISSN Online: 2446-9432. doi: 10.18580/setijbe.2021.4. Web Link: http://dx.doi.org/10.18580/setijbe.2021.4

SET INTERNATIONAL JOURNAL OF BROADCAST ENGINNERING - SET IJBE V.7, 2021, Article 4, 9p.
© 2021 SET / ISSN Print: 2446-9246 | ISSN Online: 2446-9432

Immersive Audio Application Coding Proposal to the SBTVD TV 3.0 Call for Proposals

Oliver Major, Ziad Shaban, Bernd Czelhan, and Adrian Murtaza,
Fraunhofer Institute for Integrated Circuits (IIS)

Abstract— **In July 2020 the Brazilian Terrestrial Television System Forum (SBTVD) has issued a Call for Proposals (CfP) for their next-generation digital TV system called TV 3.0. Fraunhofer IIS and ATEME have proposed the MPEG-H Audio system, based on the open international standard ISO/IEC 23008-3, MPEG-H 3D Audio, as a candidate technology for the Application Coding component of SBTVD TV 3.0. The submitted proposal specifies a new Application Programming Interface (API) enabling applications to make use of the next-generation interactivity features of the MPEG-H Audio system.**

This paper provides a detailed description of the proposed API, as well as the submitted JavaScript implementation, and the architecture of the prototype system. Additionally, the paper outlines the proposed evaluation process demonstrating how the MPEG-H Audio system fulfills the TV 3.0 Application Coding requirements for 3D object-based immersive audio interaction and emergency warning information delivery.

Index Terms— **3D and Immersive Audio, Accessibility, Adaptation and customization of content, ATSC 3.0, Application Coding, API, Broadcast, Broadband, Emergency warning system, HTML 5, Hybrid Delivery, Immersive Sound, MPEG-H Audio, Next Generation Audio, Object-based broadcasting, Personalized Sound, SBTVD TV 3.0, Streaming**

I. Introduction

THE Brazilian Digital Terrestrial Television System Forum (SBTVD) has issued in July 2020 a Call for Proposals (CfP) seeking input for Brazil's next-generation Digital TV system under the name “TV 3.0 Project” [1]. The SBTVD Forum has established a detailed set of TV 3.0 requirements and use cases covering six system components (Over-the-air Physical Layer, Transport Layer, Video Coding, Audio Coding, Captions, and Application Coding). The CfP was divided into two phases: Phase 1 required an initial submission from proponents identifying the candidate technology and providing basic information, while in Phase 2 the proponents were expected to submit a full specification of the candidate technology as well as hardware and software solutions for the feature evaluation.

In response to the SBTVD TV 3.0 Call for Proposals, Fraunhofer IIS, ATEME, the Digital Broadcasting Experts Group (DiBEG) and the Advanced Television Systems Committee (ATSC) have jointly proposed the MPEG-H Audio system, based on the open international standard ISO/IEC 23008-3, MPEG-H 3D Audio [2], as the audio component. Additionally, Fraunhofer IIS and ATEME have submitted a proposal for the Application Coding component, specifying a new Application Programming Interface (API) fulfilling the requirements for 3D object-based immersive audio interaction and emergency warning information delivery using an interactive application.

MPEG-H Audio was already adopted in Brazil as part of the TV 2.5 Project to enhance the audio experience over ISDB-Tb with immersive and personalized sound and it is currently fully specified in the ABNT standards [3][4][5][6]. This has enabled broadcasters and content creators in Brazil to gain experience in advanced audio productions with MPEG-H. Having professional broadcast equipment from several major providers available has been essential for using the system in the existing ISDB-T broadcast infrastructure and consequently the MPEG-H Audio system can ensure a smooth transition from TV 2.5 to TV 3.0.

The MPEG-H Audio system was developed to allow highly efficient immersive audio transmission and new capabilities such as advanced accessibility, interaction, personalization and adaptation of audio to different usage scenarios, delivering the best possible experience and taking audio to the next level. A detailed technical description of the MPEG-H Audio system is provided in [7] and lessons learned during live broadcast of major events using MPEG-H Audio are described in [8].

The use of audio objects, usually in combination with channel-based audio, enables the viewers to interact with the content in new ways and create a personalized listening experience. The MPEG-H Audio metadata carries all the information needed to allow viewers to change the properties of audio objects by attenuating or increasing their level, disabling them, or changing their position in three-dimensional space. Additionally, the MPEG-H Audio metadata structures empower broadcasters to enable or disable interactivity options and to strictly set the limits to which extent a user can interact with the content.

One of the most important use cases for personalization is dialog enhancement. With MPEG-H Audio the dialog or commentary for a program is sent as an audio object and associated metadata. This allows the viewer at home to adjust the relative volume or “presence” of the dialog relative to the rest of the audio elements in the program. This simple case can be extended to offer two or more dialog objects with different languages or commentaries (e.g., biased

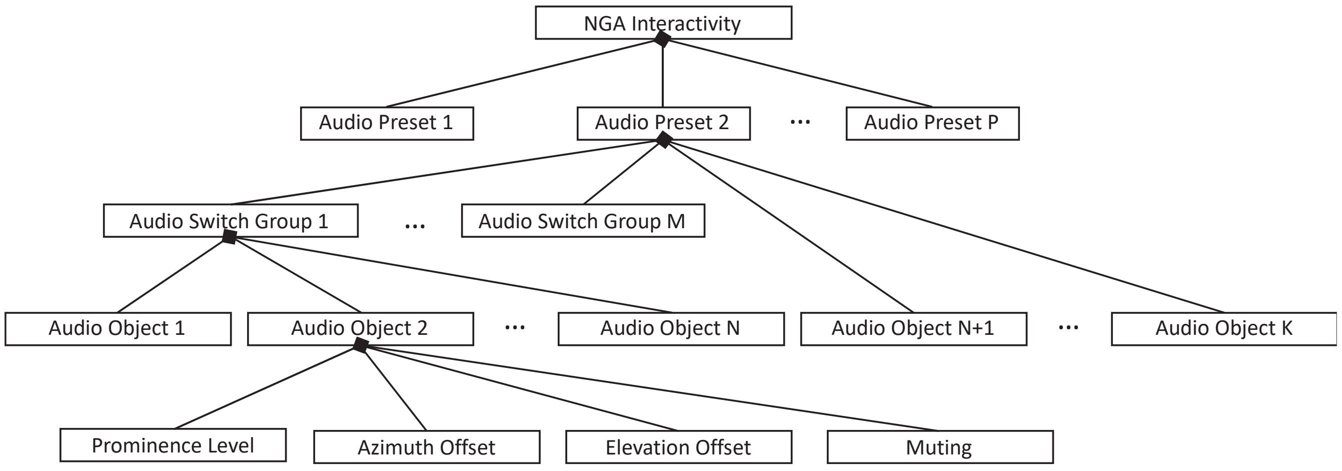

Fig. 1. NGAInteractivity overview.

commentators for each of the teams during a football game).

Moreover, the MPEG-H Audio metadata enables broadcasters to provide several versions of the content, as so-called "presets", which describe how all channels and object signals are mixed together and presented to the viewer. Choosing between different presets is the simplest way to interact with the content. Additionally, advanced interactivity settings can be offered to more experienced users for manipulating objects individually.

The MPEG-H Audio system standardizes a rich metadata set that enables the most advanced and flexible end-user interactivity and personalization experience, while still offering full control over these features to the broadcasters. The presence of metadata in the audio bitstream is extremely important for enabling the Application Coding layer to offer all these options to the user in a controlled and well-defined way.

In order to allow application developers to make use of these advanced audio features in broadcast applications, new APIs for interacting and controlling the rendering of the current audio scene are necessary. Therefore, new NCL (audio) properties have been proposed in [10]. The current document continues the work on new APIs for controlling advanced audio features and will describe the MPEG-H Audio System proposal for the Application Coding part of the SBTVD TV 3.0 Call for Proposal [1].

As part of this submission, a new JavaScript [14] API for controlling advanced audio features was defined. The submission also contains a prototype receiver device and prototype application, which practically show that the proposal fulfills the Application Coding requirements of the SBTVD TV 3.0 Call for Proposal [1].

In this paper, we present the API specification and explain the design principles which have been used. Additionally, the prototype implementation and the test results submitted for evaluation in the TV 3.0 Project will be described. The SBTVD TV 3.0 evaluation process of the proposal is supported by the proposed API and prototype. Lastly, we conclude the paper with a summary of the discussed topics and an outlook to further work.

II. API Description

This section describes the proposed API for the Application Coding part of the SBTVD TV 3.0 Call for Proposal [1]. The proposal extends the Ginga-HTML5 [9] environment by a JavaScript API that allows interaction with the built-in MPEG-H Audio decoder. This allows application developers programmatic control over the MPEG-H user interactivity features.

The basis for the proposed API is an extension of the *HTMLMediaElement* [21] with an *ngaInteractivity* attribute of type *NGAInteractivity*. This serves as an entry point for programmers to take advantage of next-generation audio features on an *HTMLMediaElement* that is playing back an MPEG-H audio stream. The structure of the *NGAInteractivity* interface and its sub-interfaces is described in the following overview:

- The *NGAInteractivity* interface contains a list of audio presets, with exactly one preset active at a time.
- Each *AudioPreset* object is representing an audio preset and contains a list of audio objects and a list of audio switch groups. This implies that different audio presets can have different audio objects and audio switch groups associated with them. It also allows each preset to maintain the state of its audio objects and audio switch groups.
- The state of each *AudioObject* instance consists of its prominence level in dB, its azimuth offset in degrees, its elevation offset in degrees, and whether or not it is muted.
- Each *AudioSwitchGroup* instance contains a set of audio objects with the same parameters as described above. Additionally, an audio switch group contains exactly one active audio object at a time.

The base interfaces used in the API are *AudioElement* and *AudioProperty*. Both of these are observable by extending the *EventTarget* API [19]. This means that the user can add and remove event listeners. The system will dispatch an event of type "change" when any of the associated parameters have changed. In addition, the *onchange* property on these interfaces allows the user to set an event handler for change events.

Fig. 1 summarizes the fundamental structure mentioned above. In the rest of this section, we describe the proposed interfaces in detail and explain how they enable users to use interactivity features.

A. *NGAInteractivity API*

The NGAInteractivity API, shown in Table I, allows the user to list all available AudioPresets, get and set the active

AudioPreset and get the default AudioPreset. Obtaining the default AudioPreset can be useful in case no active AudioPreset has been set by the user. The interface also enables the user to reset the audio scene to its default state, and set the language used for displaying the User Interface (UI) text and labels.

NGAInteractivity extends the EventTarget API [19]. A "change" event will be dispatched when the AudioPresetList or any of the parameters therein changed.

TABLE I
NGAINTERACTIVITY

Property/Method	Return type	Type
audioPresets	AudioPresetList	Read-only property
defaultPreset	AudioPreset	Read-only property
activePreset	AudioPreset	Read-only property
onchange	EventHandler	Property
setActivePresetById(int presetId)	void	Method
resetToDefault()	void	Method
setDisplayLanguage(string language)	void	Method

B. *AudioPresetList API*

The AudioPresetList API, shown in Table II, represents an iterable dynamic list of AudioPresets. Exactly one AudioPreset within a list of AudioPresets is enabled at a time.

AudioPresets within the list can be accessed either by iterating over the list indices, or by providing a *presetId* to the *getAudioPresetById* method. If the specified *presetId* matches the *id* of an AudioPreset within the list, that AudioPreset is returned, otherwise *getAudioPresetById* returns null.

TABLE II
AUDIOPRESETLIST

Property/Method	Return type	Type
length	int	Read-only property
getActivePresetById()	AudioPreset or null	Method

C. *AudioPreset API*

The AudioPreset API, shown in Table III, represents a singular AudioPreset. The Audio Preset encapsulates a list of AudioObjects and a list of AudioSwitchGroups, accessible to the user via the *audioObjects* and *audioSwitchGroups* properties respectively.

AudioPreset extends the AudioElement API and inherits its *id* and *label* properties. It also transitively inherits the *EventTarget* API, by which it will receive a "change" event when any property within an associated AudioObject or AudioSwitchGroup.

TABLE III
AUDIOPRESET

Property/Method	Return type	Type
id	int	Read-only property
label	string	Read-only property
audioObjects	AudioObjectList	Read-only property
audioSwitchGroups	AudioSwitchGroupList	Read-only property
onchange	EventHandler	Property

D. *AudioElement API*

The AudioElement API, shown in Table IV, serves as a base class for named observable UI elements. Each AudioElement is identified by a unique *id* and contains a *label* describing the AudioElement in the display language chosen by the user via the *setDisplayLanguage* method in the NGAInteractivity interface.

AudioElement extends the EventTarget API [19]. The system will dispatch a "change" event when any property associated with that AudioElement changed.

TABLE IV
AUDIOELEMENT

Property/Method	Return type	Type
id	int	Read-only property
label	string	Read-only property
onchange	EventHandler	Property

E. *AudioSwitchGroupList API*

The AudioSwitchGroupList API, shown in Table V, represents an iterable dynamic list of AudioSwitchGroups.

AudioSwitchGroups within the list are accessible either by iterating over the list indices, or via the *getAudioSwitchGroupById* method. If the specified *switchGroupId* matches the *id* of an AudioSwitchGroup within the list, that AudioSwitchGroup is returned, otherwise null is returned.

TABLE V
AUDIOSWITCHGROUPLIST

Property/Method	Return type	Type
length	int	Read-only property
getAudioSwitchGroupById(int switchGroupId)	AudioSwitchGroup or null	Method

F. *AudioSwitchGroup API*

The AudioSwitchGroup API, shown in Table VI, represents a switch group of AudioObjects. AudioSwitchGroup allows the user to list all associated AudioObjects through the *audioObjects* property. Within an AudioSwitchGroup, exactly one AudioObject is active at a time. The AudioSwitchGroup allows the user to get and set the active AudioObject and get the default AudioObject.

Moreover, AudioSwitchGroup objects can have a *mutingProperty*, which is an optional property of type AudioBooleanProperty. It allows the AudioSwitchGroup to be muted, regardless of which AudioObject is active and which is not. If muting for the AudioSwitchGroup is disallowed, the property will be null.

AudioSwitchGroup extends the AudioElement API and inherits its *id* and *label* properties. It also transitively inherits the EventTarget API, by which it will receive a "change" event when the active AudioObject or any property within an associated AudioObject changed.

TABLE VI
AUDIOSWITCHGROUP

Property/Method	Return type	Type
id	int	Read-only property
label	string	Read-only property
mutingProperty	AudioBooleanProperty or null	Read-only property
audioObjects	AudioObjectList	Read-only property
activeAudioObject	AudioObject	Read-only property
defaultAudioObject	AudioObject	Read-only property
onchange	EventHandler	Property
setActiveAudioObjectById(int objectId)	void	Method

G. *AudioObjectList API*

The AudioObjectList API, shown in Table VII, represents an iterable dynamic list of AudioObjects.

AudioObjects within the list are accessible either by iterating over the list indices, or via the *getAudioObjectById* method. If the specified *objectId* matches the *id* of an AudioObject within the list, that AudioObject is returned, otherwise *getAudioObjectById* returns null.

TABLE VII
AUDIOOBJECTLIST

Property/Method	Return type	Type
length	int	Read-only property
getAudioObjectById(int objectId)	AudioObject or null	Method

H. *AudioObject API*

The AudioObject API, shown in Table VIII, represents a singular AudioObject. The state of the AudioObject is defined by its prominence level value in dB, its azimuth offset value in degrees, its elevation offset value in degrees, and whether or not it is muted.

Each of these features is accessible through a property on the AudioObject interface. By changing the values of these properties via the AudioNumericProperty and AudioBooleanProperty interfaces, the user is able to control the state of the AudioObject.

All of these properties are optional. If user interaction with one of these properties is disallowed, it will return a null value. In case all properties are null, *isActionAllowed* returns false, indicating that no user interaction is possible on the AudioObject.

The *contentKind* property is a number representing the "mae_contentKind" as defined in [2].

AudioObject extends the AudioElement API and inherits its *id* and *label* properties. It also transitively inherits the EventTarget API, by which it will receive a "change" event when a property of the AudioObject was changed.

TABLE VIII
AUDIOOBJECT

Property/Method	Return type	Type
id	int	Read-only property
label	string	Read-only property
isActionAllowed	boolean	Read-only property
contentKind	int	Read-only property
mutingProperty	AudioBooleanProperty or null	Read-only property
prominenceLevelProperty	AudioNumericProperty or null	Read-only property
azimuthOffsetProperty	AudioNumericProperty or null	Read-only property
elevationOffsetProperty	AudioNumericProperty or null	Read-only property
onchange	EventHandler	Property

I. *AudioProperty API*

The AudioProperty API, shown in Table IX, serves as a base class for unnamed observable UI elements. Each AudioProperty has interfaces for getting and setting its value depending on its concrete subtype.

AudioElement extends the EventTarget API [19]. The system will dispatch a "change" event when the property's value was changed.

TABLE IX
AUDIOPROPERTY

Property/Method	Return type	Type
onchange	EventHandler	Property

J. *AudioBooleanProperty API*

The AudioBooleanProperty API, shown in Table X, represents a singular Boolean value. It allows the user to get and set the value through its methods. It also allows the user to get the default value of the property.

AudioBooleanProperty extends the AudioProperty API and transitively the EventTarget API, by which it will receive a "change" event when the property's Boolean value was changed.

TABLE X
AUDIOBOOLEANPROPERTY

Property/Method	Return type	Type
defaultValue	boolean	Read-only property
onchange	EventHandler	Property
getValue()	boolean	Method
setValue(boolean value)	void	Method

K. *AudioNumericProperty API*

The AudioNumericProperty API, shown in Table XI, represents a singular numeric value. It allows the user to get and set the value through its methods. It also allows the user to get the default value of the property.

The value of the property can be restricted to a specific range of values that the user can access through its *minValue* and *maxValue* properties. It is worth mentioning that the unit of the value depends on the property it represents. For instance, an AudioNumericProperty representing azimuth offset will have numeric values in degrees, while an AudioNumericProperty representing prominence gain will have numeric values in dB.

AudioBooleanProperty extends the AudioProperty API and transitively the EventTarget API, by which it will receive a "change" event when the property's numeric value was changed.

TABLE XI
AUDIONUMERICPROPERTY

Property/Method	Return type	Type
minValue	int	Read-only property
maxValue	int	Read-only property
defaultValue	int	Read-only property
onchange	EventHandler	Property
getValue()	int	Method
setValue(int value)	void	Method

III. PROTOTYPE IMPLEMENTATION

To prove the feasibility of the proposed API, we have implemented a prototype system that a) contains JavaScript bindings of the API itself as long as a native version in the browser is not available, b) shows the usage of the API to render functional user interfaces for audio interactivity in two use-case applications, and c) sets up a TV environment and a broadcaster environment to embed the use-cases in a context similar to production systems. We explain these three parts of the prototype system in this section.

A. *JavaScript Implementation*

The API proposal was written with a browser implementation and the possibility of a later standardization as a web standard in mind. As such, the specification resembles the standard documents published by the W3C in its form and content. In many cases, we resolved discussions in the development of the API in favor of consistency with other web standards, in order to give web developers, who are familiar with other browser APIs, the best possible development experience. This means that the translation of the specification into a JavaScript [14] interface follows the strict rules of the WebIDL [13] language used for the proposal. There is little room for interpretation regarding the interfaces.

To show the adequacy of the APIs for the purposes of enabling user interfaces capable of triggering audio interactivity in an MPEG-H Audio [2] playback, we implemented them in the *MpeghUiLib* JavaScript library. For the implementation, the TypeScript language[1] was chosen for three reasons. First, it is strictly typed, allowing us to declare the interface types and having them checked automatically. Through this, we enforced that our implementation's interfaces match the API proposal with the help of static checking tools. Secondly, TypeScript generates pure JavaScript code which can run on any modern and unmodified browser. Lastly, we also provide type declarations with the library to aid the user in developing applications for the proposed API. TypeScript's wide adoption in the web developer community lowers the barrier of entry to inspect and modify the prototype's code.

The interfaces specified in the proposal are fully implemented in the MpeghUiLib. The structure of NGAInteractivity objects and its sub-objects is fully available as described in the specification. That should allow carefully developed applications dependent on the MpeghUiLib to work with the browser implementation once it is available.

Contrary to the interfaces, the full implementation of the specified behavior is in a prototype state. The functionality that is needed for the use-case apps to work correctly is fully implemented as specified and was the main focus of the submission. In its current state, all parsing of MPEG-H audio scene data and the full translation to JavaScript objects is done correctly, as well as the dispatching of decoder events in the case of user interaction.

The main difference between the specification and the implementation in the MpeghUiLib is the entry point, which is supposed to be an extension of the HTMLMediaElement as defined in [21]. In the current phase of the proposal, we connect to an external decoder and cannot take advantage of HTMLMediaElements natively decoding MPEG-H in the browser. Hence, it makes no sense to connect the NGAInteractivity object to an HTMLMediaElement directly. Instead, our implementation offers the MpeghUiLib constructor to the user, so they can explicitly instantiate an MPEG-H user interface for their decoder. The additional *parseAudioScene* interface function and the user-defined *onUiAction* callback are available to feed audio scenes from the decoder to the library and user interactions from the library to the decoder respectively. These two additional functions will not be available in the native implementation of the API.

Furthermore, since multi-client interactivity was not needed for the use-case applications in the submission, the EventTarget dispatching is not fully implemented as specified: in the current state, only the respective element that a change event was dispatched on will trigger its *onchange* callbacks. According to the proposed API, also the parent elements of changed properties should be notified about a change, similar to bubbling events in the DOM specification [19]. This offers users to implement more granular handling of changes in their applications and is mostly useful if we assume that property changes can also be initiated in places other than the implemented user interface, which is not applicable to the provided use-case apps. The only case that needs to be considered in the current state is when the decoder itself triggers a scene change. This proves not to be a problem though, because the user can redraw the whole user interface in that case, which does not require bubbling events.

B. *Use-case Applications*

To show the usage of the API from an application developer's point of view, we have included two use-case applications in our submission. The difference between the two applications is primarily the content being played back to showcase the different application scenarios, specifically audio interactivity and the emergency warning system. Beyond that, there is no difference between the applications.

[1] https://www.typescriptlang.org/

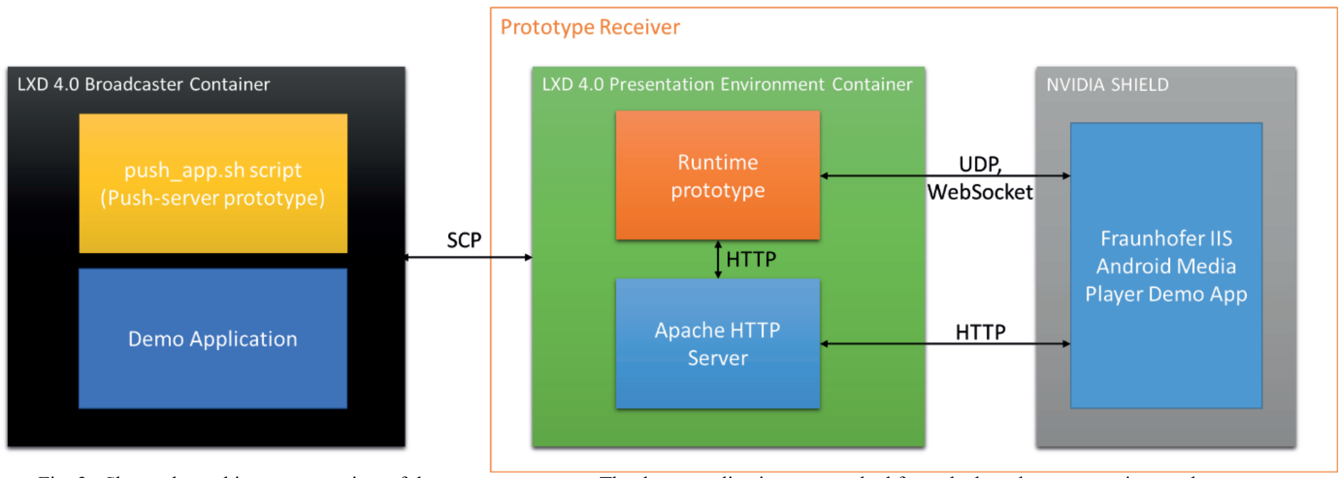

Fig. 2. Shows the architecture overview of the prototype system. The demo applications are pushed from the broadcaster container to the prototype receiver, where the runtime prototype containing the use-case apps is executed and controls the decoding on the NVIDIA Shield.

Hence, the rest of this section is applicable to both unless stated otherwise.

The apps are divided into two main parts. The first part is a renderer library that renders a view of the NGAInteractivity data on an HTML page. Its main task is to receive an NGAInteractivity object and render a DOM subtree that maps one-to-one onto the object's structure. As this is a library that we intend to reuse, we chose to implement it in TypeScript for the reasons mentioned above. In practice this is not strictly required; a plain JavaScript implementation or any other compatible language could also be used.

The second part is the entry point for the browser and consists of a simple HTML page with space for the UI on the bottom. The accompanying JavaScript file loads the MpeghUiLib and renderer libraries and initializes them. It also sets up the connection to the decoder which, as this is not a browser-internal implementation, takes some additional effort. In the prototype, a WebSocket [20] is used for communicating the interactivity information with the external decoder. This connection is also set up by the JavaScript file.

The final result is packed into one easily deployable file by Webpack[2]. An example of a user interface as rendered by the test application can be seen in Fig. 3. The item provides four different presets of which the "Padrão" preset is active. Inside the active preset, the user has access to an audio switch group "Língua" with selectable audio objects "Inglês" and "Francês" the former of which is active at the time.

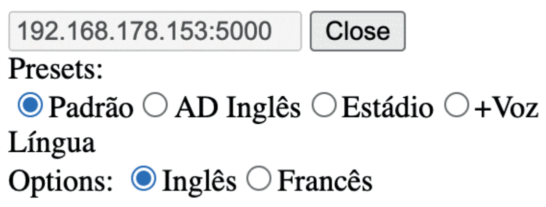
192.168.178.153:5000 Close
Presets:
[✓ Padrão] AD Inglês Estádio +Voz
Língua
Options: [✓ Inglês] Francês

Fig. 3. Illustrates the user interface of the use-case apps.

C. *Prototype System*

The use-case apps generally work fine on their own, but require some additional components to be set up to reflect a real broadcasting scenario. These are a) an external decoder to do the actual MPEG-H Audio decoding and interpretation of the interactivity commands, and b) respective presentation and broadcaster environments that transmit and launch the apps from a broadcasting facility to a consumer's simulated TV set. An architectural overview of the demonstration setup can be seen in Fig. 2.

For the external decoder, we use an Android-based NVIDIA Shield[3] running the Fraunhofer IIS Android Media Player App. The App is capable of decoding MPEG-H Audio from local MP4 [16] files, as well as network streams containing MPEG-H and using transport formats like MPEG DASH [15], Transport Stream [17] or plain fragmented MP4 files with progressive HTTP download. We chose MPEG DASH and progressive MP4 playback over the network as content delivery options for the use-case test applications. This is done to avoid pre-deploying content to the decoding device and instead have the content come from the presentation environment for a more realistic delivery scenario.

In addition to the content delivery via HTTP, the player app listens for JSON-formatted playback commands on a UDP port. These commands also contain the type and URL of the resource to be played back, so the use-case apps have control over the playback, even though they are running on another device.

Furthermore, once the playback is started, the player will listen to WebSocket connections on another TCP port. This second, bidirectional connection is used for the transmission of user interactivity data. The player uses an established connection to send audio scene information including available presets, audio objects, switch groups, labels, and interactive properties with their ranges to the presentation environment. The user interface uses the same connection to send user interactivity commands to the player. All information is sent in an XML [18] format and is a detail of the player implementation.

The decoding is done on an external device in order to demonstrate the capability to deploy the system as a distributed architecture across multiple devices. At the same time, the already available features in the Fraunhofer IIS Android Media Player Application could be used during the evaluation. In practice, the player and the user interface can be encapsulated in a single system; a player can run on the same device as the user interface, they can draw to the same screen or buffer, and the connection does not need to happen

[2] https://webpack.js.org/

[3] https://www.nvidia.com/en-us/shield/

TABLE XII
APPLICATION CODING REQUIREMENTS (EXCERPT FROM [12])

Use case		Minimum technical specification		Over the air delivery	Internet delivery
AP13	Enable emergency warning information delivery using an interactive application	AP13.1	Emergency warning information interactive application	desirable	desirable
AP14	Support for immersive TV	AP14.4	3D object-based immersive audio interaction	required	required
		AP14.5	3D media positioning and interaction	required	required

over a WebSocket, but can also be achieved through other protocols or natively with a suitable decoder API.

Finally, the presentation environment and the broadcaster environment were submitted as LXD[4] containers due to the easy handling and deployment in a test environment. The broadcaster container is mainly responsible to push the use-case apps and content from the broadcaster to the presentation environment container running on a consumer's simulated TV set. The presentation environment container is then responsible for delivering the content to the decoder (using the Apache 2 HTTP server[5] in the demonstration) and running the use-case apps to render the HTML user interfaces on a Chromium Browser[6].

IV. EVALUATION

The "SBTVD Forum – TV 3.0 – CfP Phase 2 / Testing and evaluation" document [12] defines 17 use-cases for the Application Coding component. Two of these use cases are addressing next-generation audio and all corresponding requirements are fulfilled by our Application Coding proposal for MPEG-H Audio. Table XII lists these fulfilled use-cases and requirements.

For evaluation purpose, we have defined and conducted five specific test procedures, which allow an easy evaluation of the proposed system as required in [12]. Each of these test procedures covers a unique feature of MPEG-H Audio [2] that can be mapped to a concrete requirement specified in [12] as shown in Table XIII. It is important to note, that the more advanced tests (4 and 5) are fulfilling multiple requirements. For example, test procedure 5 is fulfilling AP 13.1 and AP 14.4 from [12].

TABLE XIII
TEST PROCEDURES

Test procedure	MPEG-H Audio feature	SBTVD requirement
1	Preset interactivity	AP 14.4 - 3D object-based immersive audio interaction
2	Switch group interactivity	AP 14.4 - 3D object-based immersive audio interaction
3	Gain interactivity	AP 14.4 - 3D object-based immersive audio interaction
4	Position interactivity	AP 14.5 –3D media positioning and interaction
5	Emergency warning	AP 13.1 – emergency warning information interactive application

As an example, for one of these test procedures, Fig. 4 illustrates the MPEG-H Audio UI of test procedure 3. The gain of the "Língua" switch group can be decreased or increased by moving the corresponding "ProminenceLevel" slider to the left or right respectively. Since MPEG-H Audio [2] includes advanced Loudness Normalization functionality [7], the overall loudness of the audio output will stay the same, and only the balance between the "Língua" object and the rest of the audio scene will change.

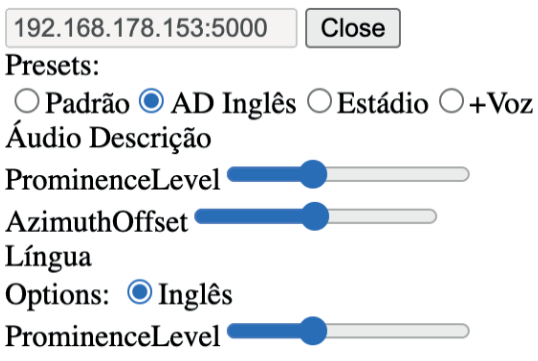
192.168.178.153:5000 Close
Presets:
Padrão [✓ AD Inglês] Estádio +Voz
Áudio Descrição
ProminenceLevel
AzimuthOffset
Língua
Options: [✓ Inglês]
ProminenceLevel

Fig. 4. Illustrates the prototype UI while decoding an audio item with 4 presets. The "ProminenceLevel" slider can be used to control the balance between the corresponding audio switch group or audio object in relation to the overall audio scene.

V. OUTLOOK

The previous sections describe the design, implementation and internal evaluation of our approach to enable next-generation 3D interactive audio for the Application Coding component of the SBTVD TV 3.0 Project using the MPEG-H Audio system. Our proposal is a proof-of-concept that demonstrates the required features according to the TV 3.0 CfP [1]. After the TV 3.0 Phase 2 evaluation process will be finalized, it is foreseen that all accepted proposals will be adapted and further refined for standardizing the best solutions for the TV 3.0 Application Coding layer. As an active member of the SBTVD Forum, Fraunhofer IIS will continue to support the standardization effort and is committed to work closely with the SBTVD Forum experts for integrating a 3D object-based audio API into the TV 3.0 suite of standards according to the Forum decisions.

It should be noted that the user interaction in the illustrated scenario benefits from bidirectional communication between the decoder and the user interface component. The decoder needs to be notified about user interactions and has to adapt the audio scene rendering accordingly. On the other hand, the rendered user interface depends on the metadata present in the audio stream and has to react to changes therein. For ensuring a high quality of experience, it is desired to have low latency between an interaction and the expected effect, which is achieved with a close integration of the two components.

[4] https://linuxcontainers.org/lxd/introduction/
[5] https://httpd.apache.org/
[6] https://www.chromium.org/Home

The event-based design of our proposal enables this close integration and could optimally be achieved by a native implementation in Ginga-HTML5. As a less strict option, bidirectional communication via WebSockets can be used to enable this form of close integration of the components, as shown in the submitted prototype.

It would also be possible to implement the desired behavior as a library on top of a REST API in Ginga-CC-WebServices similar to the existing SBTVD TV 2.5 specification [10]. However, as this is a fundamentally unidirectional mode of communication, active polling of the REST API or a different method of notifying the user interface about bitstream-triggered UI changes has to be implemented in order to ensure a good quality of the experience.

Similarly, the submitted setup for TV 3.0 is using an MPEG-H decoder running on an external device with stereo or binaural headphone output for the prototype. However, the described API design is not restricted to this setup and can also be used with other audio outputs like immersive AVRs, soundbars or integrated TV speakers.

VI. Conclusion

With its standardized metadata and decoder interfaces, the MPEG-H Audio system [2] offers a stable foundation for extending HTML5 web APIs for controlling advanced audio features in a device-independent and interoperable fashion. It was proposed as a candidate technology for the Application Coding component of the SBTVD TV 3.0 Project Call for Proposals [1]. This API extends the Ginga-HTML5 standard [9] and enables applications to offer user interaction with MPEG-H interactivity features, like switching presets and audio switch groups, as well as adjusting the prominence level and position of audio objects.

A prototype JavaScript implementation of the API was submitted to the SBTVD Forum for testing and evaluation. Accompanying runtime prototypes for broadcast and receiver environments show the feasibility of the proposal. The prototype environments employ a distributed architecture including an external decoder, illustrating the flexibility of the MPEG-H ecosystem.

Furthermore, following the evaluation test procedures described in this paper, the proposed API fulfills the TV 3.0 Application Coding requirements for "3D object-based immersive audio interaction", "3D media positioning and interaction" and support for "emergency warning information interactive applications".

Acknowledgment

The authors would like to thank the SBTVD Forum for the efforts to organize the TV 3.0 Project and support us during the development of the proposal, especially Professor Marcelo F. Moreno, who is chairing the Application Coding working group in the SBTVD Technical Module and has led the technical foundation of the proposal together with Rafael Diniz in the paper on immersive audio properties for NCL media elements [11].

References

[1] Brazilian Digital Terrestrial Television System Forum. (2020, July 17). "Call for Proposals: TV 3.0 Project". [Online]. Available: https://forumsbtvd.org.br/wp-content/uploads/2020/07/SBTVDTV-3-0-CfP.pdf

[2] "Information technology - High efficiency coding and media delivery in heterogeneous environments - Part 3: 3D audio," International Organization for Standardization (ISO), Geneva, Standard ISO/IEC 23008-3:2019, 2019.

[3] Fraunhofer Audio Blog. (2019, Aug. 27). "MPEG-H Audio selected to enhance Brazilian digital television with immersive and personalized sound", [Online]. Available: https://www.audioblog.iis.fraunhofer.com/mpegh-brazil-isdbtb

[4] "Televisão digital terrestre - Codificação de vídeo, áudio e multiplexação - Parte 2: Codificação de áudio," ABNT NBR 15602-2:2020. [Online]. Available: https://forumsbtvd.org.br/legislacao-e-normas-tecnicas/normas-tecnicas-da-tv-digital/english/

[5] "Televisão digital terrestre - Multiplexação e serviços de informação (SI)," ABNT NBR 15603:2020. [Online]. Available: https://forumsbtvd.org.br/legislacao-e-normas-tecnicas/normas-tecnicas-da-tv-digital/english/

[6] "Televisão digital terrestre – Receptores," ABNT NBR 15604:2020. [Online]. Available: https://forumsbtvd.org.br/legislacao-e-normas-tecnicas/normas-tecnicas-da-tv-digital/english/

[7] R. L. Bleidt et al., "Development of the MPEG-H TV Audio System for ATSC 3.0," in *IEEE Transactions on Broadcasting*, vol. 63, no. 1, pp. 202-236, March 2017, doi: 10.1109/TBC.2017.2661258.

[8] A. Murtaza and S. Meltzer, "First Experiences with the MPEG-H TV Audio System in Broadcast," *SET INTERNATIONAL JOURNAL OF BROADCAST*, ISSN Print: 2446-9246 ISSN [Online]. 2446-9432. doi: 10.18580/setijbe.2018.6. Available: https://www.set.org.br/ijbe/ed4/Artigo%206.pdf

[9] "Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 10: Ginga-HTML5 - Especificação do perfil HTML5 no Ginga", ABNT NBR 15606-10:2021. [Online]. Available: https://forumsbtvd.org.br/legislacao-e-normas-tecnicas/normas-tecnicas-da-tv-digital/english/

[10] "Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 11: Ginga CC WebServices - Especificação de WebServices do Ginga Common Core", ABNT NBR 15606-11:2021. [Online]. Available: https://forumsbtvd.org.br/legislacao-e-normas-tecnicas/normas-tecnicas-da-tv-digital/english/

[11] R. Diniz and M. Moreno. "Immersive audio properties for NCL media elements", in *Anais Estendidos do XXV Simpósio Brasileiro de Sistemas Multimídia e Web, Florianópolis*, 2019, pp. 195-197, doi: https://doi.org/10.5753/webmedia_estendido.2019.8164. W.-K. Chen, *Linear Networks and Systems.* Belmont, CA: Wadsworth, 1993, pp. 123–135.

[12] Brazilian Digital Terrestrial Television System Forum. (2021, Mar. 15). "CfP Phase 2/Testing and Evaluation: TV 3.0 Project". [Online]. Available: https://forumsbtvd.org.br/wp-content/uploads/2021/03/SBTVD-TV_3_0-P2_TE_2021-03-15.pdf

[13] C. McCormack. (2016, Dec. 15). WebIDL Level 1. *W3C Recommendation* [Online]. Available: https://www.w3.org/TR/WebIDL-1/

[14] ECMA International. "ECMAScript® 2021 language specification," ECMA-262, 12th edition, June 2021. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[15] "Information Technology - Dynamic Adaptive Streaming Over HTTP (DASH) -- Part 1: Media Presentation Description and Segment Formats," International Organization for Standardization (ISO), Geneva, Standard ISO/IEC 23009-1:2019, 3th edition, 2019.

[16] "Information Technology - Coding of Audio-Visual Objects -- Part 12: ISO Base Media File Format," International Organization for Standardization (ISO), Geneva, Standard ISO/IEC 14496-12:2020, 6th edition, 2020.

[17] "Information Technology - Generic coding of moving pictures and associated audio information -- Part 1: Systems," International Organization for Standardization (ISO), Geneva, Standard ISO/IEC 13818-1:2018, 6th edition, 2018.

[18] T. Bray, J. Paoli, M. Sperberg-McQueen. (2008, Nov.). Extensible Markup Language (XML) 1.0. *W3C Recommendation.* [Online]. Available: https://www.w3.org/TR/xml/

[19] Y. Zhu et al. (2021, Jun.). DOM Living Standard. *W3C Recommendation.* [Online]. Available: https://www.w3.org/TR/dom/

[20] I. Fette, A. Melnikov. "The WebSocket Protocol" *IETF, Request for Comments (RFC 6455)*, Dec. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6455.txt

[21] S. Faulkner, A. Eicholz, T. Leithead, A. Danilo, S. Moon. "HTML 5.2." *W3C Recommendation, Jan. 2021.* [Online]. Available: https://www.w3.org/TR/html52/

Oliver Major received both his B.Sc. and his M.Sc. degrees in computer science at the RWTH Aachen University in Aachen, Germany in 2014 and 2016 respectively.

Between 2014 and 2015, he gained some experience with network communication and integrated platforms as a Student Assistant at devolo AG in Aachen, Germany. After his studies, he joined the Mobile Audio Rendering group at the Fraunhofer Institute for Integrated Circuits (IIS) in Erlangen, Germany as a Research Engineer in 2017, where his main topics were binaural rendering, platforms and web technologies. After joining the Web Media Technologies group in 2020, his focus shifted more towards system architecture and web technologies.

Ziad Shaban received his B.Sc. degree in electrical engineering - communications and electronics from the Jordan University of Science and Technology, Irbid, Jordan in 2013 and his M.Sc. degree in communication and multimedia engineering from the Friedrich-Alexander University Erlangen-Nürnberg, Erlangen, Germany, in 2015.

Between 2013 and 2016, he was active as a Research Assistant in the fields of software-defined radio and wireless communication at the Fraunhofer Institute for Integrated Circuits (IIS) in Erlangen, Germany, where he also wrote his thesis titled “A Study of a Testbed for Multi-channel Simulators”. In 2016, he joined the Multimedia Transport group of Fraunhofer IIS as a Research Engineer with a focus on the research and development of streaming technologies. As of 2020, he has been a member of the Web Media Technologies group.

Bernd Czelhan received the B. Sc. in 2011 and M. Sc. in 2012 degrees in Computer Science from the Technische Hochschule Nürnberg Georg Simon Ohm, in Nuremberg, Germany.

In 2012, he joined the Fraunhofer Institute for Integrated Circuits (IIS) as a research engineer, where his main working topic is the next-generation audio codec MPEG-H and Web development. Since 2020, he is heading the Web Media Technologies group. In addition, he is supporting the practical implementation of MPEG-H. He is especially interested in Web technologies and modern transport mechanism and system aspects of today’s audio codecs, such as MMT, DASH/Route, and hybrid delivery.

Adrian Murtaza received his M.Sc. degree in Communication Systems from the École Polytechnique Fédérale de Lausanne, Switzerland in 2012 with a thesis on “Backward Compatible Smart and Interactive Audio Transmission”. Upon graduation he joined Fraunhofer IIS, where he works as a Senior Manager, Technology and Standards.

Adrian joined MPEG in 2013 and since then contributed to development of various audio technical standards in MPEG-D and MPEG-H. He serves as Fraunhofer's Standards Manager in a number of industry standards bodies, including SBTVD, ATSC, CTA, DVB, HbbTV and SCTE, and is the co-author of multiple specifications in those groups.

More recently he focused on specification of Next-Generation Audio delivery and transport in ATSC 3.0 systems and MPEG-2 Transport Stream based systems, as well as on enabling of MPEG-H Audio services in different broadcast and streaming ecosystems. With a strong interest in VR/AR media solutions he is actively involved in MPEG-I efforts targeting future immersive applications.

Received in 2021-08-16 | Approved in 2021-12-07