|  |  |
| --- | --- |
|  | Moving Picture, Audio and Data Coding by Artificial Intelligence  www.mpai.community |

|  |
| --- |
| **Public Document** |

|  |  |
| --- | --- |
| N105 | 2020/12/16 |
| Source | Andrea Basso, Danilo Pau, Davide Ghezzi |
| Title | Validation efforts for MPAI-AIF model |
| Target | Communication AC |

## Summary

The document reports the results of the first efforts in validating the MPAI-AIF model by means of an initial implementation of a MPAI-AIF applied to an MPAI-CAE Use Case. The results demonstartes the validity of the MPAI-AIF model that MPAI has adopted for its MPAI-AIF Call for Technologies.

The implementation is based on ST Microelectronics low-cost microcontrollers and an initial set of APIs. The implemented MPAI-AIF model comprises two AIMs. The first one is AI-based and performs Acoustic Scene Classification (ASC) the second one is a traditional processing module that is able to perform beamforming and source localization on signals coming from a microphone array of 4 microphones. On the basis of the classification of the acoustic environment performed by the fist AIM the second module can adapt its processing switching from direct signal passthrough to beamforming based on source localization.

## What is MPAI-AIF

MPAI-AIF is a framework that

* Provides means to encapsulate existing or novel technologies with standard interfaces that are defined on a per use case basis, so that they can be made interoperable and easily connectable. AIF enables the development of products and services with a plug and play approach to achieve low cost and very fast time to market.
* Includes a series of (essential) systems and methods that allow the Framework to

Support a mix of AI and traditional processing components and their computational paradigms (e.g., the traditional ML lifecycle)

* Is scalable
* Support a variety of service levels. A Service level measures the performance of a system. Certain goals are defined, and the service level gives the percentage to which those goals should be achieved. Fill rate is different from service level. Examples of service level: Percentage of calls answered in a call center. Or time a given system is running without interruptions (high reliability and availability of a system).
* It is the ground for at least the following more specific technologies:
  + Context-based Audio Enhancement (MPAI-CAE)
  + Integrative Genomic/Sensor Analysis (MPAI-GSA)
  + AI-Enhanced Video Coding (MPAI-EVC)
  + Server-based Predictive Multiplayer Gaming (MPAI-SPG)
  + Multi-Modal Conversation (MPAI-MMC)
  + Compression and Understanding of Industrial data (MPAI-CUI)

## Demo Description

The CAE use case considered is **“**enhanced audio conference”. The implemented AIF comprises two AIMs. The first one is AI-based and performs Acoustic Scene Classification (ASC) the second one is a traditional signal processing module that performs either singnal passthrough or beamforming and source localization on signals coming from a microphone array (4 microphones). On the basis of the classification of the acoustic environment performed by the fist AIM the second AIM adapts it processing, switching from direct signal passthrough to beamforming based on source localization. The demo relies of STM32 microcontrollers and ST AI software components freely available. The block diagram is shown in figure 4.

With reference to figure 1, 2, 3, the first sub-system is composed by a ST Bluecoin STEVAL-BCNKT01V1 which is equipped with the microphone array (4 microphones) and a STM32 micro controller and implements the AIM that can adaptively process the input microphone signal.

The second sub-system composed by a ST Sensortile STEVAL-STLKT01V1 implements a second AIM that is AI-based and performs Automatic Scene Classification (ASC[[1]](#footnote-1)) and the AIF management and control.

Via the AIF control and management interface the Bluecoin module switches it processing configuration from omnidirectional audio capture to strong beamforming based on source localization when the ASC class recognized is “outdoor”. The implementation is based on low-cost low power mass market micro controller as proof of its viability.

A picture containing indoor, table

Description automatically generated

ST Sensortile STEVAL-STLKT01V1

ST Bluecoin

STEVAL-BCNKT01V1

Audio Output

![A picture containing electronics

Description automatically generated]()

A picture containing text, electronics, circuit

Description automatically generated

Figure 1 Demonstrator based on ST boards

Figure 3 SensortileSTEVAL-STLKT01V1

Figure 2 Bluecoin STEVAL-BCNKT01V1



Figure 4 Demo Architecture

## AIF API model

The MPAI-AIF API model involves handling processing of data with the following characteristics

1. has been designed to allow modular routing
2. Basic processing operations are performed with MPAI AIMs, which are linked together to form a routing graph (DAG).
3. Several sources with different types of channel layout

AI Modules (AIMs) are linked into Direct Acyclic Graphs (DAG) by their inputs and outputs. They typically start with one or more sources. Sources provide arrays of data samples at very different time slices. Outputs of these nodes could be linked to inputs of others, which mix or modify these streams of data into different streams. Once the data has been processed for the intended effect, it can be linked to the input of a destination, which sends the processed data to the destination.

### Timing

Timing is controlled with high precision and low latency, allowing developers to write code that responds accurately to events and is able to target specific samples, even at a high sample rate.

## Design principles

The design of the API has been based on the notion of event-based workflows and worklets.

In computer science a worklet is an object representing a set of tasks created to be reused in the context of multiple workflows. The workflow that contains the worklet can called the parent workflow. When a worklet is run, it expands to run the tasks it contains and evaluate its links and connections.

A single instance of the AIF framework is a worklet that contains

* Definition of a processing graph
* Configuration methods for the individual AIMs
* Allocated storage for framework usage
* Configuration and handling of communication interfaces
* Configuration and management of control interfaces
* Event handling of the AIF

Thus, a typical implementation of MPAI AIF would look something like:

* *Create AIF instance and inside the AIF Instance* 
  + *Create sources*
  + *Create AIMs*
  + *Choose final destination of output*
  + *Connect the sources to AIMs*
  + *Connect AIMs to the destination*
  + *Start the AIF instance*

![Graphical user interface, text

Description automatically generated]()

Figure 5 – AIF instance creation

An initial partial description of the API is presented in Annex A

**Demonstration**

A video of the demo is presented in figure 6. The 2nd sub-system assumes to be an indoor environment (yellow led blinking on the Sensortile and 4 red leds on the Bluecoin) once the AI Audio Scene Classification module identifies an outdoor scene, the yellow led on the Sensortile changes to yellow solid on and the Bluecoin that implements the audio processing AIM changes configuration from omnidirectional capture to strong directional beamforming (single red led on in the direction of the sound identified by source localization).



Figure Demo

A recording of the audio output while the operator is counting 1 to 8 is presented below. After the first count the acoustic scene is classified as outdoor and the beamforming with source localization is activated.



## Next steps

A second phase that will include also an AI based denoising component will be implemented as shown in figure 7



Figure

*DISCLAIMER: This is a first attempt to validate some aspects of the AIF and AIM structure*

*and does not claim to be a complete implementation nor a reference software and/or hardware. The goal of this feasibility study is to support the design and the standardization process of AIF and related use cases.*

## ANNEX A

## API interfaces

The AIF API has a number of interfaces and associated events.

### AIFBaseContext

It acts as a base definition of the AIF that can be extended (overloaded). AIFBaseContext it is not to be used directly — but its features are used via AIFContext inheriting interface.

Properties

* **AIFBaseContext.Worklet** Read only Secure

Returns the Worklet object, which can be used to create and manage AIMs. It is also the mean to access the AIMs belonging to a given AIF.

* **AIFBaseContext.Connect** Secure context connection with other AIF instances

Returns the Connect object, which can be used to create and manage connections to other AIFs.

* **AIFBaseContext.Control** Secure context control with application instances

Returns the Control object, which can be used to create and manage control paths.

* **AIFBaseContext.currentTime** Read only

Returns a double representing an ever-increasing hardware time in seconds used for scheduling. It starts at 0.

* **AIFBaseContext.destinations** Read only

Returns an AIFDestinations representing the final destinations of all data in the context. Note that the input of the AIF is defined by means of the AIMs.

* **AIFBaseContext.state** Read only

Returns the current state of the AIFContext.

* **AIFBaseContext.ExtendedFunctions** Read only Secure context

Returns the list of extended functions which can be used to create and manage AIMs in which code implementing the AIMWorkletProcessor interface are run to process data.

* **AIFBaseContext.onstatechange**

An event handler that runs when an event of type statechange has fired. This occurs when the AIFContext's state changes, due to the calling of one of the state change methods (AIFContext.suspend, AIFContext.resume, or AIFContext.close).

#### Methods

* **AIFBaseContext.createStorage()**

Creates a new, empty AIFBuffer object, which can then be populated by data and played via an AIFBufferSourceNode.

* **AIFBaseContext.createControl()**

Creates an AIFBufferSourceNode, which can be used to play and manipulate audio data contained within an AIFBuffer object. AIFBuffers are created using AIFContext.createBuffer or returned by AIFContext.decodeAIFData when it successfully decodes an audio track.

* **EventHandling()**

Manages the events occurring inside the AIF including the error management.

### AIFContext

The AIFContext interface represents a processing graph built from modules linked together, each represented by an AIM. An AIFContext controls the creation of the nodes it contains and the execution of the processing. You need to create an AIFContext before you do anything else, as everything happens inside a context.

#### Properties

*Also inherits methods from its parent interface* ***AIFBaseContext****.*

* **AIFContext.baseLatency**Read only

Returns the number of seconds of processing latency incurred by the AIFContext passing the audio from the AIFDestinationNode to the subsystem.

* **AIFContext.outputLatency**Read only

Returns an estimation of the output latency of the current context.

#### Methods

*Also inherits methods from its parent interface* ***AIFBaseContext****.*

* **AIFContext.init()**

initialize the AIFcontext to a certain state.

* **AIFContext.create()**

Create the AIFcontext

* **AIFContext.close()**

Closes the AIFcontext, releasing any system resources that it uses.

* **AIFContext.getOutputTimestamp()**

Returns a new AudioTimestamp object containing two audio timestamp values relating to the current audio context.

* **AIFContext.resume()**

Resumes the progression of time in an audio context that has previously been suspended/paused.

* **AIFContext.suspend()**

Suspends the progression of time in the audio context, temporarily halting audio hardware access and reducing CPU/battery usage in the process.

### AIFContextOptions

The AIFContextOptions dictionary is used to provide options when instantiating a new AIFContext.

#### Properties

* **supportUpdate** Optional

Support seamless update of the AIMs

## AIMs

Each AIM has in general inputs and outputs. Each input and output have a given amount of channels. For example, mono audio has one channel, while stereo audio has two channels.

Multiple AIMs are connected to build a processing graph. This graph is contained in an AIFContext. A source node has zero inputs but one or multiple outputs, and can be used to generate a signal or data. A destination node has no outputs; instead, all its inputs are directly outputs output for example to the AIFcontext.destination).

The exact processing done varies from one AIM to another and potentially from use case to use case but, in general, a node reads its inputs, does some processing, and generates new values for its outputs.

AIMs have some common properties and methods that can be overloaded in the specification of AIMs for individual use cases.

### Properties

* **AIM.context** Read only

Returns the associated BaseAIMContext, that is the object representing the processing graph the node is participating in.

* **AIM.numberOfInputs** Read only

Returns the number of inputs feeding the node. Source nodes are defined as nodes having a numberOfInputs property with a value of 0.

* **AIM.numberOfOutputs** Read only

Returns the number of outputs coming out of the node. Destination nodes — like AudioDestinationNode — have a value of 0 for this attribute.

* **AIM.channelCount**

Represents an integer used to determine how many channels are used when combining connections to any inputs to the node. Its usage and precise definition depend on the value of AIM.channelCountMode.

* **AIM.channelCountMode**

Represents an enumerated value describing the way channels must be matched between the node's inputs and outputs.

* **AIM.channelInterpretation**

Represents an enumerated value describing the meaning of the channels. This interpretation will define for example in an audio scenario up-mixing and down-mixing will happen.

### Methods

* **AIM.connect()**

Allows us to connect the output of this node to be input into another node, either as audio data or as the value of an AudioParam.

* **AIM.disconnect()**

Allows us to disconnect the current node from another one it is already connected to.

### AIMParam

The AIMParam interface represents a processing-related parameter. It can be set to a specific value or a change in value, and can be scheduled to happen at a specific time and following a specific pattern.

**Example**

This simple snippet of code shows the creation of an AIF and AIM configuration

const AIFCtx = new AIFContext();

const ASCNode = new AIM(AIFCtx);

ASCNode.Configure( ASCnetwork,

ASCnetwork.params,

AudioInput)

ASCNode.connect(InputNode).connect(audioCtx.destination);

## Sources

Interfaces that define sources for use in the AIF API. These interfaces are use case dependent. TBC

## Destinations

Interfaces that define sources for use in the AIF API. These interfaces are use case dependent. TBC

1. <https://www.st.com/en/embedded-software/fp-ai-sensing1.html> [↑](#footnote-ref-1)