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

MPAI Technical Specification

AI Framework
MPAI-AIF

WD 0.12

WARNING

Use of the technologies described in this Technical Specification may infringe patents, copyrights or intellectual property rights of MPAI Members or non-members.

MPAI and its Members accept no responsibility whatsoever for damages or liability, direct or consequential, which may result from use of this Technical Specification.

Readers are invited to review Annex 2 – Notices and Disclaimers.

AI Framework
Version 1

1 Introduction (Informative)

Moving Picture, Audio and Data Coding by Artificial Intelligence (MPAI) is an international Standards Developing Organisation with the mission to develop *AI-enabled data coding standards*. Research has shown that data coding with AI-based technologies is generally *more efficient* than with existing technologies. Compression and feature-based description are notable examples of coding. MPAI Application Standards enable the development of AI-based products, applications and services.

In the following, Terms beginning with a capital letter are defined in *Table 1* if they are specific to this Standard and in *Table 4* if they are common to all MPAI Standards.

Figure 1 depicts the Reference Model of this AI Framework (AIF) Standard (MPAI-AIF) that provides the foundation on which Implementations of MPAI Application Standards operate.

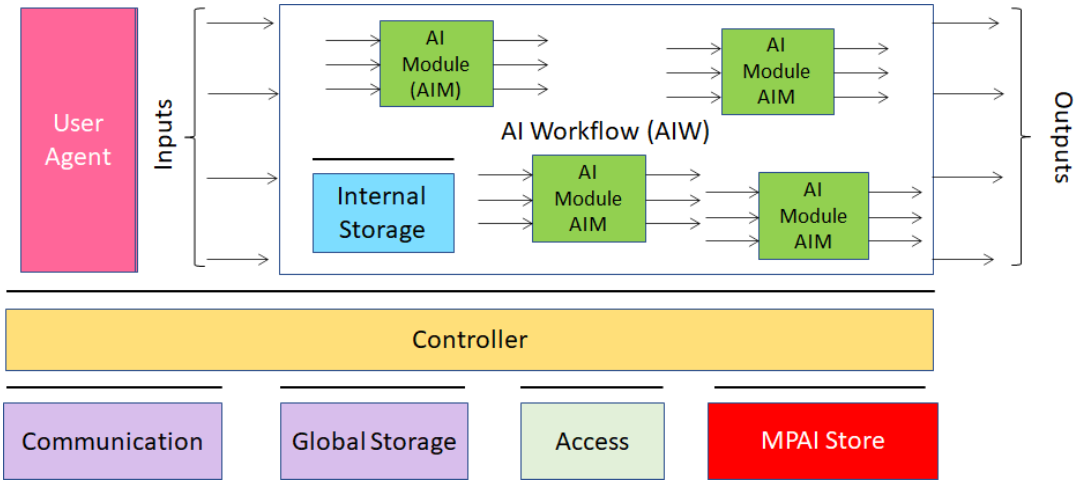

Figure 1 – The AI Framework (AIF) Reference Model and its Components

An AIF Implementation allows execution of AI Workflows (AIW), composed by basic processing elements called AI Modules (AIM).
MPAI Application Standards normatively specify Semantics and Format of the input and output data and the Function of the AIW and the AIMs, and the Connections between and among the AIMs of an AIW.
In particular, an AIM is defined by its Function and Data, but not by its internal architecture, which may be based on AI or data processing, and implemented in software, hardware or hybrid software and hardware technologies.
MPAI defines Interoperability as the ability to replace an AIW or an AIM Implementation with a functionally equivalent Implementation. MPAI also defines 3 Interoperability Levels of an AIW that executes an AIW. The AIW may be:

1. Proprietary and composed of AIMs with proprietary functions using any proprietary data Format (*Level 1*).
2. Composed of AIMs having all their Functions, Formats and Connections specified by an MPAI Application Standard (*Level 2*).
3. Composed of AIMs that have the characteristics of point 2. above and certified by an MPAI-appointed Assessor to possess the attributes of Reliability, Robustness, Replicability and Fairness – collectively called Performance (*Level 3*).

MPAI is the root of trust of the MPAI Ecosystem [1] offering Users access to the promised benefits of AI with a guarantee of increased transparency, trust and reliability as the Interoperability Level of an Implementation moves from 1 to 3. Additional information is provided by Annex 3.

2 Scope (Normative)

The MPAI *AI Framework* (MPAI-AIF) Standard specifies architecture, interfaces, protocols and APIs of an AI Framework (AIF) capable of executing AI-based products, services and applications.
MPAI-AIF has the following main features:

- Is component-based.
- Defines the interfaces amongst it Components.
- Is secure as the components operate in a trusted zone.
- Supports mixed hardware-software implementations.
- Supports distributed and local execution environments.
- Supports Machine Learning.
- Supports operation of AIFs in proximity.

The current version of MPAI-AIF has been developed by the MPAI AI Framework Development Committee (AIF-DC). Future Versions may revise and/or extend the Scope of the Standard.

3 Terms and definitions (Normative)

The Terms used in this standard whose first letter is capital are defined in *Table 1.* The Terms of MPAI-wide applicability are defined in *Table 4.*

Table 1 – MPAI-AIF Terms

Term	Definition
Access	Static or slowly changing data that are required by an application such as domain knowledge data, data models, etc.
AIF Metadata	The data set describing the capabilities of an AIF set by the AIF Implementer.

AIM Metadata	The data set describing the capabilities of an AIM set by the AIM Implementer.
AI Module (AIM)	A data processing element receiving AIM-specific inputs and producing AIM-specific outputs according to its Function. An AIM may be an aggregation of AIMs.
AI Workflow (AIW)	A structured aggregation of AIMs implementing a Use Case receiving AIM-specific inputs and producing AIM-specific outputs according to its Function.
Channel	A physical or logical connection between an output Port of an AIM and an input Port of an AIM. The term "connection" is also used as synonymous.
Communication	The infrastructure that implements message passing between AIMs.
Component	One of the (AIF elements: Access, AI Module, AI Workflow, Communication, Controller, Internal Storage, Global Storage, MPAI Store, and User Agent.
Controller	A Component that manages and controls the AIMs in the AIWs, so that they execute in the correct order and at the time when they are needed.
Data Type	An instance of the Data Types defined by 6.1.1.
Device	A hardware and/or software entity running at least one instance of an AIF.
Event	An occurrence acted on by an Implementation.
Global Storage	A Component to store data shared among AIMs.
Internal Storage	A Component to store data of individual AIMs.
Knowledge Base	Structured and/or unstructured information made accessible to AIMs via MPAI-specified interfaces.
Message	A sequence of Records.
MPAI Store	The repository of Implementations.
Port	A physical or logical communication interface of an AIM.
Record	A data structure with a specified structure.
Resource policy	The set of permissions under what conditions this applies specific actions may be applied.
Status	The set of parameter characterising a Component.
Time Base	The protocol specifying how Components can access timing information
Topology	The set of Channels connecting AIMs in an AIW.
User Agent	The Component interfacing the user with an AIF through the Controller
AIW Metadata	The data set describing the capabilities of an AIW set by the AIW Implementer.

4 References

4.1 Normative references

MPAI-AIF normatively references the following documents:

1. Technical Specification: The Governance of the MPAI Ecosystem V1.
2. GIT protocol, https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols.
3. ZIP format, https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT.
4. Date and Time in the Internet: Timestamps; IETF RFC 3339; July 2002.
5. Uniform Resource Identifiers (URI): Generic Syntax, IETF RFC 2396, August 1998.
6. The JavaScript Object Notation (JSON) Data Interchange Format; https://datatracker.ietf.org/doc/html/rfc8259; IETF rfc8259; December 2017

4.2 Informative references

7. Message Passing Interface (MPI), https://www.mcs.anl.gov/research/projects/mpi/

5 Architecture of the AI Framework (Normative)

5.1 AI Framework Components

MPAI-AIF normatively specifies the Components of *Figure 1* whose functions are:

1. Access: provides access to static or slowly changing data that are required by an application such as domain knowledge data, data models, etc.
2. AI Module (AIM): a data processing element receiving AIM-specific Inputs and producing AIM-specific Outputs according to according to its Function. An AIM may be an aggregation of AIMs.
3. AI Workflow (AIW): an organised aggregation of AIMs implementing a Use Case receiving AIM-specific Inputs and producing AIM-specific Outputs according to its Function.
4. Communication: connects the Components of an AIF.
5. Controller:
    a. Provides basic functionalities such as scheduling, inter AIMs communication, access to all the AIF components such as Internal and Global Storage.
    b. Activates/suspends/resumes/deactivates AIWs or AIMs according to the user's or other inputs
    c. Supports complex application scenarios by balancing load and resources.
    d. Exposes three APIs:
        i. AIM API through which modules can communicate with it (register themselves, communicate and access the rest of the AIF environment)
        ii. User API through which the user or other Controllers can perform high-level tasks (e.g., switch the Controller on and off, give inputs to the AIW through the Controller).
        iii. MPAI Store API to enable communication between the AIF and the Store.
    e. May run one or more AIWs.
6. Global Storage: stores data shared by AIMs.
7. Internal Storage: stores data of the individual AIMs.
8. MPAI Store: stores Implementations for users to download.
9. User Agent: The Component interfacing the user with an AIF through the Controller

5.2 AI Framework Features

General features of the AI Framework are:

1. Messages are of two types: High-Priority Messages are expressed as up to 16 bit integers. Normal-Priority Messages are expressed as Types defined by this Standard. Messages may be communicated through Channels or Events.
2. Communication needs not be persistent and AIMs may be hot-plugged or dynamically disconnected.
3. A Channel is unicast.
4. Controller may run on a different computing platform than the AIW.
5. The AIMs of a AIW may run on different computing platforms, e.g., in the cloud or on swarms of drones.
6. Appropriate API Profiles allow Implementation on different computing platforms and different programming languages.
7. The Controller will always be present even if the AIF is a lightweight Implementation. The Controller acts as a resource manager, according to instructions given by the User.

8. Modules may be hot-pluggable and register themselves on the fly. However, modules are executed:
    a. Locally, i.e., they encapsulate hardware physically accessible to the Controller.
    b. Elsewhere and encapsulate communication with a remote Controller.
9. When different Controllers, each one running on a different agent ("swarm element"), interact with one another, controllers cooperate in one of the two following ways:
    a. A Controller runs in local device (e.g., a traffic light) communicating with the swarm elements. The Controllers of the swarm elements register as communication AIMs with the local device whose Controller can feed them with input, thus controlling them. Control is implemented as internal logic to the AIW run by each swarm elements in response to information or signals sent by the local device seen by the Controllers of the swarm elements as an encapsulated AIM.
    b. There is no Controller and swarm elements in range register with all other swarm elements as a communication AIM. Each Controller will have to react to signals and information coming from as many AIMs as the swarm elements.

5.3 AI Framework Implementations

MPAI-AIF enables a wide variety of Implementations:

1. AIF Implementations may be tailored to different execution environments, e.g., High-Performance Computing systems or resource-constrained computing boards. For instance, the Controller might be a process on a HPC system or a library function on a computing board. However, the API through which the AIMs and AIWs are implemented are common.
2. The API may have different MPAI-defined profiles and, depending on the hardware and resources available, only some of them might be offered by a specific AIF implementation. Interoperability between AIMs, irrespective of whether they are implemented in hardware or in software, is ensured by the way communication between AIM Ports is defined.
3. AIMs may be Implemented in hardware, software and mixed-hardware and software.
4. MPAI-AIF specifies implementation-independent (i.e., hardware, software and hybrid) ways to communicate Messages between AIMs:
    a. Generation and management of Events.
    b. Use of Ports and Channels ensuring that compatible AIM Ports may be connected together irrespective of the AIMs' implementation technology.

5.4 AIMs

5.4.1 Implementation types

This section defines how HW and SW AIMs can communicate. AIMs can be implemented in either in HW or SW keeing the same interfaces.

Although AIMs communicate through the same API irrespective of whether they are implemented in HW or SW, there might be constraints imposed on the specific values of certain API parameters depending on the nature of the AIM. Different profiles may have different constraints. One example is that of Events (easy to accommodate in SW but less so in HW); and another is persistent connections (easy to make in HW, less so in SW).

While SW-SW and HW-HW connections are homogeneous, a HW-SW mixed scenario is inherently heterogeneous and requires the specification of additional communication protocols, which are used to wrap the HW part and connect it to SW.

Examples of supported architectures are:

- CPU-based devices running an operating system. They are presented as software blocks.
- Memory-mapped devices (FPGAs, GPUs, TPUs) which are presented as accelerators. They are presented as software blocks.

- Cloud-based frameworks. They are presented as software blocks.
- Naked hardware devices (i.e. IP in FPGAs) that communicate through hardware Ports. They are presented as (non-encapsulated) hardware blocks.
- Encapsulated blocks of a hardware design (i.e. IP in FPGAs) that communicate through a memory-mapped bus. In this case, the low-level communication protocol used by the Ports specified in the metadata must also be specified. They are presented as software blocks.

It is always possible to encapsulate the hardware block into a software block. To achieve that, suitable hardware and associated low-level protocols shall be added to the communication channels of the non-encapsulated hardware block, in order to make the block accessible from software.

5.4.2 Combination

MPAI-AIF supports the following ways of combining AIMs :

- *Software AIMs* connected to other software AIMs resulting in a software AIM
- *Non-encapsulated hardware blocks* connected to other non-encapsulated hardware blocks, resulting in a larger, non-encapsulated hardware AIM
- *Encapsulated hardware blocks* connected to either other encapsulated hardware blocks or other software blocks, resulting in a larger software AIM.
- Connection between a non-encapsulated hardware AIM and a software AIM is not supported as in such a case direct communication between the AIMs cannot be defined in any meaningful way.

5.4.3 Hardware-software compatibility

The possibility to always connect communication channels with one another shall be ensured, irrespective of whether the channel has been implemented in software or hardware. This results in the following matrix of requirements:

	Hardware	**Software**
Hardware	In structures, each named type is transmitted as a separate channel. Vector types are implemented as two channels, one transmitting the size and the second transmitting the data	Data structures are turned into a default strategy to fill out memory buffers, obtained by recursively traversing the definition (breadth-first). Sub-fields are written down according to their type, in little-endian order
Software	Data structures are turned into a default strategy to fill out memory buffers, obtained by recursively traversing the definition (breadth-first). Sub-fields are written down according to their type, in little-endian order	Data structures are turned into a default strategy to fill out memory buffers, obtained by recursively traversing the definition (breadth-first). Sub-fields are written down according to their type, in little-endian order

Both software-hardware and hardware-software cases require the specification of a memory-mapped communication protocol in order to allow communication between architectures.

5.4.4 Actual implementations

5.4.4.1 Hardware

Code implementations in hardware are defined as stand-alone objects. Metadata ensures that hardware blocks can be directly connected to other hardware/software blocks, provided the specification platforms for the two blocks are compatible. Hence, specifications for hardware

components only rely upon one or more implementations in code (source or binary) and their associated metadata.

5.4.4.2 Software

Software implementations must rely on a support library provided by the AIM Implementers in order to make sure that communication between the different constituent AIMs, and with other AIMs outside the block, is performed correctly. The support library can have different implementations, each one specific to one platform/operating system/programming language.
In addition, AIM Implementations must contain a number of well-defined steps so as to ensure that the controller is correctly initialised and remains in a consistent internal state, i.e.:

1. A code segment registering the different AIMs used by the AIW. The AIMs can be defined as:
    a. Source code, which can be local or remote
    b. Local instances of AIM Implementations that have been previously registered with the MPAI Store. Such AIMs are software functions or hardware designs executed on a local machine/HPC cluster/MPC machine.
    c. Remote instances that have been previously registered with the MPAI Store. They are AIMs executed on a remote machine.
    d. They are transparently executed by a remote Controller, and the only relevant point about them is their communication ports and protocols. The Controller automatically takes care of selecting an implementation of the AIM which is suitable for the hosting machine.
    e. Registering an AIM with the Controller results in it retrieving a suitable implementation function for the AIM, either from local code or from the MPAI Store.
2. Code starting/stopping the AIMs.
3. Code registering the input/output Ports for the AIM.
4. Code instantiating unicast channels between AIM Ports belonging to AIMs used by the module, and connections from/to the AIM being defined to/from external AIMs.
5. Registering Ports and connecting them may result in a number of steps performed by the Controller – some suitable data structure (including, for instance, data buffers) will be allocated for each port or channel, in order to support the functions specified by the communication API.
6. It is also possible to explicitly write/read data to/from, any of the existing Ports.
7. In general, arbitrary functionality can be added to a software AIM. For instance, depending on the AIM Function, one would typically link libraries that allow a GPU or FPGA to be managed through DMA, or link and use high-level libraries (such as TensorFlow) that implement AI-related functionality.
8. The API implementation depends on the architecture the Implementation is designed for.

6 Metadata

Metadata specifies static properties pertaining to the interaction between the AIW and the Controller. The configuration is specified in JSON as specified in the following sections.

6.1 Communication channels and their data types

6.1.1 Type system

The data interchange happening through buffers involves the exchange of structured data.
Data types for the record exchanged through Ports and communication Channels are defined by the following BNF specification. Words in bold typeface are keywords; capitalised words such as NAME are tokens.

```
fifo_type :=
       | /* The empty type */
       | base_type NAME
recursive_type :=
       | recursive_base_type NAME
base_type :=
       | toplevel_base_type
       | recursive_base_type
       | ( base_type )
toplevel_base_type :=
       | array_type
       | toplevel_struct_type
       | toplevel_variant_type
array_type :=
       | recursive_base_type []
toplevel_struct_type :=
       | { one_or_more_fifo_types_struct }
one_or_more_fifo_types_struct :=
       | fifo_type
       | fifo_type ; one_or_more_fifo_types_struct
toplevel_variant_type :=
       | { one_or_more_fifo_types_variant }
one_or_more_fifo_types_variant :=
       | fifo_type | fifo_type
       | fifo_type | one_or_more_fifo_types_variant
recursive_base_type :=
       | signed_type
       | unsigned_type
       | float_type
       | struct_type
       | variant_type
signed_type :=
       | int8
       | int16
       | int32
       | int64
unsigned_type :=
       | uint8 | byte
       | uint16
       | uint32
       | uint64
float_type :=
       | float32
       | float64
struct_type :=
       | { one_or_more_recursive_types_struct }
one_or_more_recursive_types_struct :=
       | recursive_type
       | recursive_type ; one_or_more_recursive_types_struct
variant_type :=
       | { one_or_more_recursive_types_variant }
one_or_more_recursive_types_variant :=
       | recursive_type | recursive_type
       | recursive_type | one_or_more_recursive_types_variant
```

Valid types for FIFOs are those defined by the production `fifo_type`.
Note that arrays can only occur at top level, so as to make the offsets for subtypes computable.
Although by using this language it is perfectly possible to specify types having a fixed length, the general record type written to, or read from, the port will not have a fixed length. However, some

modules (especially those implemented both in software and hardware) might need to keep the record type simpler.

6.1.2 Mapping the type to buffer contents

From a definition in this language, an automated way of filling and transmitting buffers can be derived both for hardware and software implementations. In particular, data structures are turned into low-level memory buffers, filled out by recursively traversing the definition (breadth-first). Sub-fields are written down according to their type, in little-endian order.
For instance, a definition for transmitting a video frame through a FIFO might be:

```
{int32 frameNumber; int16 x; int16 y; byte[] frame} frame_t
```

and the corresponding memory layout would be
`[32 bits: frameNumber | 16 bits: x | 16 bits: y | 32 bits: size(frame) | 8*size(frame) bits: frame]`.
API functions are provided in order to parse the content of raw memory buffers in a platform- and implementation-independent fashion (see Subsection 7.5.9).

6.2 AIF Metadata

Table 2 – AIF Metadata

Field name	Field syntax	Field description	M/O
ImplementerID	Number	Provided by MPAI Store	M
Version	String	Provided by Implementer. Replaced by "*" in Technical Specifications.	M
Profile	Enumeration	Defined by MPAI, selected by Implementer	M
ResourcePolicies	Structure array		O
- Name	String	An entry in the MPAI-specified Ontologies.	
- Minimum	Structure		O
o Memory	Integer	Memory size in KBytes	O
o CPUClass	Enumeration	Reserved for future definition	O
o CPULimit	Integer	Number of CPUs allocatable	O
- Maximum			O
o Memory	Integer	Memory size in KBytes	O
o CPUClass	Enumeration	Reserved for future definition	O
o CPULimit	Integer	Number of CPUs allocatable	O
- Request	Structure		O
o Memory	Integer	Memory size in KBytes	O
o CPUClass	Enumeration	Reserved for future definition	O
o CPULimit	Integer	Number of CPUs allocatable	O
Authentication	Enumeration	An entry in the MPAI-specified Ontologies.	M
TimeBase	Structure	An entry in the MPAI-specified Ontologies.	O

AI Identifiers shall be represented by ImplementerID, Version and Profile fields in JSON format.

6.3 AIW/AIM Metadata

AIM Metadata specifies static, abstract properties pertaining to one or more AIM implementations, and how the AIM will interact with the Controller. The configuration is specified in JSON, according to the following syntax:

Table 3 – AIW/AIM Metadata

Field name	Field syntax	Field description	M/O
ImplementerID	Number	Provided by MPAI Store	M
Standard	Structure	Data set identifying the standard implemented by AIM, if the AIM is standard. If absent, the UserDefined field must be present.	O
- Name	String	Defined by the standard itself	M
- Use_Case	Integer	Sequential in the order of the standard	M
- Version	String	Defined by the standard itself	M
- Profile	Enumeration	Defined by MPAI, selected by Implementer	M
UserDefined	Structure	Data set identifying an AIM not implementing a Standard. Mandatory if not an AIM atandard.	O
- Name	String	Provided by Implementer.	M
- Version	Number	Provided by Implementer.	M
AIFAPIProfile	Enumeration	Defined by MPAI, selected by implementer	M
Description	String	Free text describing the AIM	O
Types	Structure array	Defines Channel data types according to 6.1.1	O
Ports	Structure array	Optional. If present, all the fields of the structure shall be present.	O
- Name	String	Implementer-defined name	M
- Direction	Enumeration	Valid values: InputOutput or OutputInput	M
- Record_Type	String	Valid values: Channel Data Type defined in the Types dictionary or a Channel data type defined according to 6.1.1	M
- Type	Enumeration	Valid values: Software or Hardware	M
- Protocol	Enumeration	An entry in the MPAI-specified Ontology	M
AIMs	Structure array	Each record identifies an AIM according to the short-hand notation defined in **Error! Reference source not found.**.	M
Topology	Structure array	Array of Channels connecting one output to one input port	M
- Output	Structure	Identifies an AIM Port.	M
o AIMName	String	An AIM Identifier defined in **Error! Reference source not found.**. An empty string indicates a Port of the AIM being defined.	M
o PortName	String	A Port Identifier as defined in the corresponding field in the AIM.	M
- Input	Structure	Identifies an AIM Port.	M
o AIMName	String	An AIM Identifier defined in **Error! Reference source not found.**. An empty string indicates a Port of the AIM being defined.	M

o PortName	String	A Port Identifier as defined in the corresponding field in the AIM.	M
ResourcePolicies	Structure array		O
- Name	String	An entry in the MPAI-specified Ontologies.	
- Minimum	Structure		O
o Memory	Integer	Memory size in KBytes	O
o CPUClass	Enumeration	Reserved for future definition	O
o CPULimit	Integer	Number of CPUs allocatable	O
- Maximum			O
o Memory	Integer	Memory size in KBytes	O
o CPUClass	Enumeration	Reserved for future definition	O
o CPULimit	Integer	Number of CPUs allocatable	O
- Request	Structure		O
o Memory	Integer	Memory size in KBytes	O
o CPUClass	Enumeration	Reserved for future definition	O
o CPULimit	Integer	Number of CPUs allocatable	O
Documentation			O
- URI	String	According to [5]	O

AIW/AIM Identifiers shall be represented by ImplementerID and Standard or UserDefined fields in JSON format.

7 API

7.1 General

This section defines and describes a subset of the API of the software support library which is sufficient to implement the basic functionality of the Standard. More functions might be added in the future.

7.2 Conventions

For simplicity, the API is written in a C-like fashion. However, the specification should be meant as a definition for a general programming language.
Note that namespaces for modules, ports and communication channels (strings belonging to which are indicated in the next sections with names such as *`module_name`*, *`port_name`*, and *`channel_name`*, respectively) are all independent.

7.2.1 API types

We assume that the implementation defines a number of types, as follows:
`message_t`, the type of messages being passed through communication ports and channels
`parser_t`, the type of parsed message datatypes (a.k.a. “the high-level protocol”)
`error_t`, the type of return codes returned by library functions.

The exact types are opaque, and its exact definition is left to the implementer. The only meaningful way to operate on library types with defined results is by using library functions.
On the other hand, the type of modules, `module_t`, is always defined as

```
typedef error_t *(module_t)()
```

across all implementations, in order to ensure cross-compatibility.

Types such as `void`, `size_t`, `char`, `int` are regular C types.

7.2.2 Error codes

The following error codes having type `error_t` are returned by the library:

Code	Semantic value

`MPAI_AIF_OK`	The function returned successfully. It must always evaluate to 0, so that one can write tests on errors originating from a function `f` as
	`if (f(...)) {`
	`  // Error handler`
	`  ...`
	`}`

MPAI_AIF_ERROR	A generic error code
`MPAI_AIF_MEM_ALLOC`	Memory allocation error
MPAI_AIF_MODULE_NOT_FOUND	The operation requested of a module cannot be executed since the module has not been found
MPAI_AIF_INIT_ERROR	The AIW cannot be initialized
MPAI_AIF_TERM_ERROR	The AIW cannot be properly terminated
MPAI_AIF_MODULE_CREATION_FAILED	A new module cannot be created
MPAI_AIF_PORT_CREATION_FAILED	A new module port cannot be created
MPAI_AIF_CHANNEL_CREATION_FAILED	A new channel cannot be created
MPAI_AIF_WRITE_ERROR	A generic message writing error
MPAI_AIF_TOO_MANY_PENDING_MESSAGES	A message writing operation failed because there are too many pending messages waiting to be delivered
MPAI_AIF_PORT_NOT_FOUND	One or both ports of a connection has (or have) been removed
MPAI_AIF_READ_ERROR	A generic message reading error
MPAI_AIF_OP_FAILED	The requested operation failed

7.3 Controller API for MPAI Store

It is assumed that all the communication between the controller and the MPAI Store occur via https protocol. Thus the APIs reported refer to the http secure protocol functions (i.e. GET, POST, etc). The MPAI Store supports the GIT protocol [2]. The MPAI Store offers 3 REST interfaces: one for accessing the packages, i.e., assembled AIMs and/or AIW in a single file container, the second one for querying the MPAI Store database and the third one to handle GIT communications. The controller implements the functions relative to the file retrieval as described in 7.3.5.

7.3.1 Get file archive

Get an archive from the MPAI Store.

7.3.1.1 MPAI_AIF_Store_GetFile

```
error_t MPAI_AIF_Store_GetFile()
```

File Format default is tar.gz. Options are tar.gz, tar.bz2, tbz, tbz2, tb2, bz2, tar, and zip. For example, specifying archive.zip would send an archive in ZIP format [3].

7.4 Controller API for User Agent

7.4.1 General

The functions provided to the user agent are the following:

1. Initialise all the Components of the AIF.
2. Manage Start/Stop/Suspend/Resume.
3. Manages Resource Allocation.

7.4.1.1 MPAI_AIF_USER_Initialize

```
error_t MPAI_AIF_USER_Initialize()
```

Make sure that the Controller is properly switched on and initialized.

7.4.1.2 MPAI_AIF_USER_Destroy

```
error_t MPAI_AIF_USER_Destroy()
```

Switch off the Controller, after data structures related to running AIWs have been disposed of.

7.4.1.3 MPAI_AIF_USER _Register

```
error_t
  MPAI_AIF_USER _Register(const char* name, AIM_t pointer_to_function)
```

Register the AIW with the Controller, with name *name* and implementation *pointer_to_function*.

7.4.1.4 MPAI_AIF_USER _Deregister

```
error_t
  MPAI_AIF_USER _Deregister(const char* name, AIM_t pointer_to_function)
```

Deregister the AIW with the Controller, with name *name* and implementation *pointer_to_function*.

7.4.2 Start/Pause/Resume/Stop Messages to other AIMs

Note: Errors encountered while transmitting/receiving these Messages are non-recoverable – i.e., they terminate the entire AIW. AIMs can communicate with other AIMs and the Controller uses this API to Start/Pause/Resume/Stop the AIMs.

7.4.2.1 MPAI_AIF_USER_Start

```
error_t MPAI_AIF_USER_Start(const char* name)
```

Start the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.4.2.2 MPAI_AIF_USER_Pause

```
error_t MPAI_AIF_USER_Pause(const char* name)
```

Pause the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.4.2.3 MPAI_AIF_USER_Resume

```
error_t MPAI_AIF_USER_Resume(const char* name)
```

Resume the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.4.2.4 MPAI_AIF_USER_Stop

```
error_t MPAI_AIF_USER_Stop(const char* name)
```

Stop the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.4.3 Inquire about state of AIWs and AIMs

7.4.3.1 MPAI_AIF_USER_GetStatus

```
error_t MPAI_AIF_USER_GetStatus(const char* name, char* status)
```

7.4.4 Management of Global and Internal Storage for AIWs

7.4.4.1 MPAI_AIF_USER_GStorage_Init

```
error_t MPAI_AIF_USER_GStorage_init(const char* name)
```

7.4.4.2 MPAI_AIF_USER_IStorage_Init

```
error_t MPAI_AIF_USER_IStorage_init(const char* name)
```

7.4.5 Communication management

Communication takes place with Messages that can be communicated via Events or Ports and Channels. Their actual implementation and signal type depends on the MPAI-AIF implementation (and hence on the specific platform, operating system and programming language the implementation is defined for). Events are defined AIF wide while Ports, Channels and Messages are specific to the AIM and thus part of the AIM API.

7.4.5.1 MPAI _USER_Comm_Init

```
error_t MPAI_AIF_USER_Comm_init(const char* name)
```

7.4.5.2 MPAI _USER_Comm_Destroy

```
error_t MPAI_AIF_USER_Comm_Destroy(const char* name)
```

7.4.5.3 MPAI USER_Comm_Event

```
error_t MPAI_AIF_USER_Comm_Event(const char* name_event)
```

Actual Event handling is left to the AIM.

7.4.6 Resorce allocation management

```
error_t MPAI_AIF_USER_AIF_Resource_Allocation_init( int MinMem, int MaxMem, int
ReqMem, int MinCPU, int MaxCPU, int ReqCPU)
```

7.5 Controller API for AIMs

7.5.1 General

The functions executed by a AIW are:

1. Identify AIF, AIW, AIM, Storage, Use Case.
2. Describe the topology and connections of AIMs in the AIW.
3. Describe the Status of AIF.
4. Describe the Time base.
5. Describe the Resource policy.

7.5.2 Execution environment

These API calls allow AIMs to interrogate the Controller about details of the execution environment (e.g., the MPAI-AIF profile implemented by the API).

7.5.3 Initialization/Deinitialization

7.5.3.1 MPAI_AIF_Initialize

```
error_t MPAI_AIF_Initialize()
```

Make sure that the Controller is properly initialized and the AIF-specific data structures are created.

7.5.3.2 MPAI_AIF_Destroy

```
error_t MPAI_AIF_Destroy()
```

Controller destroys the AIW.

7.5.4 Register/deregister AIMs to the Controller

7.5.4.1 MPAI_AIF_AIM_Register

```
error_t
  MPAI_AIF_AIM_Register(const char* name, AIM_t pointer_to_function)
```

Register the AIM with the Controller, with name *name* and implementation *pointer_to_function*.

7.5.4.2 MPAI_AIF_AIM_Deregister

```
error_t
  MPAI_AIF_AIM_Deregister(const char* name, AIM_t pointer_to_function)
```

Deregister the AIM with the Controller, with name *name* and implementation *pointer_to_function*.

7.5.4.3 MPAI_AIF_AIM_Use_Local_ByCode

```
error_t
  MPAI_AIF_AIM_Use_Local_ByCode(const char* name, AIM_t pointer_to_function)
```

Declare to the Controller that the AIM depends on another AIM, to be run locally, with name *name* and implementation *pointer_to_function*. This API call may be executed only in a Trusted Zone The source code must be available in the same implementation unit.

7.5.4.4 MPAI_AIF_AIM_Use_Local_ByID

```
error_t MPAI_AIF_AIM_Use_Local_ByID(const char* name, const char* ID)
```

Declare to the Controller that the AIM depends on another AIM, to be run locally, with name *name* and implementation identified by the MPAI-AIF *ID*. A binary implementation corresponding to the *ID* and matching the platform on which the AIW is being executed must be available from the MPAI Store.

7.5.4.5 MPAI_AIF_AIM_Use_Remote

Declare to the Controller that the AIM depends on another AIM, to be run on the remote MPAI-AIF instance identified by URI *URI*, with name *name* and implementation identified by the MPAI-AIF *ID*. A binary implementation corresponding to the *ID* and matching the platform on which the AIW for the server available at *URI* is being executed must be available from the MPAI Store.

7.5.5 Start/Pause/Resume/Stop Messages to other AIMs

Note: Errors encountered while transmitting/receiving these Messages are non-recoverable – i.e., they terminate the entire AIW. AIMs can communicate with other AIMs and the Controller uses this API to Start/Pause/Resume/Stop the AIMs.

7.5.5.1 MPAI_AIF_AIM_Start

```
error_t MPAI_AIF_AIM_Start(const char* name)
```

Start the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.5.5.2 MPAI_AIF_AIM_Pause

```
error_t MPAI_AIF_AIM_Pause(const char* name)
```

Pause the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.5.5.3 MPAI_AIF_AIM_Resume

```
error_t MPAI_AIF_AIM_Resume(const char* name)
```

Resume the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.5.5.4 MPAI_AIF_AIM_Stop

```
error_t MPAI_AIF_AIM_Stop(const char* name)
```

Stop the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.5.5.5 MPAI_AIF_AIM_EventHandler

```
error_t MPAI_AIF_AIM_EventHandler(const char* name)
```

Create EventHandler for the AIM with given name `name`. If the operation succeeds, it has immediate effect.

7.5.6 Registering AIM Ports

Note that the following calls define *two* Ports, one in input and one in output. This is done because for each input/output external Port there always is one implicit output/input internal Port. By convention, internal Ports can be accessed from within AIM code by giving *“”* as the AIM name (there is no assigned name for the AIM being defined). External code will use the external Ports, and refer to the AIM by using its assigned name.

7.5.6.1 MPAI_AIF_Port_InputOutput_Create

```
error_t MPAI_AIF_Port_InputOutput_Create(
          const char* name, size_t maxMessages, size_t maxMemory)
```

Define a new output port with the given name for the calling AIM. *maxMessages* and *maxMemory* define the maximum number of pending messages and total buffer memory in bytes, respectively, according to the Resource Allocation Policy. A value of 0 for any of the arguments uses the defaults provided by the Controller.

7.5.6.2 MPAI_AIF_Port_InputOutput_Destroy

```
error_t MPAI_AIF_Port_InputOutput_Destroy(
          const char* name)
```

Destroy an output port with the given name.

7.5.6.3 MPAI_AIF_Port_OutputInput_Create

```
error_t MPAI_AIF_Port_OutputInput_Create(
          const char* name, size_t maxMessages, size_t maxMemory)
```

Define a new input port with the given name for the calling AIM. *maxMessages* and *maxMemory* define the maximum number of pending messages and total buffer memory in bytes, respectively, according to the Resource Allocation Policy. A value of 0 for any of the arguments uses the defaults provided by the Controller.

7.5.6.4 MPAI_AIF_Port_OutputInput_Destroy

```
error_t MPAI_AIF_Port_OutputInput_Destroy(
          const char* name)
```

Destroy an input port with the given name.

7.5.6.5 MPAI_AIF_Port_InputOutput_Open

```
error_t MPAI_AIF_Port_InputOutput_Open(const char* name)
```

Open the output Port *name*. This call is needed before any Message can be transmitted through the Port.

7.5.6.6 MPAI_AIF_Port_InputOutput_Close

```
error_t MPAI_AIF_Port_InputOutput_Close(const char* name)
```

Close the output Port *name*. After this call no more Message can be transmitted through the Port.

7.5.6.7 MPAI_AIF_Port_OutputInput_Open

```
error_t MPAI_AIF_Port_OutputInput_Open(const char* name)
```

Open the input Port *name*. This call is needed before any Message can be transmitted through the Port.

7.5.6.8 MPAI_AIF_Port_OutputInput_Close

```
error_t MPAI_AIF_Port_OutputInput_Close(const char* name)
```

Close the output Port *name*. After this call no more Message can be transmitted through the Port.

7.5.7 Register Connections between AIMs

7.5.7.1 MPAI_AIF_Channel_Create

```
error_t
  MPAI_AIF_Channel_Create(const char* name, const char* out_AIM_name, const char*
out_port_name, const char* in_AIM_name, const char* in_port_name)
```

Create a new interconnecting channel between an output port and an input port. AIM and port names are specified with the name used when constructed.
Note: The Channel identifies and connects one output Port to an input Port.

7.5.7.2 MPAI_AIF_Channel_Destroy

```
error_t
  MPAI_AIF_Channel_Destroy(const char* name)
```

Destroy the channel with name *name*. This API Call closes all Ports related to the Channel.

7.5.8 Using Ports

7.5.8.1 MPAI_AIF_Port_Output_Read

```
message_t* MPAI_AIF_Port_Output_Read(
            const char* AIM_name, const char* port_name)
```

Reads a message from the port identified by (*AIM_name*,*port_name*). The read is blocking. Hence, in order to avoid deadlocks, one should first probe the port with `MPAI_AIF_Port_Probe`. It returns a copy of the original message.

7.5.8.2 MPAI_AIF_Port_Input_Write

```
error_t MPAI_AIF_Port_Input_Write(
          const char* AIM_name, const char* port_name, message_t* message)
```

Writes a message *message* to the port identified by (*AIM_name*,*port_name*). The write is blocking. Hence, in order to avoid deadlocks one should first probe the port with `MPAI_AIF_Port_Probe`. The message being transmitted must remain available until the function returns, or the behaviour will be undefined.

7.5.8.3 MPAI_AIF_Port_Reset

```
error_t MPAI_AIF_Port_Reset(const char* AIM_name, const char* port_name)
```

Reset an input or output port identified by (*AIM_name*,*port_name*) by deleting all the pending messages associated with it.

7.5.8.4 MPAI_AIF_Port_CountPendingMessages

```
size_t MPAI_AIF_Port_CountPendingMessages(
         const char* AIM_name, const char* port_name)
```

This function returns the number of pending messages on a input or output port identified by (*AIM_name*,*port_name*).

7.5.8.5 MPAI_AIF_Port_Probe

```
error_t MPAI_AIF_Port_Probe(const char* port_name, message_t* message)
```

For this function, a return value of `MPAI_AIF_OK` means that one can write to the port if the port is a FIFO input port, or that data is available to be read from the port if the port is a FIFO output port.

7.5.8.6 MPAI_AIF_Port_Select

```
int MPAI_AIF_Port_Output_Select(
       const char* AIM_name_1,const char* port_name_1,...)
```

Given a list of output Ports, returns the index of one Port for which data has become available in the meantime. The call is blocking to address potential race conditions.

7.5.9 Operations on messages

All implementations must provide a common Message passing functionality which is abstracted by the following functions.

7.5.9.1 MPAI_AIF_Message_Copy

```
message_t* MPAI_AIF_Message_Copy(message_t* message)
```

Make a copy of a message structure *message*.

7.5.9.2 MPAI_AIF_Message_Delete

```
message_t* MPAI_AIF_Message_Delete(message_t* message)
```

Delete a message *message* and its allocated memory.

7.5.9.3 MPAI_AIF_Message_GetBuffer

```
void* MPAI_AIF_Message_GetBuffer(message_t* message)
```

Get access to the low-level memory buffer associated with a message structure *message*.

7.5.9.4 MPAI_AIF_Message_GetBufferLength

```
size_t MPAI_AIF_Message_GetBufferLength(message_t* message)
```

Get the size in bits of the low-level memory buffer associated with a message structure *message*.

7.5.9.5 MPAI_AIF_Message_Parse

```
parser_t* MPAI_AIF_Message_Parse (const char* type)
```

Create a parsed representation of the data type defined in *type* according to the metadata syntax defined in section 3.7, Type system, in order to facilitate the successive parsing of raw memory buffers associated with message structures (see functions below).

7.5.9.6 MPAI_AIF_Message_Parse_Get_StructField

```
void* MPAI_AIF_Message_Parse_Get_StructField(
        parser_t* parser, void* buffer, const char* field_name)
```

Assume that the low-level memory buffer *buffer* contains data of type `struct_type` whose complete parsed type definition (specified according to the metadata syntax defined in section 3.7, Type system) can be found in *parser*. Fetch the member of the `struct_type` named *field_name*, and return it in a freshly allocated low-level memory buffer. If a member with such name does not exist, return NULL.

7.5.9.7 MPAI_AIF_Message_Parse_Get_VariantType

```
void* MPAI_AIF_Message_Parse_Get_VariantType(
        parser_t* parser, void* buffer, const char* type_name)
```

Assume that the low-level memory buffer *buffer* contains data of type `variant_type` whose complete parsed type definition (specified according to the metadata syntax defined in section 3.7, Type system) can be found in *parser*. Fetch the member of the `variant_type` named *field_name*, and return it in a freshly allocated low-level memory buffer. If a member with such name does not exist, return NULL.

7.5.9.8 MPAI_AIF_Message_Parse_Get_ArrayLength

```
int MPAI_AIF_Message_Parse_Get_ArrayLength(parser_t* parser, void* buffer)
```

Assume that the low-level memory buffer *buffer* contains data of type `array_type` whose complete parsed type definition (specified according to the metadata syntax defined in section 3.7, Type system) can be found in *parser*. Retrieve the length of such an array. If the buffer does not contain an array, return -1.

7.5.9.9 MPAI_AIF_Message_Parse_Get_ArrayField

```
void* MPAI_AIF_Message_Parse_Get_ArrayField(
        parser_t* parser, void* buffer, const int field_num)
```

Assume that the low-level memory buffer *buffer* contains data of type `array_type` whose complete parsed type definition (specified according to the metadata syntax defined in section 3.7, Type system) can be found in *parser*. Fetch the element of the `array_type` named *field_num*, and return it in a freshly allocated low-level memory buffer. If such element does not exist, return NULL.

7.5.9.10 MPAI_AIF_Message_Parse_Delete

```
void MPAI_AIF_Message_Parse_Delete(parser_t* parser)
```

Delete the parsed representation of a data type defined by *parser*, and deallocate all memory associated to it.

7.5.10 Functions specific to machine learning

7.5.10.1 Support for model update

The following API are provided to support AIM ML model update.
Such update occurs via the MPAI Store by using the MPAI Store specific APIs or via Global (GStorage) or Internal (IStorage) storage by using the specified APIs.
The Golobal and/or Internal storage needs to be initialized via the corresponding API and then the secure protocol of choice (as specified in the AIF metadata) can be used for the transfer.

8 Implementation Guidelines (Informative)

This chapter is to be fully developed. It will provide guidelines for implementation of:

- Message queues
- Control structures
- Messages vs Events
- Support for HW and SW
- Support for local and remote AIMs
- Scope of programming language dependence
- Scheduling of AIMs and AIWs

9 Examples (Informative)

9.1 AIF Implementations

The following two informative examples are high-level descriptions of possible AIF operations:

9.1.1 Resource-constrained implementation

1. Controller is a single process that implements the AIW and operates based on interrupts call-backs
2. AIF is instantiated via a secure communication interface
3. AIMs can be local or has been instantiated through a secure communication interface
4. Controller initialises the AIF
5. AIF asks the AIMs to be instantiated
6. Controller manages the Events and Messages

7. User Agent can act on the AIWs at the request of the user.

9.1.2 Non-resource-constrained implementation

1. Controller and AIW are two independent processes
8. Controller manages the Events and Messages
2. AIW contacts Controller on Communication and authenticates itself
3. Controller requests AIW configuration metadata
4. AIW sends Controller the configuration metadata
5. The implementation of the AIW can be local or can be downloaded from the MPAI Store
6. Controller authenticates itself with the MPAI Store and requests implementations for the needed AIMs listed in the metadata from the MPAI Store
7. MPAI Store sends the requested AIM implementations and the configuration metadata
8. Controller
    a. Instantiates the AIMs specified in the AIW metadata
    b. Manages their communication and resources by sending Messages to AIMs.
9. User Agent can gain control of AIWs running on the Controller via a specific Controller API, e.g., User Agent can test conformance of a AIW with an MPAI standard through a dedicated API call.

9.2 Examples of types

```
byte[] bitstream_t
```

An array of `bytes`, with variable length.

```
{int32 frameNumber; int16 x; int16 y; byte[] frame} frame_t
```

A `struct_type` with 4 members named `frameNumber`, `x`, `y`, and `frame` — they are an `int32`, an `int16`, an `int16`, and an array of `bytes` with variable length, respectively.

```
{int32 i32 | int64 i64} variant_t
```

A `variant_type` that can be either an `int32` or an `int64`.

9.3 Examples of Metadata

9.3.1 AIF Metadata

```
{
	"AIF": {
		"ImplementerID": 100,
		"Version": "*",
		"Profile": "Main",
		"Description": "",
		"ResourcePolicies": [
			{
				"Name": "CPU",
				"Minimum": {
					"Memory": 50000,
					"CPUClass": "OntologyEntry",
					"CPULimit": 1
				},
				"Maximum": {
					"Memory": 150000,
					"CPUClass": "OntologyEntry",
					"CPULimit": 4
				},
				"Request": {
					"Memory": 100000,
					"CPUClass": "OntologyEntry",
					"CPULimit": 2
				}
			}
```

```
                ],
                "Authentication": "OntologyEntry",
                "TimeBase": "OntologyEntry",
                "UserAPIProfile": "Low.V",
                "ControllerAPIProfile": {
                        Version: "27",
                        Level: "High"
                },
                "Implementations": [
                ],
                "Documentation": [
                        { "URI": "https://mpai.community/standards/mpai-aif/"
                        }
                ]
        }
}
```

9.3.2 AIW Metadata

The following example provides the AIW Metadata of the MPAI-MMC UST Use Case.

```
{
   "AIW":{
      "ImplementerID": 100,
      "Standard":{
         "Name":"MMC",
         "Use_Case": UST,
         "Version":"1",
         "Profile":"Main"
      },
      "Description":"This AIW implements UST application of MPAI-MMC",
      "Types":[
         {
            "Text_t":"{byte[] One_Byte_Text | uint16[] Two_Byte_Text}",
            "Speech_t":"uint16[]",
            "InputSelection_t":"{Enum Text | Enum Speech}",
            "Language_t":"uint8[]"
         }
      ],
      "Ports":[
         {
            "Name":"InputSelection",
            "Direction":"InputOutput",
            "Record_Type":"InputSelection_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"RequestedLanguage",
            "Direction":"InputOutput",
            "Record_Type":"uint8[5] Language_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"InputText",
            "Direction":"InputOutput",
            "Record_Type":"Text_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"InputSpeech1",
            "Direction":"InputOutput",
            "Record_Type":"Speech_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"InputSpeech2",
            "Direction":"InputOutput",
            "Record_Type":"Speech_t",
```

```
      "Type":"Software",
      "Protocol":""
   },
   {
      "Name":"TranslatedText",
      "Direction":"OutputInput",
      "Record_Type":"Text_t",
      "Type":"Software",
      "Protocol":""
   },
   {
      "Name":"TranslatedSpeech",
      "Direction":"OutputInput",
      "Record_Type":"Speech_t",
      "Type":"Software",
      "Protocol":""
   }
],
"AIMs":[
   {
      "SpeechRecogniton":"@*:(S:(MMC:UST:2:SpeechRecogniton)):*",
      "Translation":"*:(S:(MMC:UST:2:Translation)):*",
      "SpeechFeatureExtraction":"@*:(S:(MMC:UST:2:LanguageUnderstanding)):*",
      "SpeechSynthesis":"@*:(S:(MMC:UST:2:SpeechSynthesis)):*"
   }
],
"Topology":[
   {
      "RequestedLanguage":{
         "Output":{
            "Module":"",
            "Port":"RequestedLanguage"
         },
         "Input":{
            "Module":"Translation",
            "Port":"RequestedLanguage"
         }
      },
      "InputText":{
         "Output":{
            "Module":"",
            "Port":"InputText"
         },
         "Input":{
            "Module":"Translation",
            "Port":"InputText "
         }
      },
      "InputSpeech1":{
         "Output":{
            "Module":"",
            "Port":"InputSpeech1"
         },
         "Input":{
            "Module":"SpeechRecognition",
            "Port":"InputSpeech1"
         }
      },
      "InputSpeech2":{
         "Output":{
            "Module":"",
            "Port":"InputSpeech2"
         },
         "Input":{
            "Module":"SpeechFeatureExtraction",
            "Port":"InputSpeech2"
         }
      },
      "OutputSpeech":{
         "Output":{
            "Module":"SpeechSynthesis",
            "Port":"TranslatedSpeech"
         },
         "Input":{
```

```
                  "Module":"",
                  "Port":"TranslatedSpeech"
               }
            },
            "SpeechFeatures":{
               "Output":{
                  "Module":"SpeechFeatureExtraction",
                  "Port":"SpeechFeatures"
               },
               "Input":{
                  "Module":"SpeechSynthesis",
                  "Port":"SpeechFeatures"
               }
            },
            "RecognizedText":{
               "Output":{
                  "Module":"SpeechRecognition",
                  "Port":"RecognizedText"
               },
               "Input":{
                  "Module":"Translation",
                  "Port":"RecognizedText"
               }
            },
            "TranslatedText":{
               "Output":{
                  "Module":"Translation",
                  "Port":"TranslatedText"
               },
               "Input":{
                  "Module":"SpeechSynthesis",
                  "Port":"TranslatedText"
               }
            },
            "OutputText":{
               "Output":{
                  "Module":"Translation",
                  "Port":"TranslatedText"
               },
               "Input":{
                  "Module":"",
                  "Port":"TranslatedText"
               }
            }
         }
      ],
      "ResourcePolicies": [
        {
                     "Name": "CPU",
                            "Minimum": {
                                   "Memory": 50000,
                                   "CPUClass": "OntologyEntry",
                                   "CPULimit": 1
                            },
                            "Maximum": {
                                   "Memory": 100000,
                                   "CPUClass": "OntologyEntry",
                                   "CPULimit": 2
                            },
                            "Request": {
                                   "Memory": 75000,
                                   "CPUClass": "OntologyEntry",
                                   "CPULimit": 1
                            }
                     }
            ],
      "Documentation":[
         {
            "Type":"tutorial",
            "URI":"https://mpai.community/standards/mpai-mmc/"
         }
      ]
   }
}
```

9.3.3 AIM Metadata

9.3.3.1 SpeechRecognition


```
{
   "AIM":{
      "ImplementerID": 100,
      "Standard":{
         "Name": "SpeechRecognition",
         "Use_Case": UST,
         "Version":"1",
         "Profile":"Main"
      },
      "Description":"This AIM implements speech recognition function for UST that converts
speech to text of user utterance.",
      "Types":[
         {
            "Text_t":"{byte[] One_Byte_Text | uint16[] Two_Byte_Text}",
            "Speech_t":"uint16[]"
         }
      ],
      "Ports":[
         {
            "Name":"InputSpeech1",
            "Direction":"InputOutput",
            "Record_Type":"Speech_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"RecognizedText",
            "Direction":"OutputInput",
            "Record_Type":"Text_t",
            "Type":"Software",
            "Protocol":""
         }
      ],
      "AIMs":[

      ],
      "Topology":[

      ],

      "Documentation":[
         {
            "Type":"tutorial",
            "URI":"https://mpai.community/standards/mpai-mmc/"
         }
      ]
   }
}
```


9.3.3.2 Translation


```
{
   "AIM":{
      "ImplementerID": 100,
      "Standard":{
         "Name":"SpeechTranslation",
         "Use_Case": UST,
         "Version":"1",
         "Profile":"Main"
      },
      "Description":"This AIM implements Translation function.",
      "Types":[
         {
            "Text_t":"{byte[] One_Byte_Text | uint16[] Two_Byte_Text}",
            "InputSelection_t":"{Enum Text | Enum Speech}",
            "Language_t":"uint8[]"
         }
      ],
      "Ports":[
```

```
            {
               "Name":"InputSelection",
               "Direction":"InputOutput",
               "Record_Type":"InputSelection_t",
               "Type":"Software",
               "Protocol":""
            },
            {
               "Name":"RequestedLanguage",
               "Direction":"InputOutput",
               "Record_Type":"uint8[5] Language_t",
               "Type":"Software",
               "Protocol":""
            },
            {
               "Name":"InputText",
               "Direction":"InputOutput",
               "Record_Type":"Text_t",
               "Type":"Software",
               "Protocol":""
            },
            {
               "Name":"OutputText",
               "Direction":"OutputInput",
               "Record_Type":"Text_t",
               "Type":"Software",
               "Protocol":""
            },
            {
               "Name":"TranslatedText",
               "Direction":"OutputInput",
               "Record_Type":"Text_t",
               "Type":"Software",
               "Protocol":""
            }
         ],
         "AIMs":[

         ],
         "Topology":[

         ],
         "Documentation":[
            {
               "Type":"tutorial",
               "URI":"https://mpai.community/standards/mpai-mmc/"
            }
         ]
      }
   }
```

9.3.3.3 Speech Feature Extraction

```
{
   "AIM":{
      "ImplementerID": 100,
      "Standard":{
         "Name": "SpeechFeatureExtraction",
         "Use_Case": UST,
         "Version":"1",
         "Profile":"Main"
      },
      "Description":"This AIM implements Speech Feature Extraction function.",
      "Types":[
         {
            "Speech_t":"uint16[]",
            "SpeechFeatures_t":"{byte pitch; string<256 tone; string<256 intonation;
string<256 intensity; string<256 speed; Emotion_t emotion; float32[] NNspeechFeatures}"
         }
      ],
      "Ports":[
         {
            "Name":"InputSpeech2",
            "Direction":"InputOutput",
```

```
            "Record_Type":"Speech_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"SpeechFeatures",
            "Direction":"OutputInput",
            "Record_Type":"SpeechFeatures_t",
            "Type":"Software",
            "Protocol":""
         }
      ],
      "AIMs":[

      ],
      "Topology":[

      ],
      "Documentation":[
         {
            "Type":"tutorial",
            "URI":"https://mpai.community/standards/mpai-mmc/"
         }
      ]
   }
}
```

9.3.3.4 Speech Synthesis

```
{
   "AIM":{
      "ImplementerID": 100,
      "Standard":{
         "Name":"SpeechSynthesis",
         "Use_Case": UST,
         "Version":"1",
         "Profile":"Main"
      },
      "Description":"This AIM implements Speech Synthesis function.",
      "Types":[
         {
            "Text_t":"{byte[] One_Byte_Text | uint16[] Two_Byte_Text}",
            "Speech_t":"uint16[]",
            "SpeechFeatures_t":"{byte pitch; string<256 tone; string<256 intonation;
string<256 intensity; string<256 speed; Emotion_t emotion; float32[] NNspeechFeatures}"
         }
      ],
      "Ports":[
         {
            "Name":"TranslatedText",
            "Direction":"InputOutput",
            "Record_Type":"Text_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"SpeechFeatures",
            "Direction":"InputOutput",
            "Record_Type":"SpeechFeatures_t",
            "Type":"Software",
            "Protocol":""
         },
         {
            "Name":"OutputSpeech",
            "Direction":"OutputInput",
            "Record_Type":"Speech_t",
            "Type":"Software",
            "Protocol":""
         }
      ],
      "AIMs":[

      ],
      "Topology":[
```

```
      ],
      "Documentation":[
         {
            "Type":"tutorial",
            "URI":"https://mpai.community/standards/mpai-mmc/"
         }
      ]
   }
}
```

Annex 1 – MPAI-wide terms and definitions (Normative)

The Terms used in this standard whose first letter is capital and are not already included in *Table 1* are defined in *Table 4.*

Table 4 – MPAI-wide Terms

Term	Definition
Access	Static or slowly changing data that are required by an application such as domain knowledge data, data models, etc.
AI Framework (AIF)	The environment where AIWs are executed.
AI Workflow (AIW)	An organised aggregation of AIMs implementing a Use Case receiving AIM-specific Inputs and producing AIM-specific Outputs according to its Function.
AI Module (AIM)	A processing element receiving AIM-specific Inputs and producing AIM-specific Outputs according to according to its Function.
Application	A usage domain target of an Application Standard
Channel	A connection between an output port of an AIM and an input port of an AIM. The term "connection" is also used as synonymous.
Communication	The infrastructure that implements message passing between AIMs
Component	One of the 7 AIF elements: Access, Communication, Controller, Internal Storage, Global Storage, MPAI Store, and User Agent
Conformance	The attribute of an Implementation of being a correct technical Implementation of a Technical Specification.
Conformance Tester	An entity authorised by MPAI to Test the Conformance of an Implementation.
Conformance Testing	The normative document specifying the Means to Test the Conformance of an Implementation.
Conformance Testing Means	Procedures, tools, data sets and/or data set characteristics to Test the Conformance of an Implementation.
Connection	A channel connecting an output port of an AIM and an input port of an AIM.
Controller	A Component that manages and controls the AIMs in the AIF, so that they execute in the correct order and at the time when they are needed
Data format	The standard digital representation of data and their semantics.
Ecosystem	The ensemble of the following actors: MPAI, MPAI Store, Implementers, Conformance Testers, Performance Testers and Users of MPAI-AIF Implementations as needed to enable an Interoperability Level.
Explainability	The ability to trace the output of an Implementation back to the inputs that have produced it.
Fairness	The attribute of an Implementation whose extent of applicability can be assessed by making the training set and/or network open to testing for bias and unanticipated results.
Function	The operations effected by an AIW or an AIM on input data.
Global Storage	A Component to store data shared by AIMs.
Internal Storage	A Component to store data of the individual AIMs.
Identifier	A name that uniquely identifies an Implementation.
Implementation	1. An embodiment of the MPAI-AIF Technical Specification, or

	2. An AIW or AIM of a particular Level (1-2-3) conforming with a Use Case of an MPAI Application Standard.
Interoperability	The ability to functionally replace an AIW or AIM with another AIW of AIM having the same Interoperability Level
Interoperability Level	The attribute of an AIW and its AIMs to be executable in an AIF Implementation and to be proprietary (Level 1) or to pass the Conformance Testing (Level 2) or the Performance Testing (Level 3) of an MPAI Application Standard.
Knowledge Base	Structured and/or unstructured information made accessible to AIMs via MPAI-specified interfaces
Message	A sequence of Records transported by Communication through Channels.
Normativity	The set of attributes of a technology or a set of technologies specified by the applicable parts of an MPAI standard.
Performance	The attribute of an Implementation of being Reliable, Robust, Fair and Replicable.
Performance Assessment	The normative document specifying the procedures, the tools, the data sets and/or the data set characteristics to Assess the Grade of Performance of an Implementation.
Performance Assessment Means	Procedures, tools, data sets and/or data set characteristics to Assess the Performance of an Implementation.
Performance Assessor	An entity authorised by MPAI to Assess the Performance of an Implementation in a given Application domain
Profile	A particular subset of the technologies used in MPAI-AIF or an AIW of an Application Standard and, where applicable, the classes, other subsets, options and parameters relevant to that subset.
Record	A data structure with a specified structure
Reference Software	A technically correct software implementation of a Technical Specification containing source code, or source and compiled code.
Reliability	The attribute of an Implementation that performs as specified by the Application Standard, profile and version the Implementation refers to, e.g., within the application scope, stated limitations, and for the period of time specified by the Implementer.
Replicability	The attribute of an Implementation whose Performance, as Assessed by a Performance Assessor, can be replicated, within an agreed level, by another Performance Assessor.
Robustness	The attribute of an Implementation that copes with data outside of the stated application scope with an estimated degree of confidence.
Service Provider	An entrepreneur who offers an Implementation as a service (e.g., a recommendation service) to Users.
Standard	The ensemble of Technical Specification, Reference Software, Conformance Testing and Performance Assessment of an MPAI application Standard.
Technical Specification	(Framework) the normative specification of the AI Framework. (Application) the normative specification of the set of Use Cases belonging to an Application Domain along with the AIMs required to Implement the Use Cases. the collection of Use Cases relevant to the Application Domain that include: 1. The formats of the Input/Output data of the AIWs implementing the Use Cases. 2. The Topology of the AIMs of the AIWs.

	3. The formats of the Input/Output data of the AIMs belonging the AIW.
Time Base	The protocol specifying how Components can access timing information
Topology	The set of AIM Connections of an AIW.
Use Case	A particular instance of the Application domain target of an Application Standard.
User	A user of an Implementation.
User Agent	The Component interfacing the user with an AIF through the Controller
Version	A revision or extension of a Standard or of one of its elements.

Annex 2 - Notices and Disclaimers Concerning MPAI Standards (Informative)

The notices and legal disclaimers given below shall be borne in mind when downloading and using approved MPAI Standards.

In the following, "Standard" means the collection of four MPAI-approved and published documents: "Technical Specification", "Reference Software" and "Conformance Testing" and, where applicable, "Performance Testing".

Life cycle of MPAI Standards

MPAI Standards are developed in accordance with the MPAI Statutes. An MPAI Standard may only be developed when a Framework Licence has been adopted. MPAI Standards are developed by especially established MPAI Development Committees who operate on the basis of consensus, as specified in Annex 1 of the MPAI Statutes. While the MPAI General Assembly and the Board of Directors administer the process of the said Annex 1, MPAI does not independently evaluate, test, or verify the accuracy of any of the information or the suitability of any of the technology choices made in its Standards.

MPAI Standards may be modified at any time by corrigenda or new editions. A new edition, however, may not necessarily replace an existing MPAI standard. Visit the web page to determine the status of any given published MPAI Standard.

Comments on MPAI Standards are welcome from any interested parties, whether MPAI members or not. Comments shall mandatorily include the name and the version of the MPAI Standard and, if applicable, the specific page or line the comment applies to. Comments should be sent to the MPAI Secretariat. Comments will be reviewed by the appropriate committee for their technical relevance. However, MPAI does not provide interpretation, consulting information, or advice on MPAI Standards. Interested parties are invited to join MPAI so that they can attend the relevant Development Committees.

Coverage and Applicability of MPAI Standards

MPAI makes no warranties or representations of any kind concerning its Standards, and expressly disclaims all warranties, expressed or implied, concerning any of its Standards, including but not limited to the warranties of merchantability, fitness for a particular purpose, non-infringement etc. MPAI Standards are supplied "AS IS".

The existence of an MPAI Standard does not imply that there are no other ways to produce and distribute products and services in the scope of the Standard. Technical progress may render the technologies included in the MPAI Standard obsolete by the time the Standard is used, especially in a field as dynamic as AI. Therefore, those looking for standards in the Data Compression by Artificial Intelligence area should carefully assess the suitability of MPAI Standards for their needs.

IN NO EVENT SHALL MPAI BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF

THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

MPAI alerts users that practicing its Standards may infringe patents and other rights of third parties. Submitters of technologies to this standard have agreed to licence their Intellectual Property according to their respective Framework Licences.

Users of MPAI Standards should consider all applicable laws and regulations when using an MPAI Standard. The validity of Conformance Testing is strictly technical and refers to the correct implementation of the MPAI Standard. Moreover, positive Performance Assessment of an implementation applies exclusively in the context of the MPAI Governance and does not imply compliance with any regulatory requirements in the context of any jurisdiction. Therefore, it is the responsibility of the MPAI Standard implementer to observe or refer to the applicable regulatory requirements. By publishing an MPAI Standard, MPAI does not intend to promote actions that are not in compliance with applicable laws, and the Standard shall not be construed as doing so. In particular, users should evaluate MPAI Standards from the viewpoint of data privacy and data ownership in the context of their jurisdictions.

Implementers and users of MPAI Standards documents are responsible for determining and complying with all appropriate safety, security, environmental and health and all applicable laws and regulations.

Annex 3 – The Governance of the MPAI Ecosystem (Informative)

Level 1 Interoperability

With reference to *Figure 1*, MPAI issues and maintains a standard – called MPAI-AIF – whose components are:

1. An environment called AI Framework (AIF) running AI Workflows (AIW) composed of inter-connected AI Modules (AIM) exposing standard interfaces.
2. A distribution system of AIW and AIM Implementation called MPAI Store from which an AIF Implementation can download AIWs and AIMs.

Implementers' benefits	Upload to the MPAI Store and have globally distributed Implementations of - AIFs conforming to MPAI-AIF. - AIWs and AIMs performing proprietary functions executable in AIF.
Users' benefits	Rely on Implementations that have been tested for security.
MPAI Store's role	- Tests the Conformance of Implementations to MPAI-AIF. - Verifies Implementations' security, e.g., absence of malware. - Indicates unambiguously that Implementations are Level 1.

Level 2 Interoperability

In a Level 2 Implementation, the AIW must be an Implementation of an MPAI Use Case and the AIMs must conform with an MPAI Application Standard.

Implementers' benefits	Upload to the MPAI Store and have globally distributed Implementations of - AIFs conforming to MPAI-AIF. - AIWs and AIMs conforming to MPAI Application Standards.
Users' benefits	- Rely on Implementations of AIWs and AIMs whose Functions have been reviewed during standardisation. - Have a degree of Explainability of the AIW operation because the AIM Functions and the data Formats are known.
Market's benefits	- Open AIW and AIM markets foster competition leading to better products. - Competition of AIW and AIM Implementations fosters AI innovation.
MPAI Store's role	- Tests Conformance of Implementations with the relevant MPAI Standard. - Verifies Implementations' security. - Indicates unambiguously that Implementations are Level 2.

Level 3 Interoperability

MPAI does not generally set standards on how and with what data an AIM should be trained. This is an important differentiator that promotes competition leading to better solutions. However, the performance of an AIM is typically higher if the data used for training are in greater quantity and more in tune with the scope. Training data that have large variety and cover the spectrum of all cases of interest in breadth and depth typically lead to Implementations of higher "quality".

For Level 3, MPAI normatively specifies the process, the tools and the data or the characteristics of the data to be used to Assess the Grade of Performance of an AIM or an AIW.

Implementers' benefits	May claim their Implementations have passed Performance Assessment.
Users' benefits	Get assurance that the Implementation being used performs correctly, e.g., it has been properly trained.
Market's benefits	Implementations' Performance Grades stimulate the development of more Performing AIM and AIW Implementations.

MPAI Store's role	-	Verifies the Implementations' security
	-	Indicates unambiguously that Implementations are Level 3.

The MPAI ecosystem

The following *Figure 2* is a high-level description of the MPAI ecosystem operation applicable to fully conforming MPAI implementations:

1. MPAI establishes and controls the not-for-profit MPAI Store (step 1).
2. MPAI appoints Performance Assessors (step 2).
3. MPAI publishes Standards (step 3).
4. Implementers submit Implementations to Performance Assessors (step 4).
5. If the Implementation Performance is acceptable, Performance Assessors inform Implementers (step 5a) and MPAI Store (step 5b).
6. Implementers submit Implementations to the MPAI Store (step 6); The Store Tests Conformance and security of the Implementation.
7. Users download Implementations (step 7).

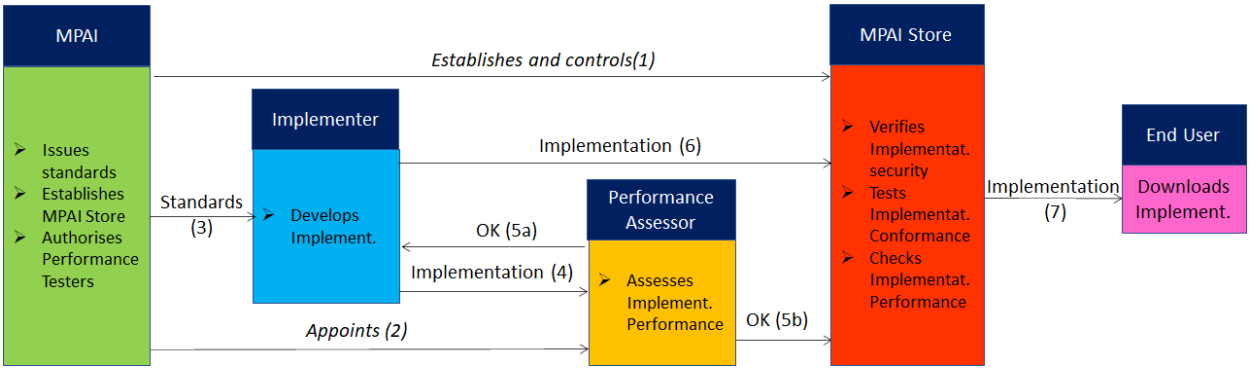

Figure 2 – The MPAI ecosystem operation