Feature Specification for

SiESoCom

Events Inference Pipeline 'EIP'

1.0

Revision History

Version	Who	When	Content	Comment
0.1	EP	May 10	Draft	Creation
0.2	EP	May 17	Draft	Multi input for AFE, first draft for review
0.3	EP	May 17	Draft	Add another AFE dataSchema 5.1.4
0.4	EP	June 18	Draft	Remove APIs not needed, update of app modeling, config file management details, cosmetics
0.5	EP	June 18	Draft	Cosmetics renaming of config parameter names
0.6	EP	June 18	Draft	Hidden plugin config parameter
0.7	FA	June 29	Draft	Added details of Python SDK Interfaces
1.0	EP	July 5 2021	baseline	Baseline

Table of Content

Contents

1 Context & Solution Topology

This document outlines the architecture and system requirements for the SiESoCom acoustic events inference pipeline. Other surrounding tiers of the application are also mentioned in this document for context but are specified in dedicated documents.

1.1 Event Inference Pipeline Topology

In the interest of system scaling and flexibility of operations, we split the inference pipeline into 2 containerized nodes: (i) Acoustic Feature Extractor (AFE) and (ii) Acoustic Event Detector (AED).

In addition, the actual AI logic, in Python, is separated from the container management logic. Ref to section below for details about this split.

The inter-container message bus technology is MQTT, which is also deployed as a containerized node.

1.2 Scale

A SiESoCom setup can include many active acoustic and/or time series sensors, so, in order to distribute the application load, the AFE shall be designed to be instantiated multiple times on the same setup.

Similarly, it may be necessary to execute multiple event detector occurrences in parallel to cover all the events the algorithms are trained to detect/classify. Also, it may be beneficial to concurrently run multiple AEDs configured with different trained models. It shall therefore be possible to instantiate multiple AEDs.

1.3 Data schema & Data schema repo

The various nodes of the inference pipeline produce outputs of different nature e.g. raw acoustic, time series of scalars, vectors etc... In order to avoid the hardcoding of data structures into the pipeline nodes code and optimize the data bus bandwidth, we define the encoding of such data structures using a standard data schema encoding technology: Apache Avro. We then serialize such data structure before transmission on the message bus.

Each data endpoint description/modeling shall therefore include a data schema description.

Notes:

- All the actual data schemas are available, using a unique name on a M³ data schema repository via a RESTful service. See section below for details of the concerned M³ services. Additional data schema can be designed as required.
- The introduction of data schema is planned for the second implementation of the SiESoCom application.

1.4 Time Series and Raw Acoustic Files Archivers

While the pipeline data are streamed over the MQTT message bus it shall also be possible to archive either the raw data (from the sensors) or the feature data (from the AFE instances) to local and/or central data bases. This archiving capabilities are optional and served by dedicated application tiers ('Time Series Archiver' and 'Raw Acoustic Filer). These components are mentioned here for context, but specified in dedicated documents.

1.5 Controller & Event Archiver

This component handles the dynamic creation of message bus accounts for the various tiers and the archiving of the system events.
This component is mentioned here for context, but specified in a dedicated document.

1.6 Topology views

From the discussions on points above, the high-level system topology is defined as follow:

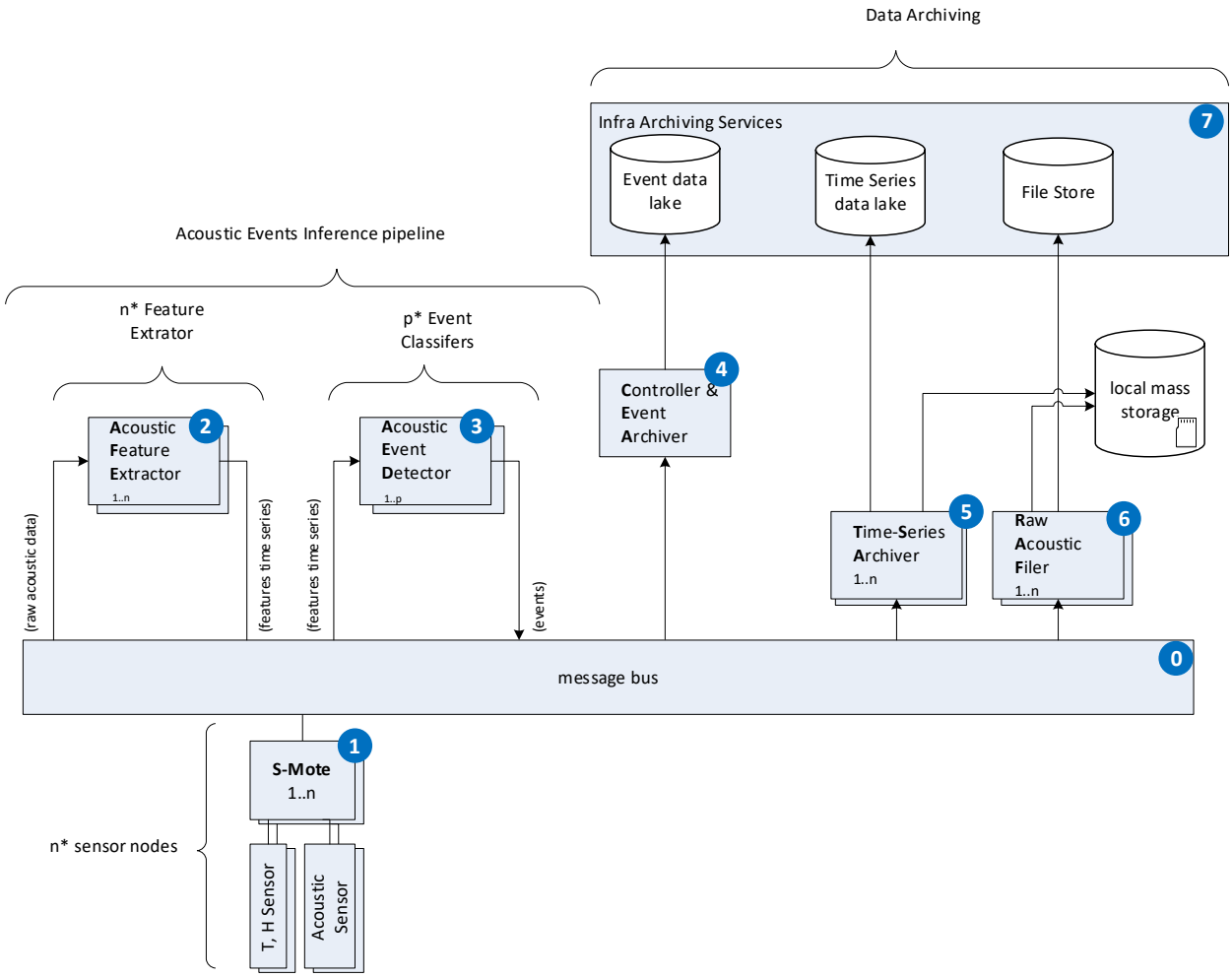

Fig: SiESoCom application topology – all tiers –

The following drawing focuses on the event inference pipeline, in an instance-level view. Notice how an AFE can ingest data from multiple sources and how an AED ingests features from multiple AFEs.

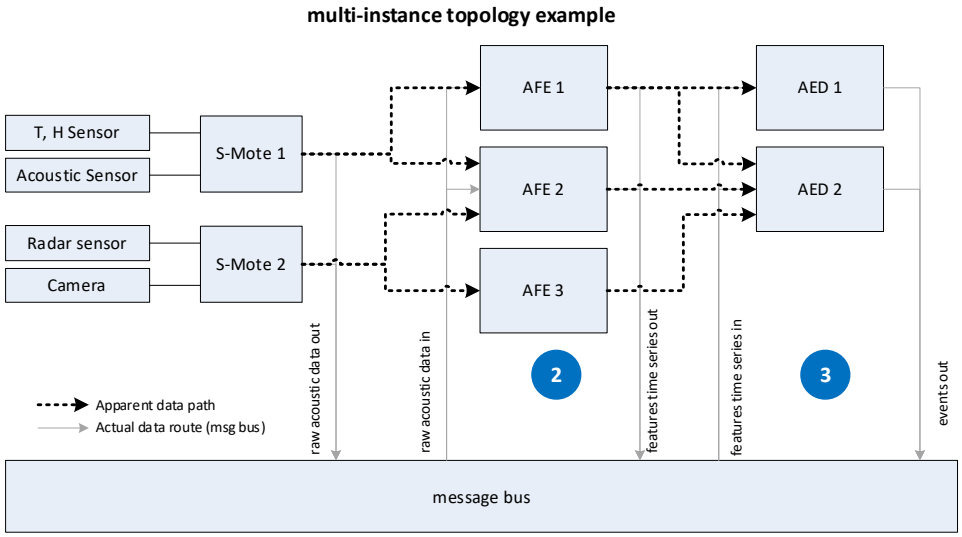

Fig: Instance view - Inference Pipeline only -

This document focuses on the specifications of the inference stage, i.e. modules (2) an (3) on the above diagram.

2 M³ Python Application

2.1 Software Architecture

Since the AI logic is typically implemented in Python, we are introducing a M³ containerized application that can accept a managed a python plugin as an extension of a 'base' application. AFE and AED tiers are implemented using this topology: A python plugin + config file shall be delivered by a data scientist and deployed on a common python base-application, part of M³ infrastructure.

This plugin is commissioned as a compressed python package stored in M³ file store. It is built assembled using a provided SDK.

The SDK contains a packaging tool to package the python plugin. The 'base application' shall provide major ML libraries as part of its runtime image. In addition to the base application built-in major ML libraries, other additional dependencies needed by the python plugin shall be resolved by the plugin developer at build time.

The additional dependencies shall be made part of the python plugin package using the packaging tool. No additional dependency shall be downloaded from

any python public repositories by the plugin manager at run-time. A complete python plugin package can be in one of the two forms as mentioned below:

1. ".wheel" file as produced by pip. In this case the plugin package contains only the plugin and there is no additional dependency.
2. ".tar.bz2" file as collection of multiple .wheel files. This shall be the case when there are additional dependencies added by the plugin developer. Every dependency is packaged in its own .wheel file and plugin is also packaged in its own .wheel file. These .wheel files shall be deployed by the plugin manager at runtime.

The plugin manager (packaging runtime component) shall install the plugin and its dependencies in a virtual environment, so it can be removed or upgraded easily. It is important that the plugin package is created using the provided packaging tool so that the plugin can be loaded by the plugin manager in the base application. A wheel file created without the packaging tool is not guaranteed to be loaded by the plugin manager at runtime.

Notices:

- For now, we assume the python plugin is a simple file. In future incarnations, we shall include further security for this package e.g. integrity check and/or encryption.
- The plugin package shall include a mandatory "m3.plugin.cfg" config file that defined (i) the supported input endpoints and (ii) the output endpoints of the plugin. The structure of this file is detailed in sections below and below. This file shall be located at the root of the plugin package.

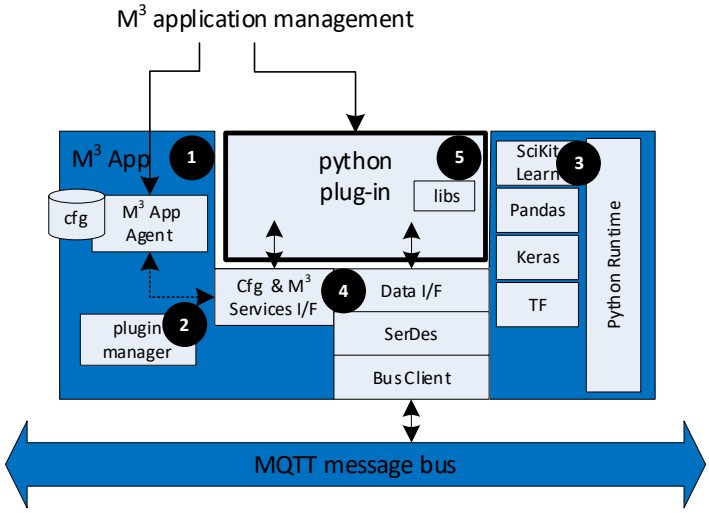

Fig: M3 app for python plugin

The actual M³ application(1) embeds a plugin manager(2), an AI-centric python runtime and libraries(3) and service & data interface services(4). A SDK and reference implementation including all these components shall be used to test and package the python plugin(5).

At application initialization time, the plugin manager:

- downloads and deploys the python plugin.
- locate the m3.plugin.cfg file, opens it and reports its json content into the plugin_config config parameter. Optionally, the plugin manager can run consistency checks on this file.
- Initialize the plugin and its Data I/O source & sinks
- Start the Plugin. Once the plugin is started, it shall start to receive input data in its registered data read callback.

2.2 Python plugin Interface Specification

The role of this interface (4) is to:

(i) Asynchronously signals the python plugin with the M³ application/plugin configuration changes and receives their commit confirmation.
(ii) Provide data I/O to the application message bus. These I/O are serialized /deserialized by the SerDes layer.
(iii) Provide access to M³ system services such as RESTFul API calls

Notice: This layer abstracts any networking or configuration details that are not necessary to the plugin logic.

Python plugin can be developed using the Python SDK (a.k.a I3S SDK). For the SDK, a plugin is considered as processing unit. This processor (or processing unit) is the primary component which represents the python plugin implemented by the developer.

In order to create a Plugin which can be deployed by the M³ Base Application plugin manager as part of the Inference Pipeline, the developers need to extend an I3SProcessor Interface available in the Python SDK to create the plugins. The I3SProcessor interface shall give access to the configuration of the plugin. In addition to the configuration, the I3SProcessor shall also provide access to the input data source & output data sink buffers to the plugin. The plugin does not explicitly have to connect or subscribe to its Data I/O source & sinks: Connection establishment and topics subscriptions shall be handled by the I3SProcessor interface for the plugin. To create a plugin, the plugin developer must provide the implementation of following APIs of I3SProcessor

1. **bool I3SProcessor::initialize_impl**
   Plugins can perform their plugin specific initialization in this method. Plugins will be provided with its configuration before its initialization which it can use for initialization. The I3SProcessor will initialize its data I/O channels after this initialization. In case of any failure during the complete I3SProcessor::Initialization, I3SProcessor::deinitialize_impl will be called where the application can perform any necessary cleanup.
2. **bool I3SProcessor::deinitialize_impl**
   Plugins can perform their cleanup in this method. This method shall be called by the I3SProcessor Interface while performing any cleanup, I.e

when the Plugin is stopping as part of close / stop operation or in case when plugin initialization or start is failed.

3. **I3SProcessor::app_conf_update_signal(MLConfiguration conf)**
   Plugins shall receive their plugin specific configuration in this API in the form of JSON. This API shall be called by the I3SProcessor interface before initialize_impl so that plugins can perform their initialization based on their configuration. Additionally plugins can also receive their updated configuration asynchronously during M3 Configuration Update signal from M3 Platform. I3SProcessor shall not keep any copy of plugin's configuration so it is the responsibility of plugins to keep a copy of their configuration received in this API.
4. **MLConfiguration I3SProcessor::get_app_config_impl()**
   I3SProcessor Interface can get the currently used plugin's configuration from the plugin using this API. In its implementation, the plugin must return its current configuration. This configuration shall be used by I3SProcessor for reporting purposes. Contrary to M3 Application Agent APIs, plugins which are developed using this Python SDK and deployed on the Base Applications, plugins do not need to explicitly commit their configuration to the M3 Platform. This operation shall be handled by the Base Application for plugins. The base application shall use this API of I3SProcessor to retrieve plugins currently applied configuration for this purpose of committing to the M3 Platform.
5. **I3SProcessor::register_source_read_callbacks(_source_ep_id)**
   Plugins must implement this API to register callback for receiving the input data. I3SProcessor interface shall call this API as part of its initialization. Once I3SProcessor is started, any input data received shall be delivered to the plugin using the registered callback. For plugin, the registered callback shall essentially becomes its primary processing function. _source_ep_id is optional for plugin to provide in case if it has only one data input. In such case a _source_ep_id shall be assigned by the I3SProcessor implicitly. Plugins can choose to pass empty string in this case. The data delivered to the plugins for MQTT protocol shall be a dictionary with following key-value pairs
   1. topic
   2. payload
   3. tuple
6. **I3SProcessor::write_data(_sink_ep_id)**
   In order to send data to output sink, plugins shall call this API. Similar to read operation, endpoint id is optional in case of single data output and plugins can choose to pass empty string. Plugins are expected to call this API in their data read callback function which is their primary processing function. For MQTT protocol, plugins shall pass a dictionary with following key-value pairs
   1. payload

NOTICE:

1. For plugin developers using the Python SDK with plugins deployed on the Base Application, it is not mandatory to commit their configurations to M3 Platform explicitly. This operation shall be handled by the M3 Base Application.
2. For details on the Python SDK, please consult the I3S SDK User Guide
3. Plugin developers using the Python SDK shall not be provided access to M3 Agent SDK and they shall not need to use its interfaces.

3 Acoustic Feature Extractor

The Acoustic Feature Extractor (AFE) shall be implemented as a python plugin for a M³ Python application, as described in section above

3.1 Application Modeling elements

3.1.1 Base Application config parameters

The AFE base application shall implement the following configurations as M³ managed config parameters. The config parameters that shall be managed by the plugin shall be mapped into the "Application/Plugin" configuration group.

- **"State"**: (i) Not Stated, (ii) Initializing and (iii) Ready.
- **"Additional state information"** can be provided optionally e.g. 'Loading Configuration' during the Initialization state.
- **"Broker host"**: A group of subscribed config parameters defining the message bus details: IP address, port, security mode.
- **"Application credentials"**: A group of config parameters for (Broker username, password) and TLS cred (CAcert, ClientCert, Key and expiry date)
- **States of the other application tiers:** Subscribed configs to controller state and message broker state at least, maybe more.
- **"Plugin package"**: A M³ file store URL pointing to the compressed Python Package that implements the AFE plugin.
- **"Plugin config"**: A json description of the configurations, input types and output descriptions of the AFE plugin. This json is prepared by the producer of the AFE plugin (see details about this structure on section 3.1.3 below). It contains:
  - Package version
  - Accepted Input descriptions i.e. acoustic stream endpoints
  - Output description i.e. a list of extracted features: For each output features, a full description of its configs and its data schema.
- **"Input EP"**: A **backend only** input config parameter that represents the sources of the raw data to ingest. The UI of the app tier will propose the user with a drop down of all eligible endpoints for the operator to select. Once selected, the M³ backend logic shall retrieve the full configuration & details of the selected end points (section 3.1.4belowbelow).

  Notice: The presentation of the list of eligible raw source endpoints requires the creation of a custom supported value list.

This list shall include all the acoustic sensor and raw time series sensors from the devices of the current setup. It's a multiple-selection scenario since more than one raw data source can be selected.

3.1.2 Plugin config parameters

- **"Input EP details"**: A system **application/plugin** config parameter (not shown to the portal user) set by the backend logic following the selection of the input endpoint via the "Input EP" config parameter. The content of this config is detailed in section below .
- **"Instance Info"**: A system **application/plugin** config parameter (not shown to the portal user) set by the base application with the AFE instance name and active setup Name. These information are required by the plugin to build it's publish topics.
- State: outgoing, read-only (Initializing, Ready)

3.1.3 AFE plugin configuration json

The purpose of this json structure is to document the static configuration of the AFE plugin. This structure is produced by the plugin developer in the m3.plugin.cfg file.

In the following example we are illustrating:

The AFE can ingest:

(i) acoustic frames of 8192 bytes
(ii) Scalar Temperature time series
(iii) Scalar Humidity time series

The AFE outputs 2 feature time series:

(i) A full spectrum across 20 freq band
(ii) A scalar profile for a given freq band

This example shall be used as a reference for the actual configuration structure/file:

```
{
  "packageVersion": "AFE 1.0",
  "dataIn": [
  {
      "endPointClass": {
        "type": "IDMT-microphone",
        "dataSchemaName": "SiESoComAcousticFrame.8192.noheader"
      }
    },
    {
      "endPointClass": {
```

```
          "type": "Temperature",
          "dataSchemaName": "ScalarTimeSeries"
        }
      },
      {
        "endPointClass": {
          "type": "Humidity",
          "dataSchemaName": "ScalarTimeSeries"
        }
      }
    ],
    "dataOut": [
      {
        "endPointClass": {
          "type": "acoustic-spectrum",
          "dataSchemaName": "VectorTimeSeries",
          "topicPattern": "@setupName/application/AFE/@instanceName/MFCC-out"
        },
        "endPoint": {
          "name": "MFCC-out",
          "filterBanks": 20,
          "windowSize": 512,
          "lowerFreq": 5,
          "higherFreq": 10000,
          "readingsPerFrame": 512
        }
      },
      {
        "endPointClass": {
          "type": "acoustic-spectrum-line",
          "DataSchemaName": "ScalarTimeSeries",
          "topicPattern": "@setupName/application/AFE/@instanceName/MFCC-Total-Energy-15"
        },
        "endPoint": {
          "name": "MFCC-Total-Energy-15",
          "filterBanks": 20,
          "windowSize": 512,
          "lowerFreq": 5,
          "higherFreq": 10000,
          "readingsPerFrame": 512
        }
      }
    ]
  }
```

Notices:

In this structure, only the class of 'supported' input endpoints are described. The actual instance(s) of input end points are set at the time the pipeline is routed by the user. The details of the input endpoint instances are stored into the "Input EPs" config parameter

This example shall be used to generate the actual AFE config json. While the details on the endPoints sub-structure are expected to be different, the class information shall match the actual value for the SiESoCom project.

3.1.4 AFE Inputs EP and Details

When the operator edits the AFE "Input EP", the portal assembles a list of eligible raw data endpoints matching the input type and dataSchemaName, as specified in the "dataIn" section of the plugin config json.

This list shall be presented to the user as a fully qualified list including the host name, sensor name and S/N e.g.

```
CSP-B827EB24A797.Microphone.001
CSP-B827EB24A797.Temperature.002
CSP-B827EB24A797.Humidity.003
CSP-B827EB2427B2.Microphone.001
CSP-B827EB2427B2.Temperature.002
CSP-B827EB2427B2.Humidity.003
```

After the operator has selected and applied the actual inputs of the AFE instance (one or many), the base application shall call a BE service to retrieve the full configuration & details of the selected endpoints, then pass this config to the plugin and report these details as a json structure, into the AFE "Input EP details" config parameter, per the following template/example (2 acoustic sources and 1 temperature source in this example):

```
{
  "dataIn": [
    {
      "endPointClass": {
        "type": "IDMT-microphone",
        "dataSchemaName": "SiESoComAcousticFrame.8192.noheader"
      },
      "endPoint": {
        "name": "Microphone",
        "HostSN": "CSP-B827EB24A797",
        "SN": "001",
        "State": "on",
        "Unit": "db",
        "samplingFrequency": 16,
        "sampleSize": 24,
        "channelCount": 2,
        "frameSize": 8192,
        "gain": 1,
        "topic": "SiESoComSetup_1/device/acoustic/CSP-B827EB24A797/001"
      }
    },
    {
      "endPointClass": {
        "type": "IDMT-microphone",
        "dataSchemaName": "SiESoComAcousticFrame.4096.noheader"
      },
      "endPoint": {
        "name": "Microphone",
        "HostSN": "CSP-B827EB2427B2",
        "SN": "001",
        "State": "on",
        "Unit": "db",
        "samplingFrequency": 48,
        "sampleSize": 16,
        "channelCount": 1,
        "frameSize": 4096,
        "gain": 0.5,
        "topic": "SiESoComSetup_1/device/acoustic/CSP-B827EB24A797/001"
```

```
      }
    },
    {
      "endPointClass": {
        "type": "Temperature",
        "dataSchemaName": "ScalarTimeSeries"
      },
      "endPoint": {
        "name": "Temperature",
        "HostSN": "CSP-B827EB2427B2",
        "SN": "002",
        "State": "on",
        "Unit": "C",
        "samplingFrequency": 2,
        "topic": "SiESoComSetup_1/device/readings/CSP-B827EB2427B2/001"
      }
    }
  ]
}
```

The AFE plugin shall take care of the incoming data rate adaptation.

Notice: In this structure, the "endPointClass" sub-structure is acquired from the "dataIn" structure of the AFE config file, which implement the 'supported' endpoints.

3.1.4.1 AFE in subscription topic

Upon receiving the dataIn configuration message, the plugin shall subscribe to the specified topics to get the actual stream of data.

3.1.5 AFE Instance Info

This plugin config parameter encodes the instance name and setupName of the current AFE instance. The AFE plugin need these data to expand its publish topics.

This information is encoded as a json, per the following description:

```
{
  "instanceName":"SiSoCom-AFE-12.2",
  "setupName":"SiESoComSetup_1"
}
```

At deployment time, the base application shall value the "instanceName" with the user-defined, unique name set for the newly created instance.

At the time the instance is attached to a setup, the base application shall value the "setupName" with the unique name of the setup and signal the plugin so it can accept this information

3.1.6 AFE outputs

AFE plugin outputs time series of scalar values and/or time series of vectors.

The AFE plugin shall structure the payload per the schema indicated in the AFE plugin config file and publish on the message bus on the topics documented in the "dataOut" section on the AFE plugin config file.

The application SERDES layer ensures the serialization.

Templates of time series of scalars and time series of vectors are documented below in section 3.1.6.1 and 3.1.6.2

3.1.6.1 Array of timetagged scalars

Example/template of output message (before serialization): array of timetagged scalars

```
{
  "endPointName": "MFCC-Total-Energy-17",
  "Payload": [
    {
      "Date": 1619167929060,
      "Value": 5.008301
    },
    {
      "Date": 1619167929560,
      "Value": 4.99762
    },
    {
      "Date": 1619167933574,
      "Value": 5.008301
    }
  ]
}
```

Data schema name:"ScalarTimeSeries"

3.1.6.2 Array of timetagged vectors

Example/template of output message (before serialization): array of timetagged vectors

```
{
  "endPointName": "MFCC",
  "Payload": [
    {
      "Date": 1619167929060,
      "Value": [
        14.4,
        18.2,
        1
      ]
    },
    {
      "Date": 1619167930060,
      "Value": [
```

```
        -21.4,
        -23.2,
        0.01
      ]
    }
  ]
}
```

Data schema: "VectorTimeSeries"

3.1.6.3 AFE out publication topic

The plugin shall expand the actual publish topic from (i) the topic template described on the "dataOut/endPointClass" section of the plugin config file and (ii) from the "Instance Info" plugin config parameter:

In this example:

```
Pattern for acoustic spectrum:
       "@setupName/application/AFE/@instanceName/MFCC-out"
setupName:
       "SiESoComSetup_1"
instanceName:
       "SiSoCom-AFE-12.2"

Actual publish topic:
       SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/MFCC-out


Pattern for acoustic spectrum line:
       "@setupName/application/AFE/@instanceName/ MFCC-Total-Energy-15"
setupName:
       "SiESoComSetup_1"
instanceName:
       "SiSoCom-AFE-12.2"

Actual publish topic:
    SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/MFCC-Total-Energy-15
```

4 Acoustic Event Detector

The Acoustic Event Detector (AED) shall be implemented as a python plugin for a M^3 Python application, as describe in section above.

4.1 Application Modeling elements

4.1.1 Base Application config parameters

The AED base application shall implement the following configurations as M³ managed config parameters. The config parameters that shall be managed by the plugin shall be mapped into the "Application/Plugin" configuration group.

- **"State"**: (i) Not Stated, (ii) Initializing and (iii) Ready.
- **"Additional state information"** can be provided optionally e.g. 'Loading Configuration' during the Initialization state.
- **"Broker Host"**: A group of subscribed config parameters defining the message bus details: IP address, port, security mode.
- **"Application credentials"**: A group of config parameters for (Broker username,password) and TLS cred (CAcert, ClientCert, Key and expiry date)
- **States of the other application tiers**: Subscribed configs to controller state, message broker state and AFE aggregated-state at least, maybe more.
- **"Plugin Package"**: A M³ file store URL pointing to the compressed Python Package that implements the AED plugin.
- **"Plugin Config"**: A json description of the configurations, input types and output descriptions of the AED plugin. This json is prepared by the producer of the AED plugin (see details about this structure on section 4.1.3 below). It contains:
  - Package version
  - Accepted Inputs description i.e. features
  - Output description i.e. a list of classified events.
- **"Input EP"**: A **backend only** input config parameter that represents the sources of the feature data to ingest. The UI of the app tier will propose the user with a drop down of all eligible endpoints for the operator to select. Once selected, the M³ backend logic shall retrieve the full configuration & details of the selected end points (section below).

  Notice: The presentation of the list of eligible acoustic feature source endpoints requires the creation of a custom supported value list. This list shall include all the features from all the AFE deployed on the current setup. It's a multiple-selection scenario since more than one feature EP can be selected.

4.1.2 Plugin config parameters

- **"Input EP details"**: A system **application/plugin** config parameter (not shown to the portal user) set by the backend logic following the selection of the input endpoint via the "Input EP" config parameter. The content of this config is detailed in section below.
- **"Instance Info"**: A system **application/plugin** config parameter (not shown to the portal user) set by the base application with the AED

instance name and active setup Name. These information are required by the plugin to build it's publish topics.

- **"Pipeline Data"**: A system **application/plugin** config parameter (not shown to the portal user) that represents pipeline-level information required for registering event occurrence.

4.1.3 AED plugin configuration json

The purpose of this json structure is to document the static configuration of the AED plugin. This structure is produced by the plugin developer in the m3.plugin.cfg file.

In the following example we are illustrating:

The AED can ingest

(i) acoustic-spectrum-line
(ii) correlation-line

The AED publishes 3 classified events

(i) Event 12: turbo compressor standy
(ii) Event 13: turbo compressor Ramp up
(iii) Event 14: turbo compressor const_WP_5600Hz

This example shall be used as a reference for the actual configuration structure/file:

```
{
  "PackageVersion": "AED 1.1",
  "dataIn": [
    {
      "EndPointClass": {
        "type": "acoustic-spectrum-line",
        "DataSchemaName": "ScalarTimeSeries"
      }
    },
    {
      "EndPointClass": {
        "type": "correlation-line",
        "dataSchemaName": "ScalarTimeSeries"
      }
    }
  ],
  "dataOut": [
    {
      "EndPointClass": {
        "type": "Event-occurence",
        "dataSchemaName": "EventOccurence",
        "topicPattern": "@setupName/application/AED/@instanceName/Event-occurence"
      },
      "endPoint": {
        "name": "Turbo-Events-out",
        "classifiedEvents": [
          "(12,turbo compressor standy)",
          "(13,turbo compressor Ramp up)",
```

```
            "(14,turbo compressor const_WP_5600Hz)"
          ]
        }
      }
    ]
  }
```

Notices:

In this structure, only the class of 'supported' input end points are described. The actual instance(s) of input end points are set at the time the pipeline is routed by the user. The details of the input endpoint instances are stored into the "Input EPs" config parameter

This example shall be used to generate the actual AED config json.

4.1.4 AED Inputs EP and Details

When the operator edits the AED "Input EP", the portal assembles a list of eligible feature endpoints matching the input type and data schema, as specified in the "dataIn" section of the plugin config json.

This list shall be presented to the user as a fully qualified list including the application/container name and feature name e.g.

```
"AFE-11.32.MFCC Total Energy (17)"
"AFE-11.32.MFCC Crest (6)"
"AFE-11.32.Autocorrelation (7)"
"AFE-11.32.Autocorrelation (3)"
"AFE-12.65.MFCC Total Energy (19)"
"AFE-12.65.MFCC Crest (4)"
```

After the operator has selected and applied the actual inputs of the AED instance (one or many), the base application shall call a BE service to retrieve the full configuration & details of the selected endpoints, then pass this config to the plugin and report these details as a json structure, into the AED "Input EP details" config parameter, per the following template/example (4 scalar time-series features in this example):

Input description example

```
{
  "dataIn": [
    {
      "endPointClass": {
        "type": "acoustic-spectrum-line",
        "dataSchemaName": "ScalarTimeSeries"
      },
      "endPoint": {
        "name": "MFCC-Total-Energy-17",
        "HostName": " AFE-11.32",
        "filterBanks": 20,
        "windowSize": 512,
        "lowerFreq": 5,
        "higherFreq": 10000,
        "readingsPerMessage": 512,
        "topic": "SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/MFCC-Total-Energy-17"
```

```
        }
      },
      {
        "endPointClass": {
          "type": "acoustic-spectrum-line",
          "dataSchemaName": "ScalarTimeSeries"
        },
        "endPoint": {
          "name": "MFCC-Crest-6",
          "HostName": " AFE-11.32",
          "filterBanks": 20,
          "windowSize": 512,
          "lowerFreq": 5,
          "higherFreq": 10000,
          "readingsPerMessage": 512,
          "topic": "SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/MFCC-Crest-6"
        }
      },
      {
        "EndPointClass": {
          "type": "correlation-line",
          "dataSchemaName": "ScalarTimeSeries"
        },
        "endPoint": {
          "name": "Autocorrelation-7",
          "HostName": " AFE-11.32",
          "filterBanks": 20,
          "windowSize": 512,
          "lowerFreq": 5,
          "higherFreq": 10000,
          "readingsPerFrame": 512,
          "topic": "SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/Autocorrelation-7"
        }
      },
      {
        "endPointClass": {
          "type": "correlation-line",
          "dataSchemaName": "ScalarTimeSeries"
        },
        "endPoint": {
          "name": "Autocorrelation-3",
          "HostName": "AFE-11.32",
          "filterBanks": 20,
          "windowSize": 512,
          "lowerFreq": 5,
          "higherFreq": 10000,
          "readingsPerFrame": 512,
          "topic": "SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/Autocorrelation-3"
        }
      }
    ]
  }
```


Notices:

In this structure, the "endPointClass" structure is acquired from the "dataIn" structure of the AED config file, which implement the 'supported' endpoints.

The "topic" information in the "endPoint" structure has to be expanded by the backend service, from the "topicTemplate" information in the "endPointClass" structure.

4.1.4.1 AED in subscription topic

Upon receiving the dataIn configuration message, the plugin shall subscribe to the specified topics to get the actual stream of data.

4.1.5 Pipeline data for event occurrences

The following meta-data provide user and pipeline structure elements that are managed by M³ and the base application. They represents system-level information required for registering event occurrence:

accountId: account Id of the client reporting the event
tenantId: tenant Id of the client reporting the event
userId: user_id of the client reporting the event
targets: Id of machines under monitoring, linked to the setup (manually set for now)
detector: list of application instances (e.g. device S/N, featurizer, inference algorithm) in the inference pipeline
sources: list of acoustic sensor endpoints in the pipeline

These configurations are extracted by an M³ service (section below)and reported to the python plugin over the plugin/base application interface at the initialization time, per the following structure:

```
{
  "pipelineData": {
    "accountId": 188,
    "tenantId": 20,
    "userId": 33159,
    "targets": "Turbo Compressor 1-3",
    "detector": [
      {
        "host": "(edge core,8M07LG2)",
        "application": {
          "image": "(AFE 1.0, 1.4)",
          "model": "(Featurizer 123, model_02)"
        }
      },
      {
        "host": "(edge core,8M07LG2)",
        "application": {
          "image": "(AED 1.1, 3.2)",
          "model": "(Classifier 67, model_34)"
        }
      }
    ],
    "sources": [
      {
        "instance": "(device, CSP-B827EB24A797)",
        "endpoints": [
          "(sensors.sensor1.Serial Number, 001)"
        ]
      }
    ]
  }
}
```

Note: The 'model' field for the detector nodes refers to the reference of the plugin package i.e. a M³ file URL.

4.1.6 AED Instance Info

This plugin config parameter encodes the instance name and setupName of the current AED instance. The AFE plugin need these data to expand its publish topics.

This information is encoded as a json, per the following description:

```
{
  "instanceName":"SiSoCom-AED-2.2.1",
  "setupName":"SiESoComSetup_1"
}
```

At deployment time, the base application shall value the "instanceName" with the user-defined, unique name set for the newly created instance.

At the time the instance is attached to a setup, the base application shall value the "setupName" with the unique name of the setup and signal the plugin so it can accept this information

4.1.7 AED output

AED plugin outputs arrays of M^3 event occurences. These M^3 event occurences are defined by a variety of meta-data. AED shall set this meta-data to allow the creation, archiving and querying of the event occurences into/from M^3 data lake.

The AED plugin shall structure the payload per the data schema indicated in the AED plugin config file and publish on the message bus on the topics documented in the "dataOut" section on the AED plugin config file.

The application SERDES layer ensures the serialization.

Templates of event occurences are documented below in section 4.1.7.1

4.1.7.1 Timetagged event occurences

The following is an example/template of the json structure the AED shall produce to report an occurence of event.

Meta data

pipelineData: The pipeline context structure reported by the base application via the "Pipeline Data" config parameter. To be appended 'as is' to the event occurrence meta-data.
eventId: Id of event the AED is trained to infer
state: 'set' (mandatory and static)
detectionDate: date at which the event has been detected, in linux epoch msec.
eventDate: date at which the event actually occured (may be before the actual detection), in linux epoch msec.
eventDuration: duration in millisec of event - if applicable

triggeringValues: series of json (key, value) pairs that encodes the actual value or series of values that triggered the event occurrence – if applicable
contextualInformation: series of json (key, value) pairs that encodes client-defined contextual information about the event occurence. – if applicable
expiryDate: date at which the event expires – mandatory, in linux epoch msec
confidenceLevel: confidence value for the inference – if applicable, otherwise set to 1.0
weightFactor: relative importance of an event occurrence – Optional
files: A list of (file url encoding: name, Id and app-enduserid) from the cc_file table. Optional

Note about "eventId": This Id refers to M³ commissioned events, i.e. the events the algorithm are trained to classify. For example:

12: turbo compressor standy
13: turbo compressor Ramp up
14: turbo compressor const_WP_5600Hz
15: turbo compressor const_WP_5900Hz
16: turbo compressor Ramp down

The actual list of commissioned events can be extracted from M³ data base at any time. An M³ request service exist to retrieve this information.

Note about "files": The AED shall be able to upload small 'companion' data files for any inferred event occurence e.g. 2 sec of feature time series before and after the classified event. This 'raw' data can be used to further qualify/rate/inspect the inference model as part of a verification logic. See section above for details about this procedure

Example/template of output message (before serialization): an occurrence of M³ event 15


```
{
  "pipelineData": {
    "accountId": 188,
    "tenantId": 20,
    "userId": 33159,
    "targets": "Turbo Compressor 1-3",
    "detector": [
      {
        "host": "(edge core,8M07LG2)",
        "application": {
          "image": "(AFE 1.0, 1.4)",
          "model": "(Featurizer 123, model_02)"
        }
      },
      {
        "host": "(edge core,8M07LG2)",
        "application": {
          "image": "(AED 1.1, 3.2)",
          "model": "(Classifier 67, model_34)"
        }
      },
      {
        "host": "(device, CSP-B827EB24A797)",
        "application": {
          "image": "(SiSoCom Mote FW 1.0 MQTT/Ethernet, 4.1)",
          "model": "(Classifier 67, model_34)"
        }
      }
    ],
    "sources": [
      {
```

```
            "instance": "(device, CSP-B827EB24A797)",
            "endpoints": [
              "(sensors.sensor1.Serial Number, 001)"
            ]
          }
        ]
      },

      "eventId": 15,
      "state": "set",
      "detectionDate": 1591757489992,
      "eventDate": 1591757489192,
      "eventDuration": 1526,
      "triggeringValues": [
        {
          "trigger1": "value1"
        },
        {
          "trigger2": 31
        }
      ],
      "contextualInformation": [
        {
          "contextKey1": "contextValue1"
        },
        {
          "contextKey2": 212
        }
      ],
      "expiryDate": 1591759489992,
      "confidenceLevel": 0.8,
      "weightFactor": 1,
      "files": [
        "https://api.m3-solutions.io/file-
    services/cspfilestore/v1/619547/f1.csv?appenduserid=0&storage=s3",
        "https://api.m3-solutions.io/file-
    services/cspfilestore/v1/619460/f2.csv?appenduserid=0&storage=s3"
      ]
    }
```


Avro schema name:"EventOccurence"

4.1.7.2 AED out publication topic

The plugin shall expand the actual publish topic from (i) the topic template described on the "dataOut/endPointClass" section of the plugin config file and (ii) from the "Instance Info" plugin config parameter:

In this example:

Pattern for EventOccurence:
"@setupName/application/AED/@instanceName/Event-occurence"
setupName:
"SiESoComSetup_1"
instanceName:
"SiSoCom-AED-2.2.1"

Actual publish topic:

5 M³ Infra Requirements

5.1 Data Schema Repo and default data schemas

5.1.1 Acoustic Frame, no header

SiESoCom AFE inputs raw acoustic in frames without header. The number of bytes per frame depends on the configuration

The example below describes a template payload and its matching Avro model

Message / binary frame

```
22FF 21FF 9CFF 3E00 B700 0801 9C01 0202
0502 BE01 5B01 E400 5200 1B00 EBFF 9BFF
25FF 2CFF D3FE BFFE D1FE 8AFE 6EFE 2DFE
2FFE 66FE E3FE 1EFF 07FF 45FF 7EFF B9FF
4B00 ED00 6B01 C601 4901 7800 F3FF ….BDFF
```

Avro Schema / 1024 Binary frame - no header

```
{
  "type": "record",
  "name": "SiESoComAcousticFrame.1024.noheader",
  "namespace": "rc_data_schema_repo",
  "fields": [
    {
      "type": "fixed",
      "name": "AudioFrame",
      "size": 1024
    }
  ]
}
```

Avro Schema / 2048 Binary frame - no header

```
{
  "type": "record",
  "name": "SiESoComAcousticFrame.2048.noheader",
  "namespace": "rc_data_schema_repo",
  "fields": [
    {
      "type": "fixed",
      "name": "AudioFrame",
      "size": 2048
    }
  ]
}
```

Avro Schema / 4096 Binary frame - no header

```
{
  "type": "record",
  "name": "SiESoComAcousticFrame.4096.noheader",
  "namespace": "rc_data_schema_repo",
  "fields": [
```

```
            {
              "type": "fixed",
              "name": "AudioFrame",
              "size": 4096
            }
          ]
        }
```

Avro Schema / 8192 Binary frame - no header

```
        {
          "type": "record",
          "name": "SiESoComAcousticFrame.8192.noheader",
          "namespace": "rc_data_schema_repo",
          "fields": [
            {
              "type": "fixed",
              "name": "AudioFrame",
              "size": 8192
            }
          ]
        }
```

Schema repo mapping:

```
        rc_data_schema_repo.id=1
        rc_data_schema_repo.schema="avro"
        rc_data_schema_repo.name="SiESoComAcousticFrame.1024.noheader"
        rc_data_schema_repo.data_schema = "<schema json here above>"

        rc_data_schema_repo.id=2
        rc_data_schema_repo.schema="avro"
        rc_data_schema_repo.name="SiESoComAcousticFrame.2048.noheader"
        rc_data_schema_repo.data_schema = "<schema json here above>"

        rc_data_schema_repo.id=3
        rc_data_schema_repo.schema="avro"
        rc_data_schema_repo.name="SiESoComAcousticFrame.4196.noheader"
        rc_data_schema_repo.data_schema = "<schema json here above>"

        rc_data_schema_repo.id=4
        rc_data_schema_repo.schema="avro"
        rc_data_schema_repo.name="SiESoComAcousticFrame.8192.noheader"
        rc_data_schema_repo.data_schema = "<schema json here above>"
```

5.1.2 Array of timetagged scalars

SiESoCom AFE may output time series of scalar value (e.g. MFCC spectral line). In order to reduce transmission overhead on the message bus, we assume that these timetagged scalar values are published in arrays i.e. with more that one reading per message.

The example below describes a template payload and its matching Avro model

Message / array of timetagged scalars

```
        {
          "endPointName": "MFCC-Total-Energy-17",
          "Payload": [
            {
              "Date": 1619167929060,
              "Value": 5.008301
            },
            {
              "Date": 1619167929560,
```

```
            "Value": 4.99762
          },
          {
            "Date": 1619167933574,
            "Value": 5.008301
          }
        ]
      }
```

Avro schema / array of timetagged scalars

```
      {
        "name": "ScalarTimeSeries",
        "type": "record",
        "namespace": "rc_data_schema_repo",
        "fields": [
          {
            "name": "endPointName",
            "type": "string"
          },
          {
            "name": "Payload",
            "type": {
              "type": "array",
              "items": {
                "name": "Payload_record",
                "type": "record",
                "fields": [
                  {
                    "name": "Date",
                    "type": "long"
                  },
                  {
                    "name": "Value",
                    "type": "float"
                  }
                ]
              }
            }
          }
        ]
      }
```

Schema repo mapping:

```
      rc_data_schema_repo.id=5
      rc_data_schema_repo.schema="avro"
      rc_data_schema_repo.name="ScalarTimeSeries"
      rc_data_schema_repo.data_schema = "<schema json here above>"
```

5.1.3 Array of timetagged vectors

SiESoCom AFE may output time series of vector value (e.g. MFCC). In order to reduce transmission overhead on the message bus, we assume that these timetagged vector values are published in arrays i.e. with more that one reading per message.

The example below describes a template payload and its matching Avro model

Message / array of timetagged vectors

```
      {
        "endPointName": "MFCC",
        "Payload": [
          {
```

```
            "Date": 1619167929060,
            "Value": [
              14.4,
              18.2,
              1
            ]
          },
          {
            "Date": 1619167930060,
            "Value": [
              -21.4,
              -23.2,
              0.01
            ]
          }
        ]
      }
```

Avro schema / array of timetagged vectors

```
      {
        "name": "VectorTimeSeries",
        "type": "record",
        "namespace": "rc_data_schema_repo",
        "fields": [
          {
            "name": "endPointName",
            "type": "string"
          },
          {
            "name": "Payload",
            "type": {
              "type": "array",
              "items": {
                "name": "Payload_record",
                "type": "record",
                "fields": [
                  {
                    "name": "Date",
                    "type": "long"
                  },
                  {
                    "name": "Value",
                    "type": {
                      "type": "array",
                      "items": "float"
                    }
                  }
                ]
              }
            }
          }
        ]
      }
```

Schema repo mapping:

```
      rc_data_schema_repo.id=6
      rc_data_schema_repo.schema="avro"
      rc_data_schema_repo.name="VectorTimeSeries"
      rc_data_schema_repo.data_schema = "<schema json here above>"
```

5.1.4 Array of timetagged Combo Scalar Features

SiESoCom AFE may output time series of multiple scalar features values. In order to reduce transmission overhead on the message bus, we assume that these timetagged values are published in arrays i.e. with more that one reading per message.

The example below describes a template payload and its matching Avro model

Message / array of multiple combo scalar features

```
{
  "endPointName": "MFCC-Feature-Combo-1",
  "Payload": [
    {
      "Date": 1619167929060,
      "ComboValues": [
        {
          "Feature": "MFCC-Total-Energy-17",
          "Value": 5.008
        },
        {
          "Feature": "spectral-flatness",
          "Value": 0.083451
        }
      ]
    },
    {
      "Date": 1619167930060,
      "ComboValues": [
        {
          "Feature": "MFCC-Total-Energy-17",
          "Value": 6.001
        },
        {
          "Feature": "spectral-flatness",
          "Value": 0.095001
        }
      ]
    }
  ]
}
```

Avro schema / array of timetagged combo-feature

```
        {
          "name": "ComboFeatureTimeSeries",
          "type": "record",
          "namespace": "rc_data_schema_repo",
          "fields": [
            {
              "name": "endPointName",
              "type": "string"
            },
            {
              "name": "Payload",
              "type": {
                "type": "array",
                "items": {
                  "name": "Payload_record",
                  "type": "record",
                  "fields": [
                    {
                      "name": "Date",
                      "type": "long"
                    },
```

```
                {
                  "name": "ComboValues",
                  "type": {
                    "type": "array",
                    "items": [
                      {
                        "name": "ComboValues_record",
                        "type": "record",
                        "fields": [
                          {
                            "name": "Feature",
                            "type": "string"
                          },
                          {
                            "name": "Value",
                            "type": "float"
                          }
                        ]
                      },
                      {
                        "name": "ComboValues_record",
                        "type": "record",
                        "fields": [
                          {
                            "name": "Feature",
                            "type": "string"
                          },
                          {
                            "name": "Value",
                            "type": "float"
                          }
                        ]
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
      ]
    }
```

Schema repo mapping:

```
rc_data_schema_repo.id=7
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="ComboFeatureTimeSeries"
rc_data_schema_repo.data_schema = "<schema json here above>"
```

5.1.5 M³ event occurrences

SiESoCom AFE outputs M³ event occurrences .

The example below describes a template payload and its matching Avro model

Message / array of M3 event occurrences

```
{
  "pipelineData": {
    "accountId": 188,
```

```
    "tenantId": 20,
    "userId": 33159,
    "targets": "Turbo Compressor 1-3",
    "detector": [
      {
        "host": "(edge core,8M07LG2)",
        "application": {
          "image": "(AFE 1.0, 1.4)",
          "model": "(Featurizer 123, model_02)"
        }
      },
      {
        "host": "(edge core,8M07LG2)",
        "application": {
          "image": "(AED 1.1, 3.2)",
          "model": "(Classifier 67, model_34)"
        }
      },
      {
        "host": "(device, CSP-B827EB24A797)",
        "application": {
          "image": "(SiSoCom Mote FW 1.0 MQTT/Ethernet, 4.1)",
          "model": "(Classifier 67, model_34)"
        }
      }
    ],
    "sources": [
      {
        "instance": "(device, CSP-B827EB24A797)",
        "endpoints": [
          "(sensors.sensor1.Serial Number, 001)"
        ]
      }
    ]
  },
  "eventId": 15,
  "state": "set",
  "detectionDate": 1591757489992,
  "eventDate": 1591757489192,
  "eventDuration": 1526,
  "triggeringValues": [
    {
      "trigger1": "value1"
    },
    {
      "trigger2": 31
    }
  ],
  "contextualInformation": [
    {
      "contextKey1": "contextValue1"
    },
    {
      "contextKey2": 212
    }
  ],
  "expiryDate": 1591759489992,
  "confidenceLevel": 0.8,
  "weightFactor": 1,
  "files": [
    "https://api.m3-solutions.io/file-
services/cspfilestore/v1/619547/f1.csv?appenduserid=0&storage=s3",
    "https://api.m3-solutions.io/file-
services/cspfilestore/v1/619460/f2.csv?appenduserid=0&storage=s3"
  ]
}
```

Avro schema / array of M3 event occurences

```
{
  "name": "EventOccurence",
  "type": "record",
  "namespace": "rc_data_schema_repo",
  "fields": [
    {
      "name": "pipelineData",
      "type": {
        "name": "pipelineData",
        "type": "record",
        "fields": [
          {
            "name": "accountId",
            "type": "int"
          },
          {
            "name": "tenantId",
            "type": "int"
          },
          {
            "name": "userId",
            "type": "int"
          },
          {
            "name": "targets",
            "type": "string"
          },
          {
            "name": "detector",
            "type": {
              "type": "array",
              "items": {
                "name": "detector_record",
                "type": "record",
                "fields": [
                  {
                    "name": "host",
                    "type": "string"
                  },
                  {
                    "name": "application",
                    "type": {
                      "name": "application",
                      "type": "record",
                      "fields": [
                        {
                          "name": "image",
                          "type": "string"
                        },
                        {
                          "name": "model",
                          "type": "string"
                        }
                      ]
                    }
                  }
                ]
              }
            }
          },
          {
            "name": "sources",
            "type": {
              "type": "array",
```

```
            "items": {
              "name": "sources_record",
              "type": "record",
              "fields": [
                {
                  "name": "instance",
                  "type": "string"
                },
                {
                  "name": "endpoints",
                  "type": {
                    "type": "array",
                    "items": "string"
                  }
                }
              ]
            }
          }
        }
      ]
    }
  },
  {
    "name": "eventId",
    "type": "int"
  },
  {
    "name": "state",
    "type": "string"
  },
  {
    "name": "detectionDate",
    "type": "long"
  },
  {
    "name": "eventDate",
    "type": "long"
  },
  {
    "name": "eventDuration",
    "type": "int"
  },
  {
    "name": "triggeringValues",
    "type": {
      "type": "array",
      "items": {
        "name": "triggeringValues_record",
        "type": "record",
        "fields": [
          {
            "name": "trigger1",
            "type": "string"
          }
        ]
      }
    }
  },
  {
    "name": "contextualInformation",
    "type": {
      "type": "array",
      "items": {
        "name": "contextualInformation_record",
        "type": "record",
        "fields": [
          {
```

```
                    "name": "contextKey1",
                    "type": "string"
                  }
                ]
              }
            }
          },
          {
            "name": "expiryDate",
            "type": "long"
          },
          {
            "name": "confidenceLevel",
            "type": "float"
          },
          {
            "name": "weightFactor",
            "type": "int"
          },
          {
            "name": "files",
            "type": {
              "type": "array",
              "items": "string"
            }
          }
        ]
      }
```

Schema repo mapping:

```
rc_data_schema_repo.id=7
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="EventOccurence"
rc_data_schema_repo.data_schema = "<schema json here above>"
```

5.2 API specifications

This section documents the additional M³ API(s) required to support this specification.

5.2.1 Data schema services

create/read/update/delete services. Both internal and external data services

5.2.1.1 API call details

GET /data-schema-repo/v1/schema/{id}
Response:

```
{
      "id":"",
      "schema":"",
      "name":"",
```

```
            "dataSchema":""
      }


POST /data-schema-repo/v1/schema
Body:
      {
            "schema":"",
            "name":"",
            "dataSchema":""
      }
Response:
      {
            "id":"",
            "schema":"",
            "name":"",
            "dataSchema":""
      }


PUT /data-schema-repo/v1/schema/{id}
Body:
      {
            "schema":"",
            "name":"",
            "dataSchema":""
      }
Response:
      {
            "id":"",
            "schema":"",
            "name":"",
            "dataSchema":""
      }


DELETE /data-schema-repo/v1/schema/{id}
```

5.2.1.2 Initial content

```
rc_data_schema_repo.id=1
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="SiESoComAcousticFrame.1024.noheader"
rc_data_schema_repo.data_schema = "<schema json here above>"

rc_data_schema_repo.id=2
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="SiESoComAcousticFrame.2048.noheader"
rc_data_schema_repo.data_schema = "<schema json here above>"

rc_data_schema_repo.id=3
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="SiESoComAcousticFrame.4196.noheader"
rc_data_schema_repo.data_schema = "<schema json here above>"
```

```
rc_data_schema_repo.id=4
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="SiESoComAcousticFrame.8192.noheader"
rc_data_schema_repo.data_schema = "<schema json here above>"


rc_data_schema_repo.id=5
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="ScalarTimeSeries"
rc_data_schema_repo.data_schema = "<schema json here above>"

rc_data_schema_repo.id=6
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="VectorTimeSeries"
rc_data_schema_repo.data_schema = "<schema json here above>"

rc_data_schema_repo.id=7
rc_data_schema_repo.schema="avro"
rc_data_schema_repo.name="EventOccurence"
rc_data_schema_repo.data_schema = "<schema json here above>"
```

5.2.2 Get Pipeline Data

The role of this service is to populate the pipelineData structure for the AED 'Pipeline Data' config parameter, defined in section above for the current AED instance, in particular the 'detector' and the 'source' lists.

We define the 'detector' for a given event occurrence as the list made of the following instances: The AED instance that actually host the inference algorithm + all the AFE instances to which the AED is connected. For each instance, the structure shall include it's host and application unique Id. For the application unique Id, we include both the base application image and the plugin (refers to as 'model')

We define the 'source' for a given event occurrence as the list made of the sensor serial numbers connected to the AFEs of the detector. For each sensor id, the structure shall include it's host.

The drawings below illustrate the 'detectors' and 'sources' in 2 different scenario.

sources & detectors in multi-instance topology example#1

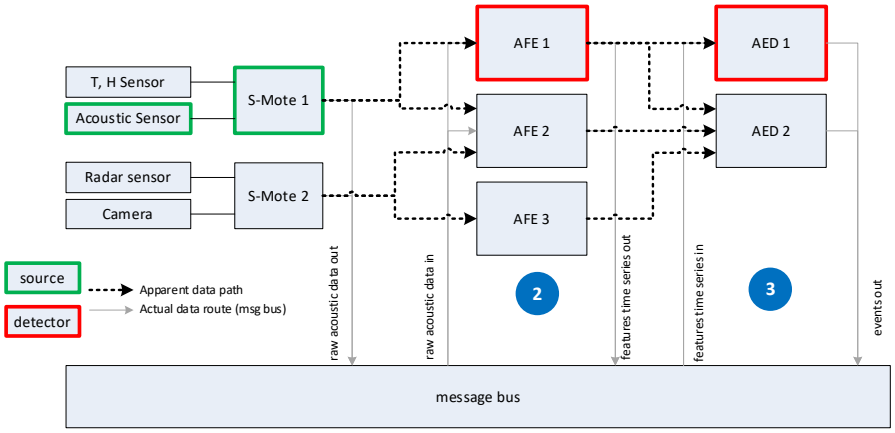

Fig:sources and detectors, example 1

sources & detectors in multi-instance topology example#2

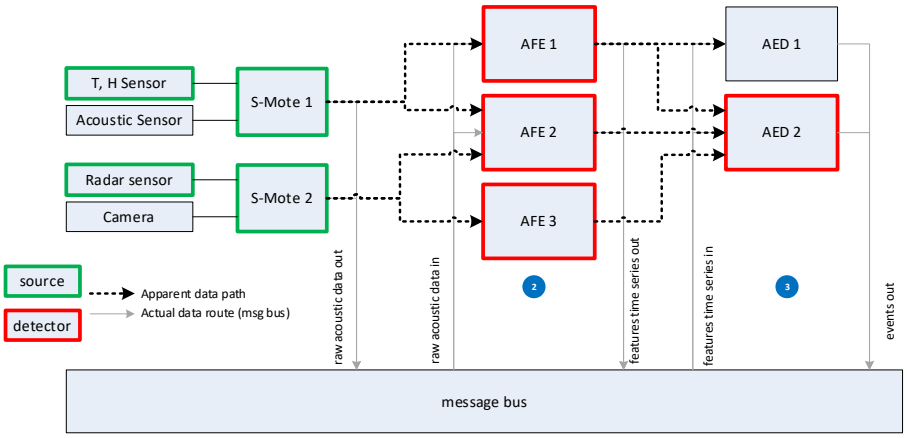

Fig:sources and detectors, example 2

```
GET /data-pipelines/v1/pipelines
Response Example (depends on app topology):
{
    "sources": [
        {
            "instance": "(device,CSP-B827EB8DFC74)",
            "endpoints": [
                "(sensors.sensor1.Serial Number, 001)",
                "(sensors.sensor2.Serial Number, 002)"
            ]
        }
    ],
    "detector": [
        {
```

```
            "host": "(server,srv3244)",
            "application": {
                "image": "(srv_detector V1.0,1.4)",
                "instance": "(server app, Detector_SETUP_2)",
                "model": "(Machine Learning Model,model_02)"
            }
        }
    ]
}
```

5.2.3 Get Data EndPoints Details

The role of this service is to populate the details of the user-selected endpoints into a json structure for the plugin

Example for AFE:

GET /data-endpoints/v1/setups/{setupId}?filters=["CSP-B827EB24A797.Microphone.001", "CSP-B827EB2427B2.Temperature.002"]

Example Response output depends on type of endpoint:

```
{
  "dataIn": [
    {
      "endPointClass": {
        "type": "IDMT-microphone",
        "dataSchemaName": "SiESoComAcousticFrame.8192.noheader"
      },
      "endPoint": {
        "name": "Microphone",
        "HostSN": "CSP-B827EB24A797",
        "SN": "001",
        "State": "on",
        "Unit": "db",
        "samplingFrequency": 16,
        "sampleSize": 24,
        "channelCount": 2,
        "frameSize": 8192,
        "gain": 1,
        "topic": "SiESoComSetup_1/device/acoustic/CSP-B827EB24A797/001"
      }
    },
    {
      "endPointClass": {
        "type": "Temperature",
        "dataSchemaName": "ScalarTimeSeries"
      },
      "endPoint": {
        "name": "Temperature",
        "HostSN": "CSP-B827EB2427B2",
        "SN": "002",
        "State": "on",
        "Unit": "C",
        "samplingFrequency": 2,
        "topic": "SiESoComSetup_1/device/readings/CSP-B827EB2427B2/001"
      }
    }
```

```
  ]
}
```

Example for AED:

GET /data-endpoints/v1/setups/{setupId}?Filters=["AFE-11.32.MFCC Total Energy (17)", "AFE-11.32.Autocorrelation (7)"]

Example Response output depends on type of EP:


```
{
  "dataIn": [
    {
      "endPointClass": {
        "type": "acoustic-spectrum-line",
        "dataSchemaName": "ScalarTimeSeries"
      },
      "endPoint": {
        "name": "MFCC-Total-Energy-17",
        "HostName": " AFE-11.32",
        "filterBanks": 20,
        "windowSize": 512,
        "lowerFreq": 5,
        "higherFreq": 10000,
        "readingsPerMessage": 512,
        "topic": "SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/MFCC-Total-Energy-17"
      }
    },
    {
      "EndPointClass": {
        "type": "correlation-line",
        "dataSchemaName": "ScalarTimeSeries"
      },
      "endPoint": {
        "name": "Autocorrelation-7",
        "HostName": " AFE-11.32",
        "filterBanks": 20,
        "windowSize": 512,
        "lowerFreq": 5,
        "higherFreq": 10000,
        "readingsPerFrame": 512,
        "topic": "SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/Autocorrelation-7"
      }
    },
    {
      "endPointClass": {
        "type": "correlation-line",
        "dataSchemaName": "ScalarTimeSeries"
      },
      "endPoint": {
        "name": "Autocorrelation-3",
        "HostName": "AFE-11.32",
        "filterBanks": 20,
        "windowSize": 512,
        "lowerFreq": 5,
        "higherFreq": 10000,
        "readingsPerFrame": 512,
        "topic": "SiESoComSetup_1/application/AFE/SiSoCom-AFE-12.2/Autocorrelation-3"
      }
    }
  ]
}
```

5.2.4 Get List of Event Occurrence outputs

The portal shall create a list of all the AED event occurrences outputs. This list shall include the topics and dataSchema and will be used by the CEA application to subscribe to and ingest the various occurrences of inferred events.

```
SELECT <topics for event occurrences>, <event occurrences details>
FROM <all AED in setup>
WHERE "EndPointClass.Type="Event-Occurrence";
```

```
Dynamic Query:
{
    "cId":"<topics for event occurrences>",
    "cVal":"<event occurrences details>",
    "table":"<all AED in setup>",
    "joins": [
        ""
    ],
    "where":"Account_Id=$accountId AND Setup_Id=$setupId AND
EndPointClass.Type=Event-Occurrence"
}
```

5.3 Portal Dynamic Custom Config Parameter List

The portal shall implement the following 'Dynamic Custom Config Parameter lists' in order to help manage the SiESocom pipeline.
The list from these queries shall be display to the user as either a dropdown (single selection) or a radio list (multiple selection).

The portal shall then be able to return the single or multiple selection a the 'requested config paramater value' to the managing application.

5.3.1 List of AFE Input EPs

The portal shall create a list of eligible AFE input for the 'AFE Input EPs' plugin config parameter, as defined by the following pseudo-query:

```
SELECT <fully qualified EP serial number>, <all details of EP>
FROM <all devices in setup>
WHERE <sensor type and dataSchema IN <list of endPointClass.type>
```

After the user has selected one *or many* endpoint(s), the portal shall create the AFE dataIn structure per section above, set the "Input EPs" config parameter with this json and signal the application agent of the update.

```
{
```

```
        "cId":"<fully qualified EP serial number>",
        "cVal":"<all details of EP>",
        "table":"<all devices in setup>",
        "joins": [
            ""
        ],
        "where":"Account_Id=$accountId AND Setup_Id=$setupId AND <sensor
    type and dataSchema> IN <list of endPointClass.type>"
    }
```

5.3.2 List of AED Input EPs

The portal shall create a list of eligible AED input for the 'AED Input EPs' plugin config parameter as defined by the following pseudo-query:

```
SELECT <fully qualified Feature name>, <all details of Feature>
FROM <all AFE in setup>
WHERE <Feature type IN <list of endPointClass.type>
```

After the user has selected one *or many* feature(s), the portal shall create the AED dataIn structure per section above, set the "Input EPs" config parameter with this json and signal the application agent of the update.

```
{
    "cId":"<fully qualified Feature name>",
    "cVal":"<all details of Feature>",
    "table":"<all AFE in setup>",
    "joins": [
        ""
    ],
    "where":"Account_Id=$accountId AND Setup_Id=$setupId AND <Feature type>
IN <list of endPointClass.type>"
}
```

5.3.3 List of TSA Scalar Input EPs

The portal shall create a list of eligible TSA input for the 'TSA Input EPs' config parameter, as defined by the following pseudo-query:

```
SELECT <fully qualified EP serial number>, <all details of EP>
FROM <all devices and applciation in setup>
WHERE <dataSchema= "ScalarTimeSeries" >
```

Example of list:

```
CSP-B827EB24A797.Temperature.002
CSP-B827EB24A797.Humidity.003
CSP-B827EB2427B2.Temperature.002
```

CSP-B827EB2427B2.Humidity.003

After the user has selected one *or many* endpoint(s), the portal shall create the TSA dataIn structure indetical to the one described in section above, set the "TSA Input EPs" config parameter with this json and signal the application agent of the update.

```
{
    "cId":"<fully qualified EP serial number>",
    "cVal":"<all details of EP>",
    "table":"<all devices and application in setup>",
    "joins": [
        ""
    ],
    "where":"Account_Id=$accountId AND Setup_Id=$setupId AND
<dataSchema>=ScalarTimeSeries"
}
```

6 References

json verifier: https://jsonformatter.org/

json to avro schema: https://toolslick.com/generation/metadata/avro-schema-from-json