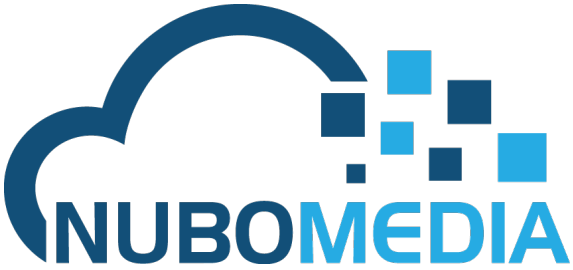
D3.7.1	
Version	1.0
Author	URJC
Dissemination	PU
Date	27/01/2015
Status	Final



D3.7.1: NUBOMEDIA social monitoring and optimization layer v1.

Project acronym:	NUBOMEDIA
Project title:	NUBOMEDIA: an elastic Platform as a Service (PaaS) cloud for interactive social multimedia
Project duration:	2014-02-01 to 2016-09-30
Project type:	STREP
Project reference:	610576
Project web page:	http://www.nubomedia.eu
Work package	WP3
WP leader	Giuseppe Antonio Carella
Deliverable nature:	Report
Lead editor:	Luis López
Planned delivery date	01/2015
Actual delivery date	27/01/2015
Keywords	NUBOMEDIA, Monitoring, debugging, billing, CDR, optimization

The research leading to these results has been funded by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement nº 610576

FP7 ICT-2013.1.6. Connected and Social Media

Contributors:

Luis Lopez (URJC)

Internal Reviewer(s):

Fco. Javier López (URJC)

Version History

Version	Date	Authors	Comments
0.1	07-01-2015	Luis Lopez	Initial Version
1.0	20-01-2015	Luis Lopez	Added KMS event software architecture.

Table of contents

List of Figures:

Acronyms and abbreviations:

API	Application Programming Interface
AR	Augmented Reality
CDN	Contend Distribution Network
FOSS	Free Open Source Software
IMS	IP Multimedia Subsystem
IoT	Internet of Things
KMS	Kurento Media Server
MCU	Multipoint Control Unit
NFV	Network Function Virtualization
QoE	Quality of Experience
QoS	Quality of Service
RTC	Real-Time Communications
RTP	Real-time Transport Protocol
SCTP	Stream Control Transmission Protocol
SFU	Selective Forwarding Unit
UE	User Equipment
VCA	Video Content Analysis
VoD	Video on Demand
WebRTC	Web Real Time Communications

1 Executive summary

This document presents the mechanism, principles, data structures and architectures that shall be used in NUBOMEDIA for collecting, storing and using relevant performance and usage information. This information shall be employed later by other NUBOMEDIA subsystem for performing functions such as optimization, billing and debugging.

2 Introduction

The development and maintenance of software systems is a complex process involving constant decisions that need to be taken with incomplete and sometimes inaccurate information. For this reason, having meaningful metrics offered to decision makers is a critical ingredient for successful technological developments. The ability to obtain the appropriate metrics in a seamless and direct manner impacts positively in all stages of the software development cycle including analysis, implementation, debugging, maintenance and optimization.

The number of metrics that may be gathered from a software system are quite large. Hence, for avoiding impacting negatively on performance (the own gathering of metrics consumes resources) and for maintaining metrics complexity under control, it is important to understand the different actors who shall be using the metrics as well as the objectives they pursuit. In the case of NUBOMEDIA, actors and objectives can be summarized in the following way:

- **Actor**: The NUBOMEDIA team
- **Objectives**: The objectives of the NUBOMEDIA team is to create an stable technological platform suitable to being installed and managed by third party providers hosting third-party developed applications.
- **Metrics of interest**:
    - Metrics making possible to detect and debug platform bugs
    - Metrics making possible to detect platform performance bottlenecks
    - Metrics making possible to automate the improvement of performance and QoE
    - Metrics making possible to measure satisfaction of the rest of actors

- **Actor**: NUBOMEDIA platform provider
- **Objectives**: The objective of the NUBOMEDIA platform provider is to install and manage a NUBOMEDIA platform for offering third-party developed applications on top of it, so that an internal or external need is satisfied.
- **Metrics of interest**:
    - Metrics making possible to detect and debug configuration problems on the platform instance.
    - Metrics making possible to detect and debug hardware problems on the platform instance.
    - Metrics making possible to optimize system performance of the platform instance
    - Metrics making possible to measure the satisfaction of actors using or depending on the platform instance.
    - Metrics making possible to bill

- **Actor**: NUBOMEDIA platform developer
- **Objectives**: The platform developer creates platform applications exposing capabilities to application developers. In this direction, the platform developer objective is to provide a number of low-level services that shall be used as building blocks by application developers.
- **Metrics of interest**:
    - Metrics making possible to detect and debug platform application bugs
    - Metrics making possible to detect platform application bottlenecks
    - Metrics making possible to bill
    - Metrics making possible to measure the satisfaction of application developers

- **Actor**: NUBOMEDIA Application developer
- **Objectives**: The application developer consumes the platform APIs created by the platform developer for creating final applications, which satisfy a need of end-users. Hence, the main objective of the application developer is to obtain and maintain such end-users.
- **Metrics of interest**:
    - Metrics making possible to detect and debug application bugs
    - Metrics making possible to detect application bottlenecks
    - Metrics making possible to bill
    - Metrics making possible to measure the satisfaction of end-users

- **Actor**: End-user
- **Objectives**: This actor uses applications for satisfying a need. This actor requires the application to work in agreement with its specification.
- **Metrics of interest**:
    - Metrics making possible to detect and solve usage problems.
    - Metrics making possible to evaluate costs.

Basing on these descriptions, this deliverable aims at providing some technological enablers suitable for making possible the different actors to obtain the metrics of their interest. However, it is important to remark that NUBOMEDIA offered metrics (i.e. the metrics offered off the shelf by the platform) cannot satisfy all the above mentioned needs, being necessary for the different actors involved to create specific purpose metrics when required.

In particular, this deliverable is specifically devoted to obtaining metrics suitable for evaluating QoS and QoE experiences of end users. Metrics related to system monitoring and performance are the target objective of NUBOMEDIA deliverable D3.6.1, while debugging metrics are the target objective of deliverable D5.4.

3 Objectives

The objective of this deliverable is to establish a technological strategy for the implementation of a monitoring system capable of generating meaningful metrics for platform administrators, platform developers, application developers, end users and even to feed automatic NUBOMEDIA QoE optimization mechanisms. Metric records should comply with the following requirements:

- They must be capable of providing time-stamped information relative to the lifecycle of media capabilities (i.e. creation and deletion events)
- They must be capable of providing time-stamped information relative to the topology of the media capabilities (i.e. media pipeline topology)
- The must provide time-stamped information relative to the QoE and QoS of the media capabilities (i.e. jitter, latency, packet loss, etc.)
- They must provide time-stamped information relative to errors of the media capabilities (i.e. software failures, hardware failures, etc.)
- They must provide time-stamped information relative to the to modules with specific media capabilities (i.e. faces recognized, etc.)

Metrics shall be generated by the platform through different mechanism and should be collected by the platform developer through a comprehensive set of APIs enabling

- Capture of relevant metric records for a platform application.
- Persistence of relevant metric records for a platform application.

Metrics shall be marshaled and stored as string values susceptible of being searched and filtered. Searching information shall include built-in identifiers and platform developer provided identifiers. The following searching and filtering criteria should be supported:

- Through type of metric
- Through type of capability (i.e. media element) generating the record.
- Through type of application generating the record, being the application an identifier provided by the platform developer to the metric generation system.
- Through username, being the username an identifier provided by the platform developer to the metric generation system.
- Through any other kind of identifier considered relevant for the platform developer (application developer, call id, session id, etc.)

Metrics shall be susceptible of being post-processed through value added algorithms capable of providing relational information among users as well as relational information among media server instances.

Metrics shall be supportive to different tasks of the different actors involved in development and maintenance of NUBOMEDIA applications and infrastructures. Hence, metrics should provide useful information for processes such as

- Platform maintenance and optimization.
- Application development and debugging.
- Billing.

4 High level architecture

From a systems perspective, this deliverable plans to specify, design and implement the appropriate software stack and extensions enabling the generation, consolidation and persistence of the different above-mentioned metrics. This stack is based on the architecture depicted in Figure 1.

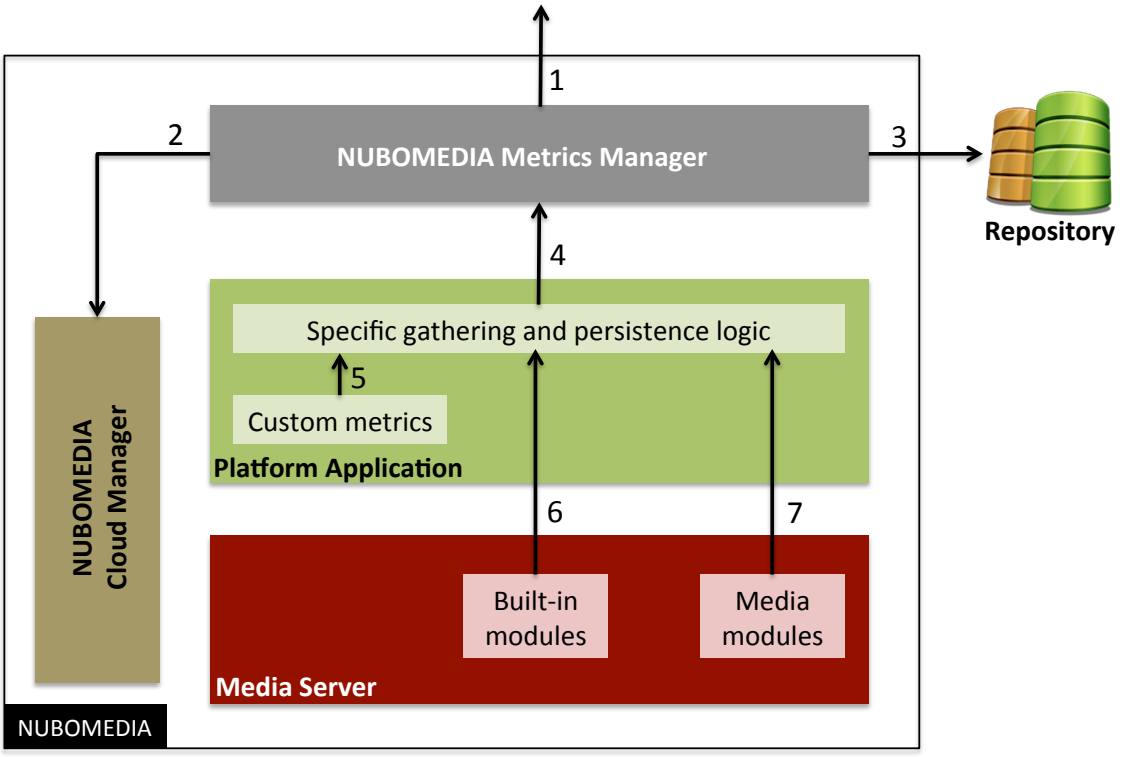


Figure 1. High level architecture of the NUBOMEDIA metrics subsystem and its relationships with the ret of NUBOMEDIA functions. This architecture also shows the different involved interfaces, which are numbered from top to bottom and from left to right.

The modules shown in this architecture are the following:

- NUBOMEDIA Metrics Manager. This module receives events from the Platform Application and persists them into a repository. At the northbound it offers the capability of querying both historic and in real-time metric records. This module may implement specific processing logic on metric records generating statistical trends or advanced composite metric records (e.g. social network metrics).
- Repository. This module provides the capability of persisting metric records into some kind of database where they can be queried later.
- Platform Application. This is the Platform Application held by the Application Server function, as described in the NUBOMEDIA Architecture (D2.4.1). The Platform Application needs to contain two types of logic created by the platform application developer.
    - Custom metrics logic. This logic generates specific purpose custom records related to internal application data.
    - Specific gathering and persistence logic. This logic gathers, processes and persists the metric records of interest, as decided by the platform application developer.
- Media Server. This represents the Media Server function, as described in the NUBOMEDIA Architecture (D2.4.1), which holds all media capabilities. The Media Server generates all built-in (i.e. provided off the shelf) QoS metrics. These have two types:
    - Built-in modules: which are metrics generated by Kurento Media Server built-in modules, as specified in D4.1.1
    - Custom media modules: which are metrics generated by third party Kurento Media Server modules, as specified in D4.1.1

- NUBOMEDIA Cloud Manager. This represents the CFM function, as specified in NUBOMEDIA Architecture (D2.4.1). This function may consume built-in or customized metrics for optimizing the different placement, scheduling and routing mechanisms that it implements.

The relevant interfaces shown in this architecture are the following:

- At the NUBOMEDIA Metrics Manager
  - At the southbound it exposes an interface enabling consuming and querying the different metrics both for external systems (1) and for internal subsystems (2)
  - It consumes a repository interface (3) for persisting and querying repository data.
- At the Platform Application
  - At the south bound it consumes an interface (4) exposed by the NUBOMEDIA Metrics Manager. This interface is used by the Platform Application Developer for exporting and storing the appropriate metrics of interest for the application.
  - Inside the application, the Platform Application Developer generates custom metric records, which are exposed to the application metric logic through an interface (5), which is internal to the application.
- At the Media Server
  - At the northbound it exposes an interface making possible for the Platform Application Developer to subscribe and gather built-in metric records being generated both by the core media server modules (6) and by third-party provided modules (7).

5 Implementation strategy

The implementation strategy for this deliverable is based on the following principles

5.1 Metric records

We define a metric record as unit of information containing a metric measurement at a given time. Metric records are an abstract concept that may be represented through different data structures and formats in different contexts. However, in all cases, a metric record must contain (and provide)

- A type: a string.
- A time stamp: measured in UCT standard type format.
- A tag: a string whose semantics depends on the type of record (see description bellow)
- Data: bytes whose format and semantics depend on the specific type.

Metric records shall be represented as typed classes in Java and as Objects in JavaScript. Record data may be exposed through the appropriate attributes performing the required unmarshalling.

Metric records shall be always marshaled and exchanged as JSON strings, with the possibility of using compact string representations (e.g. base-64) for the representation of binary data.

Built-in metric records shall be defined by the appropriate modules generating them with a precise specification on the data format. For built-in metric records, the Platform Application Developer shall be able to incorporate a customized string as tag.

Customized metric records shall be defined and consumed by the Platform Application Developer following the specific application needs.

5.2 Built-in metric records

Built in metric records shall be organized in 4 different types (eventually they may have subtypes):

- CDRs: These indicate lifecycle events of all media capabilities. Most basic lifecycle events are creation and removal.
- Stats: There indicate QoS metrics. This are of particular interest on network endpoints (i.e. media capabilities getting media from the network or sending media from the network).
- Errors: These indicate errors on media capabilities.
- Media Events: These are events associated to the specific capabilities of media elements. Media server modules shall use this type of records for indicating specific purpose event on the media streams (e.g. a face has been detected, etc.)

Accordingly to the Kurento Media Server description provided in NUBOMEDIA Deliverable D4.1.1, built-in records are generated by the following components:

- All media objects should generate CDRs
- Networked endpoints should generate Stats
- All media objects should generate Errors
- Filters, both built-in or based on external modules, might generate Media Events.

5.3 Gathering metrics and persisting metrics

Built-in metrics shall be made available for the Platform Application Developer under request. Media Server APIs (specifically the NUBO-MAPI API as described in NUBOMEDIA Deliverable D5.1) shall expose the capability of subscribing to metrics at the media object level. This means that the Platform Application Developer shall be able to execute custom logic upon reception of one or several metric records. Whenever a specific record is of interest for the platform application, the developer shall incorporate into the above mentioned logic the appropriate code filtering, pre-processing, publishing or persisting the record. Following this scheme, the platform shall not collect any kind of metrics of records off the shelf.

For publishing and persisting metric records, the NUBOMEDIA infrastructure shall provide an API (i.e. one or several Java interfaces) for integrating the specific application logic with the NUBOMEDIA Metrics Manager in a seamless way.

In the gathering of metrics, the Platform Application Developer shall be able to mandate to the specific media objects to generate the metric records including a customized application dependent tag. This tag may be extremely useful for debugging and monitoring purposes. Just as an example, the tag might contain information such as the following:

- Username owning the specific media object.
- IP address and port of the holding media server.

- Involved SLAs.
- Application status.
- Etc.

This application-dependent tag might be used at a later stage for filtering metric records accordingly to custom criteria.

5.4 Processing metrics

Independently on the custom processing that the Platform Application Developer may implement as part of the platform application logic, the NUBOMEDIA Metrics Manager shall be able to perform pre-and-post-processing of metric records in order to generate composite and non-evident additional metrics, which may include:

- Metrics statistics (e.g. average, standard deviation, moments, etc.)
- Interaction statistics (e.g. social networks derived form the interactions and mathematical characterization of them)
- Historical trends (e.g. windowed averaged, etc.)

These composite metrics shall be exposed to external systems and internal subsystems as the rest of metrics following the metric record model described above.

6 Implementation status

As shown in Figure 1, the implementation of the NUBOMEDIA social monitoring subsystem requires to develop 4 different types of components:

- Components at the Media Server layer for capturing and publishing the low level media information.
- Components at the Platform Application layer for generating custom metrics and for subscribing, processing and persisting metric records.
- A Metrics Manager providing the appropriate interfaces and routines to the rest of the architecture.
- A metric record repository.

As per NUBOMEDIA R3, the implementation has been concentrated on the low level components at the Media Server layer. For this, the implementation has been concentrated on extending Kurento Media Server, as described in NUBOMEDIA deliverable D4.1.1, for providing the following additional capabilities:

- Data types for representing metric records in its different flavors.
- An event management subsystem providing through the Kurento Protocol, as described in NUBOMEDIA deliverable D4.1.1, to applications the ability to subscribe to the events (i.e. metric record types) of interest on each media object.
- A basic set of initial events (i.e. metric records) providing preliminary relevant information related to Errors, CDRs, Stats and Media Events.

The following sections are devoted to analyzing such implementations:

6.1 Metric record data types

From the perspective of Kurento Media Server, all metric records are published as events. This means that a metric record can be seen as asynchronously generated data to

which application developers can subscribe. For this purpose, we have created several families of event data types, which are described bellow.

6.1.1 Common event data types

We call common events to event data types that may be published my all media objects, where media object is a way of specifying any KMS media element being a subclass of `KmsElement` (see NUBOMEDIA Deliverable D4.1.1 for having more information on this base class).

Currently, there is just one common event data type:

- `Error`: This event is published whenever an error (any kind of error) is detected at a specific media object instance.

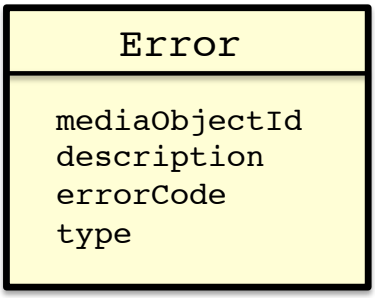


Figure 2. The Error event data type contains the identity of the media object generating the error, a textual description, an numeric code and an error type.

As per NUBOMEDIA R3, common event data types do not provide timestamp nor tagging, which shall be probably added during NUBOMEDIA R4.

6.1.2 Lifecycle events

We call lifecycle events to event data types informing about the creation or deletion of media objects. Lifecycle events are published through a special object called `ServerManager`, where applications can subscribe to them.

Currently, there are two lifecycle events:

- `ObjectCreated`: This event is published whenever a media object is created. It can be used as CDR metric associated to media object lifecycle.
- `ObjectDestroyed`: This event is published whenever a media object is destroyed. It can be used as CDR metric associated to media object lifecycle.

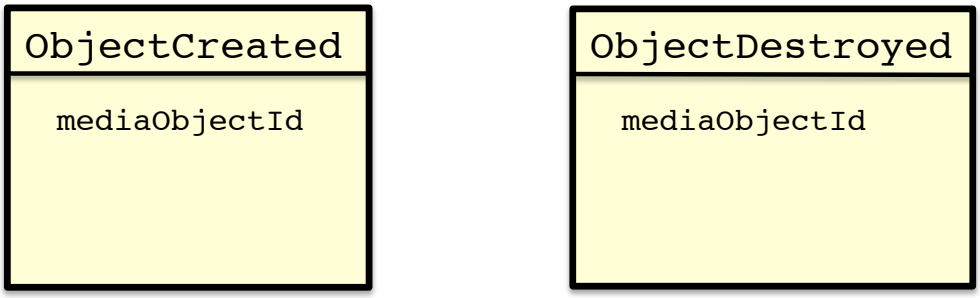


Figure 3. ObjectCreated and ObjectDestroyed data structures only contain a reference to the corresponding object owning the lifecycle event.

As per NUBOMEDIA R3, lifecycle event data types do not provide timestamp nor tagging, which shall be probably added during NUBOMEDIA R4.

6.1.3 Media event data types

We call media events to event data types that may be published by specific off-the-shelf KMS media objects. Hence, these types come in a built-in manner on any KMS distribution. A non exhaustive list of currently available media events is the following.

Among kms-core modules

- `SessionEndpoint`:
  - `MediaSessionTerminated`. Indicates a media session is terminating.
  - `MediaSessionStarted`. Indicates a media session is starting.

Among kms-element modules

- `HttpPostEndpoint`:
  - `EndOfStream`. Indicates the end of media has been reached.
- `PlayerEndpoint`:
  - `EndOfStream`. Indicates the end of media has been reached.

Among kms-filters:

- `ZBarFilter`:
  - `CodeFound`. Indicates a bar or QR code has been found.

Please, refer to KMS documentation, as presented in NUBOMEDIA Deliverable D4.1.1, for having full information in relation to these events.

It is important to remark that, from a software engineering perspective, all media events inherit from a common class called `Media`. After that, the specific attributes of each media event are defined in the KMD of the media object publishing it (please, refer to NUBOMEDIA Deliverable D4.1.1 for learning more about the KMD IDL).

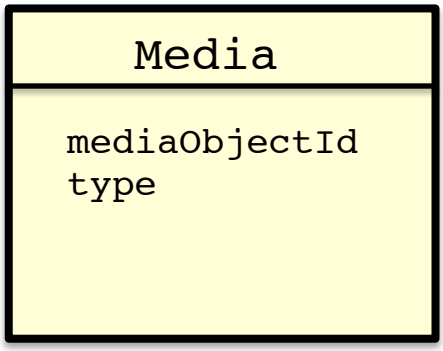


Figure 4. The Media data structure contains the source media object id and the type of media events being generated. This class is the base class for all media events.

The following code snipped shows an example on how a media event has been defined as part of the `PlayerEnpoint` capabilities.

```
{
    "name": "EndOfStream",
    "extends": "Media",
```

```
    "doc": "Event raised when the stream that the element sends out is
finished.",
    "properties": []
  }
```

As per NUBOMEDIA R3, media event data types do not provide timestamp nor tagging, which shall be probably added during NUBOMEDIA R4.

6.1.4 Custom media event data types

We call custom media events to event data types that may be published by third party modules not being distributed as part of KMS. Apart from that, custom media elements are the same than build-in media elements: they both inherit from the `Media` class and they both are specified through the KMD IDL.

Just for illustration, the following code snipped shows the KDM definition of a custom media element.

```
{
     "name": "CrowdDetectorDirection",
     "extends": "Media",
     "doc": "Event raise when a movement direction is detected in a
ROI",
     "properties": [
       {
         "name": "directionAngle",
         "doc": "Direction angle of the detected movement in the ROI",
         "type": "float"
       },
       {
         "name": "roiID",
         "doc": "Opaque String indicating the id of the involved ROI",
         "type": "String"
       }
     ],
   }
```

6.2 Implementation of the event management subsystem

The management of events has been implemented as part of KMS. For this, we have used the GStreamer event bus. KMS media elements comprise chains of multiple GStreamer media elements (see NUBOMEDIA Deliverable D4.1.1 for having more information on the relation between GStreamer and KMS). The internal workings of the event management subsystem are complex. However, it essentially works through the following steps:

- When a specific GStreamer elements needs to generate an event, it fires it into the bus.
- Every KMS media element has an event handler subscribed to the GStreamer bus.
- When an event is received at a handler, it is marshaled and published to the corresponding handler target address.

Hence, at the core of the event management subsystem we can find a notion of event handlers. Event handlers are implemented through the `EventHandler` class. A `EventHandler` instance can be seen as a delegate of a subscriber for a given event type

on a specific media object. In other words, when an application wants to receive a specific event type on a specific media object, it needs to subscribe an `EventHandler` instance on that specific media object and for that specific event type.

The `EventHandler` is then in charge of "listening" for a specific type of event on the GStreamer bus and, when it is received, it marshals the event data and sends it back to the subscriber. For this, the `EventHandler` needs to maintain a transport reference toward the subscriber. In other words, it needs to be capable of sending back to the application the event data.

For understanding how the EventHandler lifecycle is managed and how the transport mechanisms are implemented, we need some basic understanding of the KMS architecture. Please, check NUBOMEDIA deliverable D4.1.1 for having detailed information about it. For the objectives of this document, remark that the application is communicating with KMS through the Kurento Protocol. Hence, the event management needs also to take place through that protocol. This means that the protocol needs to provide two types of mechanisms:

- Event subscription mechanisms: Through this mechanism, a client application subscribes to a specific event type on a specific media object.
- Event publishing mechanisms: Through this mechanism, a media object publishes a specific event instance to a specific subscribed client application.

The event subscription mechanism is based on a `subscribe` message which has been added to the protocol. An example of such message is shown in the code snipped below

```
{
   "jsonrpc":"2.0",
   "id":4,
   "method":"subscribe",
   "params":{
     "object":"311861480",
     "type":"EndOfStream",
     "sessionId":"c93e5bf0-4fd0-4888-9411-765ff5d89b93"
   }
}
```

When KMS receives this message, it looks for the media object identified by the id specified by the `object` field and it adds a `EventHandler` instance to it listening to the events specified by the `type` field. In addition, the EventHanlder stores the `sessionId` filed as unique identifies of the transport mechanism (WebSocket) to send information back to the client.

The event publishing mechanism is based on a `onEvent` message, which has been added to the protocol. An example of such message is shown in the code snipped below

```
{
   "jsonrpc": "2.0",
   "id": 6,
   "method": "onEvent",
   "params": {
     "value": {
        "data":{
           "source":"311861480",
           "type":"EndOfStream"
       },
```

```
        "object":"311861480",
        "subscription":"353be312-b7f1-4768-9117-5c2f5a087429",
        "type":"EndOfStream",
      },
      "sessionId":"4f5255d5-5695-4e1c-aa2b-722e82db5260"
    }
  }
```

Basically, when an `EventHandler` receives from the bus an event of its interest, it locates the transport return address of the client using the corresponding `sessionId` (which was stored as part of the subscription processing) and sends to it the marshaled event, which contains the object originating the event, the identity of the subscription (for demultiplexing in case of parallel subscriptions) and the additional data associated to the event itself.