D5.5 FINAL VERSION OF SEMANTICALLY ENRICHED DATA INTEGRATED PLATFORM

Grant Agreement nr	611057
Project acronym	EUMSSI
Start date of project (dur.)	December 1st 2013 (36 months)
Document due Date :	31st July 2016
Actual date of delivery	9th September 2016
Leader	UPF
Reply to	jens.grivolla@upf.edu
Document status	Submitted

Project co-funded by ICT-7th Framework Programme from the European Commission

Project ref. no.	611057
Project acronym	EUMSSI
Project full title	Event Understanding through Multimodal Social Stream Interpretation
Document name	EUMSSI D5.5Final Version of Semantically Enriched Data Integrated Platform
Security (distribution level)	PU - Public
Contractual date of delivery	31st July 2016
Actual date of delivery	9th September 2016
Deliverable name	D5.5 Final Version of Semantically Enriched Data Integrated Platform
Type	P – Prototype
Status	Submitted
Version number	v1
Number of pages	35
WP /Task responsible	LUH / UPF
Author(s)	Jens Grivolla
Other contributors	Raul Marin
EC Project Officer	Mrs. Alina Lupu Alina.LUPU@ec.europa.eu
Abstract	Documentation for the final version of the integrated EUMSSI platform.
Keywords	MongoDB, Solr, UIMA, processing architecture
Circulated to partners	Yes
Peer review completed	N/A
Peer-reviewed by	N/A
Coordinator approval	Yes

TABLE OF CONTENTS

1. Background 5
2. Introduction 6
3. Main Advances over the First Version 7
   3.1. Infrastructure and performance 7
   3.2. Segment handling 7
   3.3. Crawlers and metadata mappings 7
   3.4. Text processing and complex workflows 8
   3.5. Quotation and attribution handling 8
4. Architecture and component overview 9
   4.1. Architecture overview 9
   4.2. Crawlers 11
   4.3. Preprocess 11
   4.4. EumssiAPI 12
   4.5. Pipelines 12
   4.6. Indexing 12
5. Installation and setup 13
   5.1. Setup 13
      5.1.1. Prerequisites 13
      5.1.2. Configuration 13
   5.2. Deployment 13
      5.2.1. Crawlers 13
      5.2.2. Preprocess 14
      5.2.3. EumssiAPI 15
      5.2.4. Audio/Video components 15
      5.2.5. UIMA based pipelines (uima-pipelines) 15
      5.2.6. Indexing 15
   5.3. Future work 16
6. Content Processing API 17
   6.1. Core services 17
   6.2. External components 18
   6.3. Queue retrieval 19
   6.4. Input data retrieval 19
   6.5. Analysis result upload 20

6.6. Meta response 21
6.7. Feedback API 22
7. Analysis modules and result formats 23
7.1. External analysis components 23
7.1.1. OCR 23
7.1.2. Person identification 25
7.1.3. Shot detection 27
7.1.4. ASR 27
7.1.5. Speaker recognition 28
7.2. UIMA pipelines 28
8. Metadata mappings 29
8.1. General fields: 29
8.2. Segment fields: 29
8.3. Metadata fields: 29
9. The application interface 32
9.1. Examples 33
9.1.1. Regular queries 33
9.1.2. Faceting 33
10. CONCLUSIONS 34
11. REFERENCES 35

1. BACKGROUND

This document presents the final version of the semantically enriched integrated EUMSSI platform, corresponding to the fourth project milestone (M32), which has been developed during the first 32 months of tasks 5.3 and 5.4. It is an update to D5.4.

It presents the current state of the platform, and the advances over the first version of the platform presented in D5.4. As with previous deliverables in WP5, the most up-to-date documentation is to be found in the EUMSSI Github wiki.

Development of the core platform goes hand in hand with the development of the demonstrators in WP6, which is documented in D6.2 and D6.3, with an upcoming update in D6.4 and D6.5.

2. INTRODUCTION

The platform is in charge of handling the complete EUMSSI workflow, from ingesting data, through processing, up to making the processed data available for end applications and demonstrators. It is thus the connection point for all components that form part of the system and are developed throughout the project, as well as the central data storage and management system.

The prototype is functional and currently deployed and accessible on the server at demo.eumssi.eu (some example queries can be found in section 9).

All code is available on Github at https://github.com/EUMSSI/EUMSSI-platform, and the documentation is continuously being updated on the platform wiki at https://github.com/EUMSSI/EUMSSI-platform/wiki. This deliverable is in fact a snapshot of the live wiki, with only minor adaptations to make it fit the required format.

Also included is a section highlighting the main advances in the platform compared to the M24 milestone (section 3).

3. MAIN ADVANCES OVER THE FIRST VERSION

While the first version presented in D5.4 was mostly feature complete regarding the core functionality and allowed the management and processing of data in the EUMSSI platform, there has still been significant work on updates and maintenance of the platform since the M24 milestone, as seen in dozens of tickets relating to the platform that were closed on the project's internal bug tracker[1] (including enhancements, tasks and defects). This section presents some of the main improvements.

3.1. Infrastructure and performance

While performance of the platform was mostly very good after the improvements (hardware infrastructure and software) introduced in year 2, a few bottlenecks showed up again as the amount of data and usage of the platform grew over time.

This has led to a quite major change to the handling of processing queues, trading some flexibility in being able to easily add new processing queues to the platform with zero overhead (which was essential in the initial phases of the project) for much improved performance. The new solution initializes the processing state for each registered processing queue on content ingestion, and thus requires setting the state for all existing content when a new processing queue gets added to the system (which takes a few minutes to complete). In turn, this avoids the reliance on "unitialized" (or null) processing state which makes the corresponding MongoDB queries, which are run constantly, much more efficient.

3.2. Segment handling

New types of segments have been added to the platform, in addition to speech transcript based segmentation, in particular segments based on OCR and person identification.

3.3. Crawlers and metadata mappings

There were further improvements to the crawlers in the EUMSSI platform.

Content from the Wikipedia Current Events Portal has now been fully integrated in the platform, as well as functionality to retrieve Youtube comments.

The platform now continuously ingests tweets using the Twitter Streaming API as well as news articles and multimedia data from Deutsche Welle (using their recently released new API), and receives periodic updates from a variety of news sources, Youtube, and the Wikipedia Current Events Portal. Metadata mappings were further improved to make all useful information available to analysis processes as well as the end applications (demonstrators), while ensuring full compatibility across all data sources.

[1] EUMSSI-TRAC powered by Trac 1.0.9 at http://demo.eumssi.eu/trac

3.4. Text processing and complex workflows

Natural language processing using UIMA is now fully integrated in the platform. Several analysis pipelines have been defined and incorporated to perform Named Entity Recognition and Linking, Sentiment Analysis and several other tasks on a variety of sources, including Twitter, textual information from news articles and videos, as well as the output of OCR and ASR processing.

Some more complex workflows have been implemented, including applying NLP to OCR output and in turn using the results to improve person identification (which itself combines visual and audio analysis).The advances in Y3 have focused on the time-aligned integration of multiple annotation layers, along with the storage, retrieval, and merging of outputs produced by different processing pipelines. This also constitutes the basis for the metadata enrichment tasks, which leverage the existing information produced by different sources.

Execution of the analysis pipelines has been automated to ensure timely processing of new content. In addition, text processing is now also available as a real-time on-demand service, and is used by the journalistic storytelling assistant for on-the-fly analysis of the journalist's input.

3.5. Quotation and attribution handling

In addition to the document-level representation of information (in MongoDB for internal processing as well as through Solr indexes for the applications), new MongoDB collections and Solr cores were created to represent represent relational (e.g. "who says what?") data, similarly to how segment indexes are use to enable sub-document level retrieval.

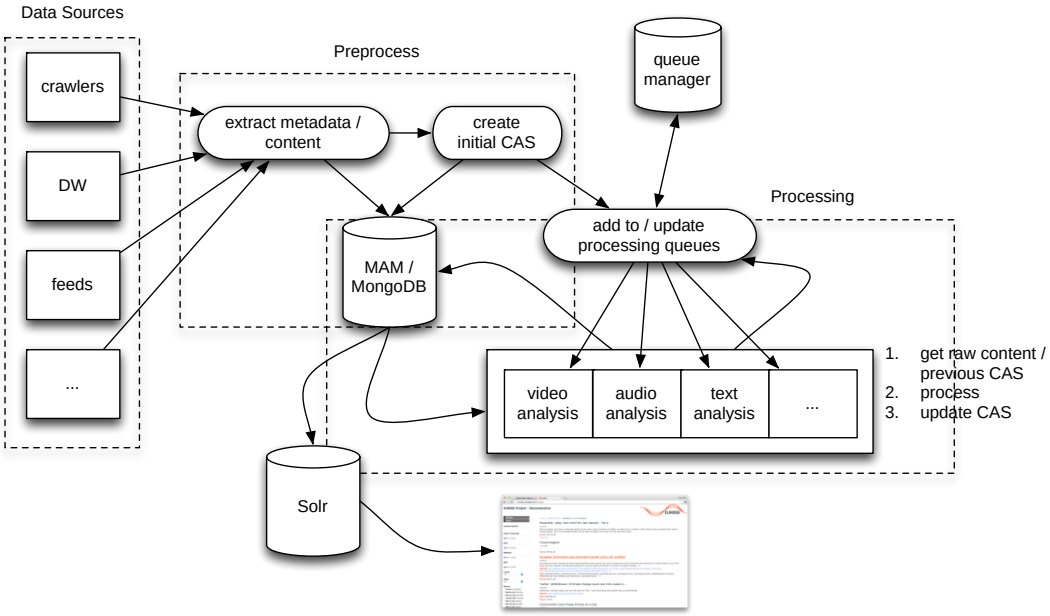
This builds the foundation for new application functionalities, such as the targeted search for politicians' opinions or statements about certain topics, and much more.

4. ARCHITECTURE AND COMPONENT OVERVIEW

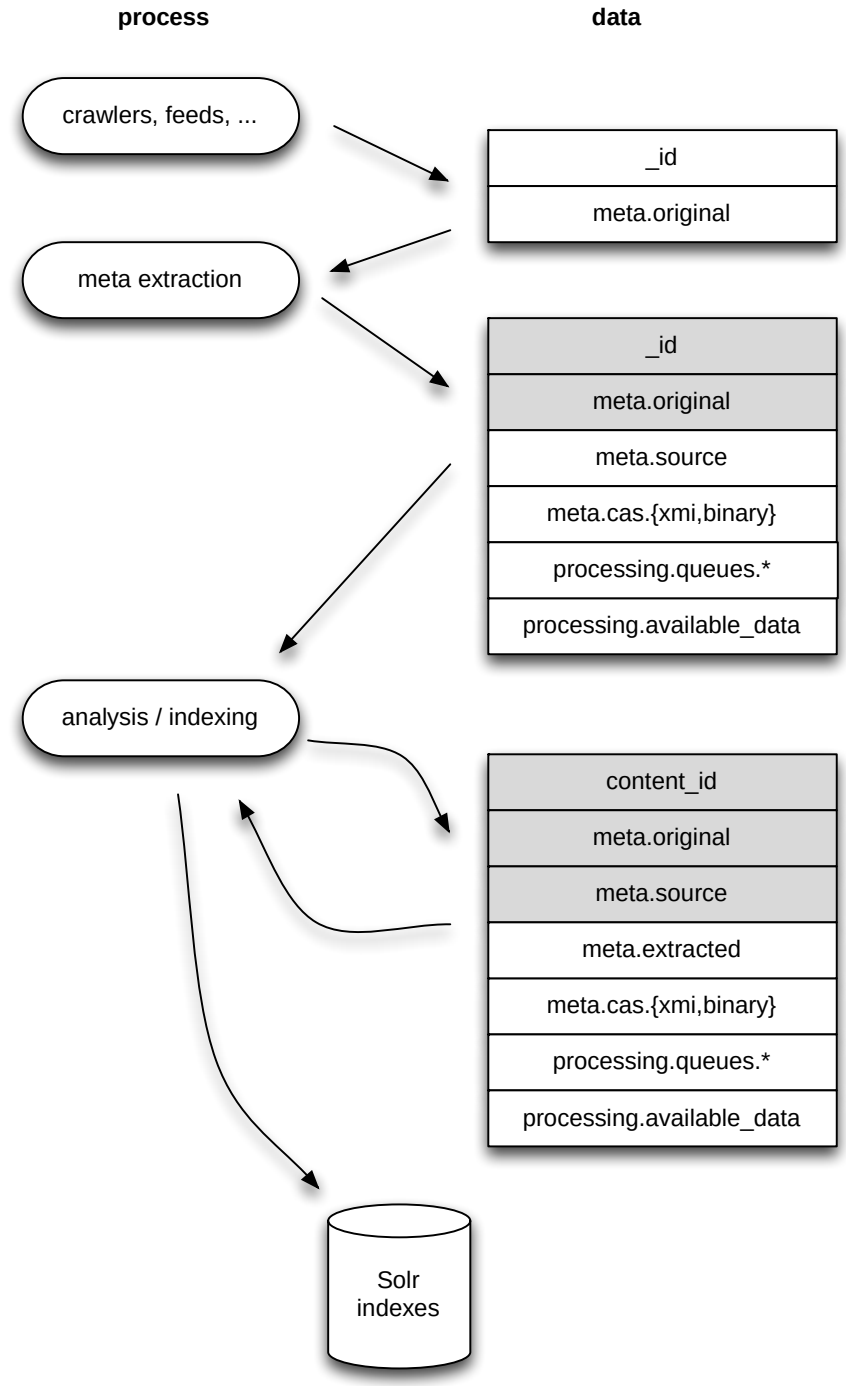
4.1. Architecture overview

The EUMSSI platform functions as a set of loosely coupled components that only interact through a common data backend (MongoDB) that ensures that the system state is persisted and can be robustly recovered after failures of individual components or even the whole platform (including hardware failures).

All components run independently and can be seen as basically "stateless" in that they maintain only the information necessary for immediate execution. As such it should be possible to restart components without affecting the system, making it relatively easy to ensure the overall reliability of the platform.



Seen from the point of view of communications and data flow, this can be represented as follows (with the active components on the left, and the document representation as maintained in MongoDB on the right):



The components that make up the EUMSSI platform can be organized into the following categories:

- the MongoDB backend
- crawlers (data sources)
- preprocessors (metadata converters)

- API layer
- analysis components
- indexing (Solr)
- demonstrators (web apps)

Crawlers, preprocessors and API layer are maintained as part of the core EUMSSI platform. The MongoDB database is installed separately and managed from within the platform components (with little or no specific configuration and setup), and the same goes for some external dependencies such as having a Tomcat server on which to run the API layer and Solr.

Analysis components for video and audio are fully external and independent and communicate with the platform through the API layer. Text analysis and cross-modality components are implemented as UIMA components and run as pipelines integrated into the platform using custom input (CollectionReader) and output (CASConsumer) modules.

Components that are part of the core platform can be found on GitHub and are organized into directories corresponding to the type of component. More detailed information about those components may be found in their respective README.md files.

4.2. Crawlers

Crawlers make external data sources available to the platform. Some crawler components are run only once to import existing datasets, whereas others feed continuously into the platform.

The following crawlers are currently integrated in the EUMSSI platform:

- twitter (Twitter streaming client)
- Deutsche Welle (API client for DW content)
- News (crawler for a variety of newspapers, e.g. El Pais, Die Zeit, Le Monde, The Guardian, ...)
- Youtube (crawler for Youtube content, by keywords and specific channels)

4.3. Preprocess

Preprocessing takes original metadata from the different sources and transforms it into a unified representation with a common metadata vocabulary. All preprocessors share a common core, represented by the EumssiConverter class, so that each converter only needs to define the needed field mappings and possible transformations to be applied to the data (such as date or language code normalization).

Each source format has a specific converter, all of which can be found in the preprocess folder.

4.4. EumssiAPI

The EUMSSI API abstracts away from the underlying storage (MongoDB and CAS data representation) to facilitate access for external components such as video and audio processing. It acts as a light-weight layer that translates between the internal data structure and REST-like operations tailored to the needs of the components.

4.5. Pipelines

Text processing as well as cross-layer integration and analysis is performed through UIMA pipelines that read an existing CAS representation of the document from the MongoDB backend, and write back a modified CAS with added annotations (and possibly layers/views) as well as extracted or "flattened" metadata that can be used by other components (e.g. a list of all detected entities in the document).

4.6. Indexing

Indexing takes care of making the metadata (from the original source as well as automatically extracted) available to demonstrators and applications by mirroring the data on a Solr server that is accessible to those applications. It is performed using mongo-connector, leveraging built-in replication features of MongoDB for low-latency real time indexing of new (even partial) content, as well as content updates.

5. INSTALLATION AND SETUP

5.1. Setup

5.1.1. Prerequisites

The platform and its components need a Linux system (tested on Ubuntu >=13.10) with the following preinstalled software:

- Python >= 2.6 with virtualenv
- MongoDb (>= 3.0)
- Tomcat (tested on Tomcat 7 and 8)
- Apache httpd as proxy to Tomcat webapps (for access control and URL rewriting)
- Java >= 1.7 (>= 1.8 for some UIMA pipelines)

It is assumed that the server has a full checkout (clone) of the EUMSSI-platform repository.

5.1.2. Configuration

- MongoDB: local access, may be password protected (only tested without passwords)
- Tomcat: no special configuration
- Apache: configure proxying for EumssiAPI webapp and possible Solr queries for demonstrators
- components: see README.md for each component

5.2. Deployment

At this point, components need to be started individually (and manually). MongoDB, Tomcat, etc. are assumed to be running. MongoDB databases are created on the fly by the components with no previous configuration unless a specific access control configuration is desired.

The following sections correspond to the folders in the EUMSSI-platform repository:

5.2.1. Crawlers

This section includes components to import data from different sources into the platform.

The existing crawlers are written in Python and prepared to run in separate virtual environments. All dependencies are specified in the requirements.txt file and can be installed using the pip install -r requirements.txt command (see individual components' README.md for more detailed instructions).

Crawlers can either run continuously or be run manually for one-time imports of data dumps. Periodic data retrieval could be handled through `cron` jobs or similar, but at this point the preferred option is for the crawler itself to handle scheduling (e.g. by sleeping for appropriate delays when no more data is available).

Continuous crawlers

- `twitter`: Twitter streaming client

Periodic crawlers

- `Youtube-Video`: get videos matching a list of keywords to follow.
- `News-import`: get videos from various newspapers (needs separate MySQL database)
- `DW-api-video-crawler`: get videos from Deutsche Welle archives (using API access)

One-time crawlers

- `twitter-import`: import of Twitter data from JSON dump
- `DW-import/news-fracking-import.py`: import of small DW "fracking" selection

deprecated:

- `DW-Solr-import`: import dump of DW data from Solr server on local host (needs to be setup and running first)
- `DW-import/video-import`: import older DW data
- `DW-import/news-import.py`: import May 2015 DW dump

5.2.2. Preprocess

Preprocessing components are in charge of converting the source metadata from different sources to the unified EUMSSI metadata format (see section 8). These components, usually written in Python, query the DB for new content given a specific source format (e.g. `twitter-api-v1.1`) and create a `meta.source` field with the converted metadata. Additionally they can set appropriate flags in the `processing.available_data` section of the document.

All preprocessing components run continuously, sleeping for some time when no more documents are available for conversion to avoid overloading the DB with queries. Deployment is similar to the continuously running crawlers, and detailed instructions can be found in each component's `README.md`.

5.2.3. EumssiAPI

The EUMSSI API layer is a Java webapp intended to be run inside a Java Servlet container such as Tomcat.

1. Follow configuration instructions in README.txt (adjusting files in EumssiApi/src/main/resources/eu/eumssi/properties/)
2. Generate .war file using mvn clean package
3. Deploy webapp (the .war is found in the target/ directory

5.2.4. Audio/Video components

These components are external to the platform and need to be set up separately.

5.2.5. UIMA based pipelines (uima-pipelines)

Text processing and cross-layer integration are implemented as UIMA components. Currently, deployment is done separately for each pipeline by locally running each pipeline's `.jar`. The pipelines communicate with the platform through a custom CollectionReader component that retrieves documents that are available for processing.

1. generate .jar for all components by running mvn clean package from the uima-pipelinesroot directory.
2. launch each pipeline by running java -jar <pipeline_name>/target/<pipeline_name>-with-dependencies.jar

5.2.6. Indexing

Preparation

1. enable replication on MongoDB if necessary
   i. add replSet=rsEumssi to /etc/mongod.conf, restart mongod service
   ii. from the mongo shell execute rs.initiate()
   iii. in case of problems run rs.initiate with an explicit host definition, e.g.rs.initiate({"_id":"rsEumssi","members":[{"_id":1,"host":"localhost:27017"}]}) (see stackoverflow)
2. launch solr server with the configuration found in indexing/solr (and ensure that it always runs)
3. virtualenv; . bin/activate; pip -r requirements.txt

Running

1. execute sync_all (or ./sync_content_items, ./sync_tweets and ./sync_segments) to startmongo-connector with appropriate arguments, and keep it running continuously
2. in case of problems, perform a full resync

i. run syncbg_all to index to a new collection
ii. stop sync_all if running
iii. swap cores in Solr admin
iv. move all.timestamp.XXX to all_timestamp
v. run sync_all to keep syncing to the newly swapped in core

5.3. Future work

The EUMSSI platform does not yet have a unified way to deploy, monitor and restart components (either for initial deployment or in case of failures).

We are currently investigating possible approaches, in particular using `systemd` to register components as services that get started and monitored automatically (or `upstart` if `system` does not become easily available on Ubuntu in the near future). Use of Docker to package components with their dependencies is also being considered.

6. CONTENT PROCESSING API

6.1. Core services

The initial processes, up to the creation of the initial CAS, are expected to run independently and interact directly with the database (MongoDB). No additional abstraction in the form of a service layer with an API (apart from MongoDB's own API) is expected to be used, in order to avoid having to maintain additional long-running server processes, and thus introducing additional points of failure and potential robustness issues.

These processes only create new fields, and don't update existing fields, and thus avoid any concurrency issues that would need to be handled by a separate service layer. Also, the processes form part of the core EUMSSI platform and will be developed and maintained as a part of that platform.

The API for these can thus be seen as being MongoDB's API, which has integration libraries for a large variety of programming languages. The following operations will be performed (nested fields are in MongoDB dot-notation):

- crawlers / feeds
  - create new document with randomly generated UUID,
  - meta.original reflecting the original metadata of the content item, as well as
  - source and meta.original_format fields
- meta extraction
  - read meta.original
  - add fields meta.source to document

These components work on the following document structure within MongoDB:

```
{
  "content_id" : UUID,
  "source" : SOURCE,
  "meta" : {
    "original_format" : SOURCE_FORMAT
    "original" : ORIGINAL_SOURCE_METADATA,
    "source" : EUMSSI_SOURCE_METADATA,
    "extracted" : METADATA_FROM_ANALYSIS
  },
  "cas" : {
    "xmi" : XMI_CAS,
    "binary" : BINARY_CAS,
    "json" : JSON_CAS
  },
  "processing" : {
    "queues" : {
      "queue1" : "done",
      "queue2" : "in_process",
      ...
      "queueN" : "pending"
    },
    "available_data" : ["video", "text", "audio_transcript", "text_nerl", "video_ocr",
...],
```

```
    "results" : {
      "queue1" : RAW_RESULT
  }
}
```

where:

- UUID is a system-wide unique content id, created when first inserting the content into the system
- SOURCE is the name of the content source
- SOURCE_FORMAT is the format of the original metadata, used to determine the appropriate mapping to the EUMSSI schema
- ORIGINAL_SOURCE_METADATA is the metadata as provided from the original content fields
- EUMSSI_SOURCE_METADATA is the original metadata mapped to the EUMSSI vocabulary / schema
- XMI_CAS is the CAS serialized in XMI format
- BINARY_CAS is the CAS serialized in binary format (alternative to XMI_CAS)
- JSON_CAS is the CAS serialized in JSON format (work in progress)
- METADATA_FROM_ANALYSIS is metadata that is generated by EUMSSI analysis processes, in the form it should be indexed with Solr (taken from CAS or raw result upload)
- RAW_RESULT are processing results in the raw original format (before converting to CAS / mapping for indexing)

Normally, the CAS will be stored only in one of the available formats (likely JSON), but potentially different serializations could be used. The `meta.extracted` information can be used for analysis results that are used as inputs to other annotators (such as detected Named Entities as input to speech recognition), to avoid the overhead of extracting that information from the CAS on demand, and contains the information that is indexed in Solr for application use.

The `processing` section stores information necessary for coordinating the different analysis components. Availability of data (e.g. `{'available_data':['audio']}` is maintained separately from the queue processing states to reflect that a) not all content has the same data to begin with, and b) there may be an intermediate step between a content being processed by an analysis queue and the results being available for further processing (e.g. result transformations, CAS merging, etc.).

MongoDB allows to store structured information (corresponding to a JSON structure), so that the content of fields like ORIGINAL_SOURCE_METADATA can reflect whatever internal structure the original data had.

6.2. External components

Analysis and indexing are distributed processes, developed and executed by different teams, and not tightly integrated with the core EUMSSI platform. It is therefore advisable to provide an abstraction layer isolating these components from the data

storage and management backend implementation details. The APIs are REST-like and use JSON for transmitting structured data.

The main operations are:

1. retrieving the list of content items pending to be processed
2. retrieving input information for the analysis
3. uploading the analysis output

All API requests need to provide a `key` for authorization.

6.3. Queue retrieval

The first point is identical for all analysis or indexing processes and can therefore have a completely unified API. We use the following REST-like API:

Request:

```
GET http://<server>/queue/pending?key=<api_key>&queueId=<queue
ID>&filters=<filters>
```

where the optional parameter `filters` can contain additional user-defined filters in MongoDB JSON syntax, e.g. `filters={"meta.source.inLanguage":"fr"}` to only retrieve items in French.

Response (JSON):

The response consists of two parts, `meta` and `data`. The `meta` section contains information about the request, such as possible error codes, etc., whereas the `data` section contains the actual response. Shown below is the `data` section, the `meta` section is documented separately as it is common to all API calls.

```
'data': {
  'queue_id': QUEUE_ID,
  'timestamp': REQUEST_TIMESTAMP,
  'items': [
    ITEM_ID_1,
    ITEM_ID_2,
    ...
  ]
}
```

As can be seen, the definition of dependencies to determine what items are ready to be processed by a given queue is handled at the central server, and analysis components only need to specify their queue ID.

6.4. Input data retrieval

Analysis components need input data to work on, at the very least the URI of the original content. There are two main cases: components that need some (item-level)

metadata, such as the content URI, the list of relevant entities, etc., and components that need a more complete view of the document including fine-grained output from previous processes, etc. The latter will receive the complete CAS representation of the item, serialized as XMI, whereas the former will receive only select metadata in JSON format that is easier to use for non-UIMA components.

Request (metadata):

```
GET
http://<server>/item/meta?key=<api_key>&itemId=<content_id>&fields=<field1>,<
field2>,...
```

Response (JSON):

```
'data': {
  'item_id': CONTENT_ID,
  'timestamp': REQUEST_TIMESTAMP,
  'meta': {
<field1>: value,
<field2>: value,
    ...
  }
}
```

Values for the `fields` parameter:

- * returns the full document structure from MongoDB
- anything else can be a field or a subtree prefix
    - e.g. meta.source.mediaurl to retrieve the video or audio URL
    - meta.source (or meta.source.*) to retrieve the full source metadata subtree

Note: UIMA components do not need to individually use this API, given that they are executed within the project's central UIMA environment.

6.5. Analysis result upload

The upload of analysis results is quite dependent on the specific output of the analysis component, and we aim to allow components to return their results in a "natural" representation, with little adaptation and integration burden for component developers. Specific server-side modules then take care to convert that information into a UIMA compatible representation and merge it into the existing CAS for that content item. The result format should be JSON.

For UIMA components (or components that work directly on the CAS representation) merging should be avoided, meaning that no two such components can work concurrently on the same content item. Parallelization is however possible if those components are executed within the same UIMA-AS environment. Sets of components

that are executed within a common UIMA pipeline will present themselves as a single queue consumer.

We only outline the common part of the API, whereas the details such as the specific result format are worked out individually for each component.

Request:

`POST http://<server>/queue/results`

POST params (x-www-form-urlencoded):

queueId: `<queue id>`

key: `<API key>`

data:

```
[
  {
    'content_id': CONTENT_ID1,
    'result': SPECIFIC_RESULT_FORMAT
  },
  {
    'content_id': CONTENT_ID2,
    'result': SPECIFIC_RESULT_FORMAT
  },
  ...
]
```

Response: only a `meta` section is returned to indicate successful completion or errors.

6.6. Meta response

All API calls respond with a `meta` section, in addition to a possible `data` section. This section includes status codes, as well as textual messages for debugging. HTTP return codes are used in addition to the status codes in the JSON message.

NOTE: do not rely on the textual message for automated error handling!

Example:

```
'meta': {
    "message": "Metadata retrieved successfully",
    "code": "0",
    "status": "ok"
}
```

The status codes are defined in the JSONMeta class.

6.7. Feedback API

The EUMSSI platform provides two APIs for collecting feedback, `/feedback/report` for manually entered feedback such as bug reports or suggestions, and `/feedback/action` to record user interactions with the system (e.g. for recommendation, personalization, etc.).

A feedback service for reports (bugs or suggestions) is running on the API layer at `/feedback/report` with the following parameters:

- state: the current system state, serialized in a way that makes it possible to reproduce
- comment: a comment provided by the user describing the problem
- type: the type of report, e.g. "bug" or "suggestion"
- user (optional): the user reporting the problem

Example:

```
http://<server>/EumssiApi/webapp/feedback/report?type=bug&comment=there%20is%
20a%20problem&state={query:%22*:*%22}
```

An additional service is available for "actions", being any kind of user-related feedback to be used for personalization/recommendation, etc. This includes both explicit (e.g. likes) and implicit feedback (e.g. clicks, watches, ...).

It's at `/feedback/action` with the following parameters:

- user: the user initiating the action
- item: the item (e.g. article or video) the action relates to
- type: the type of action, e.g. "click", "watch", ...
- detail: additional detail information (free form, e.g. for debugging)

Example:

```
http://<server>/EumssiApi/webapp/feedback/action?type=like&user=eumssi_user&i
tem=1b0a69fc-f621-46de-812a-
1f00d772d758&detail=I%20really%20liked%20that%20video
```

7. ANALYSIS MODULES AND RESULT FORMATS

In addition to the Metadata formats exposed to the demonstrators and end user applications, the system internally stores analysis results in (sometimes more complete) "native" formats for each analysis type. These results can be found in MongoDB under `processing.results.<queue_name>` and are used internally for further processing and to generate the final output in `meta.extracted.*`.

Additionally, results are stored in UIMA CAS format in the `meta.cas.*` section of the database. Analysis components that work natively in UIMA (such as the text analysis engines) do not use `processing.results.*`, but work directly with the information in `meta.cas.*`.

7.1. External analysis components

Below are the different analysis components that make use of the EUMSSI API to upload their results. The name and dependencies correspond to the definitions in queues.properties.

7.1.1. OCR

- Name: `video_ocr`
- Dependencies (`processing.available_data`): `metadata`, `video`
- Inputs: `meta.source.mediaurl`
- Output format:

Index terms and text to show at 2 levels - the whole document or individual shots: "text_to_index_shotlevel", "text_to_index_doclevel", "text_to_display_shotlevel", "text_to_display_doclevel"

List of text detection: "VideoTextDetection"


```
"video_ocr": {
    "text_to_index_shotlevel": [
        {
            "text": "Surgeon ngoodshape ngOOdShape ingOOdShape l 1gOOdShape
ngoodsnape Suvgcon Fl OOdShape Dr. Bernhard Lukas Bornhard",
            "shot_id": 22
        },
        {
            "text": "Report Julia Richter Report. julia Rvpurt Re port",
            "shot_id": 30
        }
    ],
    "text_to_index_doclevel": "Surgeon ngoodshape ngOOdShape ingOOdShape l
1gOOdShape ngoodsnape Suvgcon Fl OOdShape Dr. Bernhard Lukas Bornhard",
    "text_to_display_doclevel": "Surgeon Dr. Bernhard Lukas",
    "text_to_display_shotlevel": [
        {
            "text": "Surgeon Dr. Bernhard Lukas",
```

```
            "shot_id": 22
        }
    ],
    "VideoTextDetection": [
                {
                  "mediaRelIncrTimePoint_HMSF": "00:01:52F14",
                  "imgname": "frame_0000002814",
                  "mediaIncrDuration_S": 6.72,
                  "mediaRelIncrTimePoint": 2814,
                  "mediaRelIncrTimePoint_S": 112.56,
                  "Hypotheses": [
                    {
                      "count": 30,
                      "text": "Surgeon",
                      "score": 31.5836,
                      "rank": 1
                    },
                    {
                      "count": 24,
                      "text": "ngoodshape Surgeon",
                      "score": 30.8028,
                      "rank": 2
                    },
                    {
                      "count": 7,
                      "text": "ngOOdShape Surgeon",
                      "score": 13.8028,
                      "rank": 3
                    },
                    {
                      "count": 2,
                      "text": "ingOOdShape Surgeon",
                      "score": 10.0715,
                      "rank": 4
                    },
                    {
                      "count": 7,
                      "text": "l 1gOOdShape Surgeon",
                      "score": 9.49399,
                      "rank": 5
                    },
                    {
                      "count": 1,
                      "text": "ngoodsnape Surgeon",
                      "score": 7.7293,
                      "rank": 6
                    },
                    {
                      "count": 1,
                      "text": "ngoodshape Suvgcon",
                      "score": 6.62443,
                      "rank": 7
                    },
                    {
```

```
                    "count": 1,
                    "text": "Fl OOdShape Surgeon",
                    "score": 6.56631,
                    "rank": 8
                  }
                ],
                "Location": {
                  "column": 79,
                  "width": 484,
                  "line": 577,
                  "height": 48
                },
                "mediaIncrDuration": 168,
                "id": 8
              },
              {
                "mediaRelIncrTimePoint_HMSF": "00:01:52F14",
                "imgname": "frame_0000002814",
                "mediaIncrDuration_S": 6.72,
                "mediaIncrDuration": 168,
                "mediaRelIncrTimePoint_S": 112.56,
                "Hypotheses": [
                  {
                    "count": 88,
                    "text": "Dr. Bernhard Lukas",
                    "score": 95.6609,
                    "rank": 1
                  },
                  {
                    "count": 1,
                    "text": "Dr. Bornhard Lukas",
                    "score": 8.05242,
                    "rank": 2
                  }
                ],
                "Location": {
                  "column": 425,
                  "width": 323,
                  "line": 547,
                  "height": 34
                },
                "mediaRelIncrTimePoint": 2814,
                "id": 9
              }
            ]
}
```


7.1.2. Person identification

- Name: video_persons
- Dependencies (processing.available_data): metadata, video, video_ocr, text_ocr-nerl?

- Inputs: meta.source.mediaurl, *some OCR and entity linking output (e.g.processing.results.text_ocr-nerl, entities from metadata (?)*
- Output format:

Person_Identification is list of people identified from either video, audio, or both.

- Each person has an ID and associated attributes.
- If the person has name, "has_name" field is 1 and there is also a list of alternative names.
- If the person is visible, "appearing" field is 1 and there is a thumbnail + a list of temporal face segments.
- If the person speaks, "speaking" field is 1 and there is a list of temporal face segments.
- Field "dbpedia_uri" is inherited from text analysis of NEL.


```
{
    "Person_Identification": {
     "P_28":
     {
          "appearing": 1,
          "name": "C0013",
          "thumbnail_frame_extracted": 2954,
          "attribute": {"gender": [], "face_gender": [], "role": [],
"speaker_gender": []},
          "has_name": 0,
          "face_tracks": [{"end_S": 119.04, "start_S": 117.36}],
          "dbpedia_uri": [],
          "thumbnail_face_image_url": "http://demo.eumssi.eu/images/11413ef2-
aaf0-124e-5eec-6714ac20d194/face_thumbnails/C0013.jpg",
          "speaking": 0
     },
     "P_37":
     {
          "appearing": 0,
          "name": "S26",
          "thumbnail_frame_extracted": [],
          "attribute": {"gender": [], "face_gender": [], "role": [],
"speaker_gender": "M"},
          "has_name": 0,
          "audio_segments": [{"end_S": 253.96, "start_S": 251.82}],
          "dbpedia_uri": [],
                  "thumbnail_face_image_url": [],
                  "speaking": 1
     },
     "P_3":
     {
          "appearing": 1,
          "name": "Laura_Birling",
          "thumbnail_frame_extracted": [],
          "attribute": {"gender": [], "face_gender": [], "role": [],
"speaker_gender": "F"},
          "has_name": 1,
```

```
                "name_hypothesis": ["", "_Laura_Burling", "Laura_Bnrling",
"Laura_Birting", "Laura_Blrling", "Laura_Birllng"],
        "audio_segments": [{"end_S": 179.73, "start_S": 154.33}],
        "face_tracks": [{"end_S": 179.44, "start_S": 152.12}],
        "dbpedia_uri": [],
        "thumbnail_face_image_url": "http://demo.eumssi.eu/images/11413ef2-
aaf0-124e-5eec-6714ac20d194/face_thumbnails/Laura_Birling.jpg",
        "speaking": 1
     }
    ]
}
```


7.1.3. Shot detection

- Name: `video_shots`
- Dependencies (`processing.available_data`): `metadata`, `video`
- Inputs: `meta.source.mediaurl`
- Output format:

JSON object video_shots contains "ShotDetection", which is a list of all shot detections. Each detection contains shot id, start and end timestamp, start and end frame, and id of the keyframe. Keyframes are uploaded to http://demo.eumssi.eu/images/{item_id}/keyframes/keyframe{keyframe_id}.jpg


```
{
    "video_shots":
                "ShotDetection": [
                {
                  "end_FN": "73",
                  "end_S": "2.92",
                  "start_FN": "0",
                  "start_S": "0",
                  "keyframe": "37",
                  "id": "1"
                },
                {
                  "end_FN": "116",
                  "end_S": "4.64",
                  "start_FN": "74",
                  "start_S": "2.96",
                  "keyframe": "95",
                  "id": "2"
                }
           ]
}
```


7.1.4. ASR

- Name: `audio_transcript`
- Dependencies (`processing.available_data`): `metadata`, `video` or `audio`, ...
- Inputs: `meta.source.mediaurl`, *entities from metadata*

7.1.5. Speaker recognition

- Name: `audio_speaker`
- Dependencies (`processing.available_data`): `metadata`, `video` or `audio`, ...
- Inputs: `meta.source.mediaurl`, *entities from metadata*

7.2. UIMA pipelines

UIMA pipelines for a variety of tasks are defined in
https://github.com/EUMSSI/EUMSSI-UIMA-pipelines

This includes named entitity recognition and linking, polarity detection, speech recognition correction and cleanup, various segmentation methods, quote extraction and indexing, and many more.

All UIMA pipelines (and their components) use a common type system, defined in https://github.com/EUMSSI/EUMSSI-UIMA-TS, for project specific annotations, while leveraging DKpro's type system (https://dkpro.github.io/dkpro-core/releases/1.7.0/typesystem/), which is currently gathering strong support in the UIMA community as a common basis for analysis engines from different institutions (and possible future "official" endorsement), for all "standard" NLP task.

8. METADATA MAPPINGS

This section documents the mappings of metadata onto the unified EUMSSI metadata schema (based on schema.org). The exact mappings for source metadata can be found in the corresponding preprocessing scripts following the naming convention `<data_format>2eumssi.py`, whereas the mapping of automatic analysis outputs is performed by scripts following the convention `<queue_name>2extracted.py`.

Additionally, the analysis outputs are converted to the UIMA CAS format for further processing using the corresponding CollectionReaders.

The resulting fields (which are available to applications through Solr) are the following:

8.1. General fields:

Field	Comment / Format
_id	internal EUMSSI id (UUID)
contentSearch	main search field (text + ASR + OCR + ...)
source	source name (e.g. "DW video", "Twitter", ...)

8.2. Segment fields:

Field	Comment / Format
parent_id	ID of the full document the segment belongs to
beginOffset	begin offset of segment relative to full document (in ms)
endOffset	end offset of segment relative to full document (in ms)
segmentType	type of segmentation used

8.3. Metadata fields:

Section	Field	Comment / Format
meta.extracted	audio_transcript	full audio transcript (best hypothesis)
	text_nerl.dbpedia.City	link to dbpedia (only cities)
	text_nerl.dbpedia.Country	link to dbpedia (only countries)
	text_nerl.dbpedia.LOCATION	link to dbpedia (location)
	text_nerl.dbpedia.ORGANIZATION	link to dbpedia (organization)

	text_nerl.dbpedia.PERSON	link to dbpedia (person)
	text_nerl.dbpedia.all	link to dbpedia (all types)
	text_nerl.dbpedia.other	link to dbpedia (doesn't match specific type category)
	text_nerl.ner.LOCATION	Named Entity (location)
	text_nerl.ner.MISC	Named Entity (miscellaneous)
	text_nerl.ner.ORGANIZATION	Named Entity (organization)
	text_nerl.ner.PERSON	Named Entity (person)
	text_nerl.ner.all	Named Entity (all types)
	text_polarity.discrete	opinion polarity (POSITIVE, NEGATIVE, NEUTRAL)
	text_polarity.numeric	opinion polarity (<0 negative, 0 neutral, >0 positive)
	video_ocr.best	OCR transcript (best hypothesis)
	video_persons.amalia	person recognition in Amalia.js format
meta.source	Author	article/tweet author
	contentLocation.0	source location (e.g. or tweets), latitude
	contentLocation.1	source location (e.g. or tweets), longitude
	datePublished	publication date
	Description	article/video description (text)
	description_html	article/video description (original markup)
	Duration	content duration in milliseconds
	Hashtags	Twitter hashtags
	Headline	headline/title
	headline_html	headline/title (original markup)
	inLanguage	document language (ISO 3166-2, lowercase)

	Keywords	keywords / tags (provided by source)
	Mediaurl	video / audio file link (or Youtube embedded player link)
	numberOfComments	number of comments (e.g. Youtube)
	numberOfFavorites	number of favorites (e.g. Youtube)
	numberOfLikes	number of likes (e.g. Youtube)
	numberOfRating	number of ratings (e.g. Youtube)
	numberOfViews	number of views (e.g. Youtube)
	Page	full raw webpage
	Publisher	news publisher (e.g. Zeit, Guardian, ...)
	Rating	rating (e.g. Youtube)
	Text	article text
	text_html	article text (original markup)
	tweetId	Tweet ID
	urlMentions	URLs mentioned (e.g. Twitter)
	userMentions	users mentioned (e.g. Twitter)
	websiteUrl	link to original document web site
	youtubeVideoID	Youtube ID

9. THE APPLICATION INTERFACE

Applications such as the demonstrators interact with the platform by querying the Solr indexes (AJAX-Solr is recommended). This is a purely read-only access to the data in the platform. Whereas the MongoDB backend database used for processing and storing the data is internal and access to it is protected, the Solr indexes of the EUMSSI project are open for read access (but could be protected for commercial uses of the platform).

At this point applications access the data by issuing queries directly to Solr, using the standardSolr API. An additional API layer may be added in the future if there is a need for applications that is not covered by issuing direct queries to Solr. Such a layer could in particular be necessary in settings that require more fine-grained access control.

The fields available in Solr directly reflect the fields as found in MongoDB, restricted to a subset of the available information (exluding e.g. fields needed to control processing, etc.). Compared to MongoDB's internal representation, documents get "flattened" as follows:

```
{
"a": 2,
"b": {
"c": {
"d": 5
}
},
"e": [6, 7, 8]
}
```

becomes:

```
{"a": 2, "b.c.d": 5, "e": [6, 7, 8]}
```

The fields that are available follow the EUMSSI metadata vocabulary, with a prefix depending on provenance (prefixes and some field names may still change):

- meta.source.*: metadata provided by the original content source, e.g.
  - meta.source.author
  - meta.source.datePublished
  - meta.source.headline
  - meta.source.mediaurl
  - meta.source.keywords
  - ...
- meta.extracted.*: metadata that was automatically extracted from the content through analysis processes

Additionally the `_id` field is available, as well as the `source`.
The available fields are documented in the Metadata mapping section.

9.1. Examples

Here you find a small selection of queries that can be performed by client apps using the EUMSSI Solr server:

9.1.1. Regular queries

Tweets published in the last 10 minutes:
http://demo.eumssi.eu/Solr_EUMSSI/content_items/select?q=%2Bsource%3ATwitter+%2Bmeta.source.datePublished%3A%5BNOW-10MINUTES+TO+*%5D&rows=100&wt=json&indent=true

Documents that have high quality video:
http://demo.eumssi.eu/Solr_EUMSSI/content_items/select?q=meta.source.httpHigh%3A%5B*+TO+*%5D&wt=json&indent=true

Documents in Spanish that mention Merkel in the headline:
http://demo.eumssi.eu/Solr_EUMSSI/content_items/select?q=meta.source.headline%3Amerkel&fq=meta.source.inLanguage%3Aes&wt=json&indent=true

9.1.2. Faceting

Get the number of tweets per day for the year 2014 (from the Twitter-DW collection):
http://demo.eumssi.eu/Solr_EUMSSI/content_items/select?q=source%3ATwitter-DW&rows=0&wt=json&indent=true&facet=true&facet.range=meta.source.datePublished&facet.range.gap=%2B1DAY&facet.range.start=2014-01-01T00:00:00Z&facet.range.end=2015-01-01T00:00:00Z

Keywords that co-occur with "spain":
http://demo.eumssi.eu/Solr_EUMSSI/content_items/select?q=meta.source.keywords:spain&facet.field=meta.source.keywords&facet=true&rows=0&wt=json&indent=true

10. CONCLUSIONS

The EUMSSI project has developed a fully working data storage, management, and analysis platform, ranging from continuous live data input (from real time feeds such as Twitter or periodic retrieval of a variety of sources), through the management of the full workflow including internal and external processing components, up to the publication of query endpoints for applications and demonstrators (with real time data updates). It is possible to define dependencies between different processes in order to e.g. apply text processing to the output of speech recognition, work on multiple time-aligned layers, and retrieve information at the collection, document, segment or relation level.

The platform has been used throughout the last two years, with different partners actively using the platform to retrieve data and upload analysis results. Two demonstration applications have been developed, using the application interface provided by the Solr server, providing a rich interactive experience to professional journalists as well as end users at home. Both applications, despite their very different objectives and design, are able to build on the same common basis that had been developed for the M24 milestone and since refined and adapted to suit all the different needs of these applications.

11. REFERENCES

Web references appear as hyperlinks throughout the document.