Social Semantic Emotion Analysis for Innovative Multilingual Big Data Analytics Markets

D3.2 MixedEmotions Big Data Platform Architecture, final version

Project ref. no	**H2020 644632**
Project acronym	**MixedEmotions**
Start date of project (dur.)	**01 April 2015 (24 Months)**
Document due Date	**30 September 2016 (Month 18)**
Responsible for deliverable	**Paradigma Tecnológico**
Reply to	**cnavarro@paradigmatecnologico.com**
Document status	**Final**

Project reference no.	**H2020 644632**
Project working name	MixedEmotions
Project full name	Social Semantic Emotion Analysis for Innovative Multilingual Big Data Analytics Markets
Document name	MixedEmotions_D3.2_30_09_2016_MixedEmotions_Big_Data_Platform_Architecture_final_version_PT
Security (distribution level)	PU
Contractual delivery date	30 September 2016 (M18)
Deliverable number	D3.2
Deliverable name	MixedEmotions platform Architecture, final version
Type	Other
Version	Final
WP / Task responsible	WP3 / Paradigma Tecnológico
Contributors	PT (Carlos Navarro De Martino, Hugo Viejo), NUIG (Paul Buitelaar, Cécile Robin, Mihael Arcan, Ian Wood), UPM (Carlos Ángel Iglesias, Fernando Sánchez), ST (Giovanni Tummarello), PX (Pavel Matějka, Aneta Černá), BUT (Lubomír Otrusina), UP (Hesam Sagha), DW (Andy Giefer), ES (Francesco Adolfo Danza, Vincenzo Masucci)
Project Officer	Martina Eydner

Index

m1. English Sentiment Extraction

m2. Czech and English Sentiment Extraction

m4. Spanish and English Sentiment Extraction

m5. English Emotion recognition

m6. Audio Emotion recognition

m7. Spanish and English Emotion recognition

m8. Spanish Entity Extraction

m10. English Entity Extraction

m12. External English Topic Extraction

m13. Spanish Topic Extraction

m16. Suggestion mining

m20. Twitter media crawler

m21. Fusion

m22. Social Network Analysis

m25. Social semantic Knowledge Graph

m31. Analytics module “Kibi”

m32. YouTube Crawler

6. MixedEmotions Platform proprietary modules

m3. Italian and English Sentiment Extraction

m9. Italian and English Entity Extraction

m11. External Italian and English Topic Extraction

m15. External Italian and English Entity Linking

m17. Speech to text

m18. Machine translation

m23. Audio Emotion extraction

m24. Recommendation engine

m27. Video Emotion recognition

m28. Age estimation from audio

m29. Gender identification from audio

7. MixedEmotions Platform in use: MixedEmotions Project pilots.

7.1. Pilot 1: Social TV

7.2. Pilot 2: Brand Reputation Management

7.3. Pilot 3: Call Center

Executive summary

The present document is the final version of the architecture of the MixedEmotions platform. The starting point for this document was the previous deliverable, the initial version of the architecture. That initial architecture has been put into test in the months between. After this time, it was made apparent that the initial architecture was not addressing correctly the challenges of the project. Therefore, the architecture had to change.

The main challenge of the platform, which was not correctly addressed in the previous version of the document, was the integration of modules. The platform should be able to handle the use of numerous modules, each of them created individually by different partners and in different programming languages. That made the classic integration attempted in the previous version of the architecture an impossible task. Moreover, there was also the need for a light version of the platform, one that could be run in a single machine, for demo and testing purposes.

In order to solve these issues, a new approach has been taken, that is a microservice based architecture. This kind of architecture relies on smaller components, independent for the most part from each other, and which communicate via a REST interface. Moreover, those microservices are deployed using containers, which make them more independent and scalable.

Taking advantage of the characteristics of this type of architecture the MixedEmotions platform has now better “toolbox” capabilities, which means that the users will be able to use only the components they need and even add their own. The microservice architecture also provides the platform with Big Data capabilities, as this type of architecture allows the platform to increase horizontally with very little cost. It also allows a single machine mode.

In the new architecture, the focus has been put solely into the new modules. The support components such as databases are left for the user to choose and install, although they are still in the platform as recommended components. The idea is that a new user could test some modules independently, even in his own machine. Then, once he has tried the modules which could be relevant for him, he can shift to using them in a production environment, even in a scalable cluster of cloud machines.

In this final version, the main components of the platform are identified and their roles are conveniently established regarding the new microservices architecture.

1. MixedEmotions platform architecture

1.1. Platform Introduction

The MixedEmotions platform is a Big Data Toolbox for multilingual and multimodal emotion analysis. It can extract emotions from text, audio and video. However, it also has many other capabilities, such as sentiment analysis, social network analysis and knowledge graphs visualization among others.

The platform has been conceived as a toolbox which anyone can download and easily start using, with the processing of natural language data from text, audio and video, but also includes the ability of being used in a Big Data production environment. With its microservice architecture, the platform can be deployed in a cluster of machines and be scaled horizontally as needed.

The platform also offers a set of optional tools including data crawlers, linked data analyzers and visualization tools, not directly related to Natural Language Processing, but that can be very interesting for potential projects using the platform.

The platform is mostly composed of open source modules. However, some additional modules that have been developed and tested as part of the project will be made available with a proprietary license. They will be listed as part of the platform's components, as they have been developed for the use cases' purpose. Users interested in those modules will be able to purchase their license.

1.2. Platform architecture overview

The schema in Figure 1 provides a high-level specification of the architecture of the MixedEmotions platform, including the core toolbox for natural language processing, visualization tools, linked data analysis tools and recommended persistence engines.

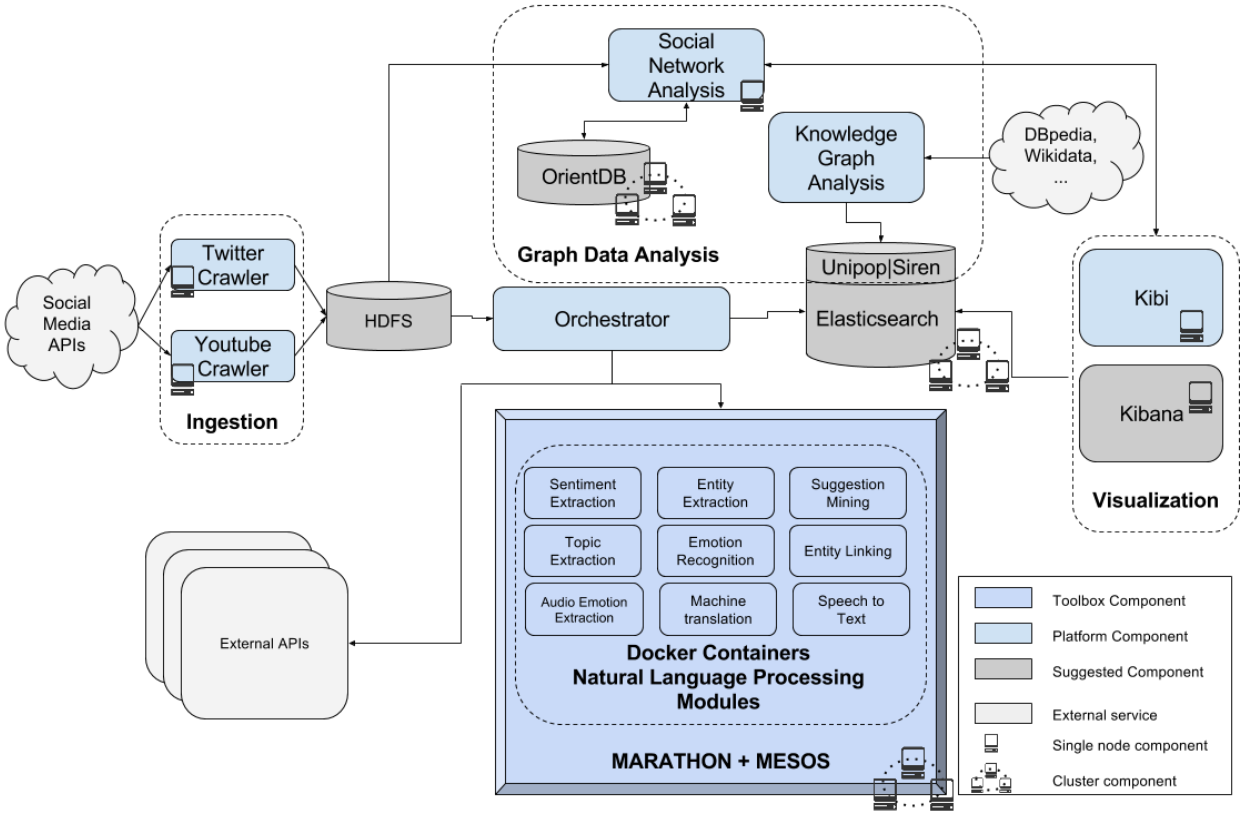

Figure 1. Full architecture of the MixedEmotions platform.

We distinguish the following elements of the MixedEmotions Big Data platform

1. **MixedEmotions platform Natural Language Processing modules**. These are the main components of the platform, and the ones that will be used in any of the modes the MixedEmotions platform can be used. Those Natural Language Processing modules are deployed as REST services inside a Docker Image. Docker containers from those images will be deployed in a Marathon and Mesos framework in the Big Data mode. More details about these in the Big Data processing section.
2. **Ingestion**. This part gathers modules that collect data from Social Network APIs, such as Twitter or YouTube. These data should be put into some persistence system. Depending on data volume, it could simply be a regular filesystem, a distributed filesystem such as HDFS, or some kind of database.
3. **Graph analysis**. The tools in this part are about social network analysis from Social Media data and knowledge graph. They need certain graph persistence systems to work, such as OrientDB or the Unipop plugin for Elasticsearch.
4. **Visualization and data discovery**. A visualization and data discovery module can be used to obtain graphical representations of the data. Kibi is an extension of Kibana that allows graph

representation and queries using linked data. Alternatively, if the user does not need this capability, regular Kibana is a simpler alternative. Both of these tools work through the reading of an Elasticsearch index.

5. **Orchestrator**. This is a software that manages the execution of various platform modules in a pipeline. It monitors the travel of the data along the platform, coordinating the execution of the modules specified by the user. As the modules work with a REST interface, it will be straightforward to integrate those modules with some external REST services. This orchestrator should be able to learn ip addresses of the Docker containers which contain the Natural Language Processing modules, by querying the Mesos DNS service.

1.3. MixedEmotions Platform as a Toolbox

In order for the MixedEmotions platform to be actually used and not put aside, it is important to reach a wide audience. For that purpose, it was a main objective of the architecture to be in the form of a flexible "toolbox" instead of a rigid platform. This way, the final user will be able to select only the software modules that are suited for its project needs, omitting the rest. For example some users might be really interested in processing tweets, thus needing the Twitter Crawler, the text processing modules and maybe the Kibi visualization tool, whereas others might want to use the Emotion Extraction from audio module, depending on their objectives.

Taking this approach further, and even though the Big Data aspect of the platform is fundamental to the project, there is also the option to offer the functionalities of MixedEmotions without the Big Data capabilities, for running them in a single machine. In this scenario, that we name "stand alone mode", the user will only need to have a Docker server so that he can download and use the Docker images of the desired modules from the platform. It could for example be useful for them to test the suitability of some modules in a project or to test the viability of custom modules.

This demonstrates the flexibility and versatility of the platform, shaped by the constraints imposed by such a multi-partnership project. This was made possible thanks to the integration of the MixedEmotions platform with external REST services and thanks the flexibility of the container technology.

The consortium tested this approach in the hackathon sessions in Galway (23-30 June 2016), as it was impossible to provide attendants with a cluster of machines each, which was a minimum requirement in the previous architecture of the platform.

1.4. Selection of technologies

The technologies selected for the implementation of the MixedEmotions platform are motivated by the lessons learnt after testing the initial version of the platform. Broadly, the requirements were the need for a scalable approach in order to process large volumes of data, as well as the management of the cluster on which the platform will be launched. In this section we address these prerequisites and specify the candidate technologies to satisfy them.

Microservices for processing

The MixedEmotions platform includes several modules for language processing. The main requirements were for the modules to be able to be written in different languages and developed by different companies. Also, the platform had to be able to handle Big Data processing. A microservice architecture was thus chosen as the best approach to address those two main issues.

A microservice architecture consists of small services that do independent tasks. Those services are usually deployed using containerization. Containerization is also known as lightweight virtualization and basically consists in packaging every service with a very barebone operative system. Then, those packages, known as images, are deployed as containers, which are very small virtual machines. Containers scalate greatly, as many of them can be deployed in a single machine or in many machines as needed. Finally, the containers are isolated from each other, which means components will never experience dependencies incompatibilities.

For implementing the containers, we chose Docker as the technology to use. Among the container technologies, it is the most lightweight and the easiest to use. It is indeed the most widespread tool of its kind at the moment, which will benefit the future adoption of the platform.

The microservice architecture needs other components to integrate those microservices. The main ones are usually discovery, orchestration, configuration and logging. Many "stacks" of technologies are currently available. For this project we will be using part of the Apache Mesos stack, mainly Mesos, Marathon and Mesos-DNS, Zookeeper, as they are open source, widely used and in the knowhow of Paradigma.

Scalability

These objectives suggest the need for a scalable platform capable of processing large volumes of data.

For this purpose, the microservice structure provided for the main "toolbox" of the platform is ideal, because it enables the capability of horizontal scaling. This means that if more processing power is

needed, it can easily be obtained by replicating the services or even by adding a new machine to the cluster.

Cluster management

The platform is then assumed to be scalable and it will be properly designed and developed to keep on this track. As a scalable software, it will run on a set of resources or machines that need to be managed. Additionally, it must be taken into account that several instances of the MixedEmotions platform could run on the same cluster. A specific software for the cluster management will be required in order achieve this functionality. It will be responsible for allocating the resources in the cluster for each process.

For the deployment of the three pilots of the MixedEmotions platform, Apache Mesos will be adopted as processing management. Apache Mesos allows us to abstract cluster resources (i.e., CPU, memory, storage and other compute resources) away from machines (physical or virtual).

Mesos is a distributed systems kernel, built using the same principles as the Linux kernel, only at a different level of abstraction. The Mesos kernel runs on every machine and provides applications (e.g., Hadoop, Spark, Elasticsearch) with API's for resource management and scheduling across entire distributed environments.

Distributed and scalable data processing

Having Mesos as a resource manager, means that the cluster can be used for distributed processing with Spark. This could be very interesting in some cases, for example if the input data volume is very high and it can be read from HDFS.

2. Big Data MixedEmotions platform

2.1. Introduction

The MixedEmotions platform was designed having in mind the processing of Big Data. This implies the capability of processing large amounts of data. In order to attain this capacity of processing, the approach is to scale horizontally, which means having more machines processing in parallel. There are many approaches to this. In the MixedEmotions platform the approach followed was a microservice architecture.

2.2. Microservice architecture

As stated before, a microservice architecture has as its main component microservices. They are independent services with a single purpose and they communicate via some network interface, usually using containers technology. However, for this kind of architecture to work, more components have to be present. First, there need to be a pipeline orchestrator that makes the corresponding calls to the services. It should function as a manager, that launches the services, checks their status and relaunches them if they are down. This manager may rely on a resource manager, which combines the resources available from all the machines and offers them to the manager, thus reducing the complexity the manager has to deal with. It is also indispensable to have a discovery service that informs the pipeline orchestrator of where each service is deployed. This kind of architecture can also have a log aggregation service and a distributed configuration service. That will not be necessary in this platform, because the service modules do not share configurations such as common databases credentials.

The components used in the MixedEmotions platform are: Mesos as resource manager, Marathon as services manager, Mesos-DNS as discovery service and the Spark or the Scala orchestrators as pipeline orchestrator.

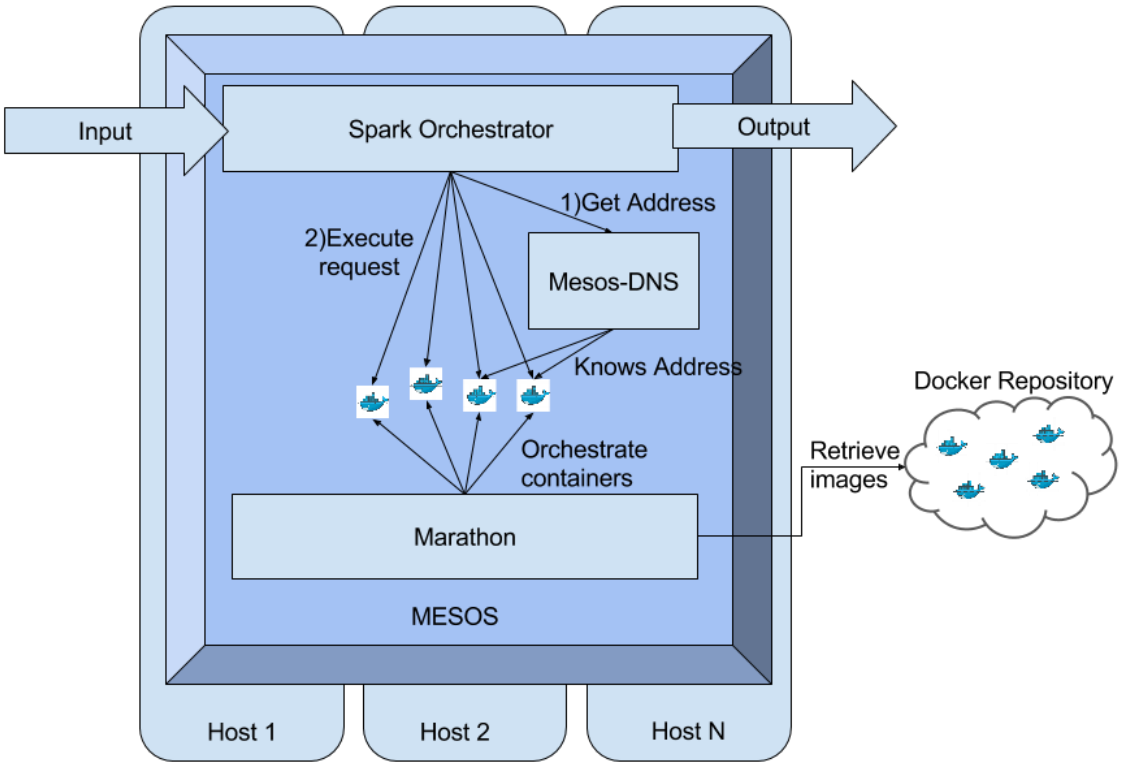

Figure 7. Big data service processing with Mesos and Docker

2.2.1. MixedEmotions Services as Docker containers

The most important part of the MixedEmotions platform are the processing modules. There are many of them, each one processing a different task, and having its own characteristics and requirements. Therefore, a great challenge of the MixedEmotions platform was to integrate all of them together. The usual chosen option is the integration of each library. However, with a Big Data architecture such an approach would involve a high effort to adapt all the libraries to the platform. After analysing the resources needed to integrate just a small sample of services, it was evident that the classical approach was not appropriate for the integration of these services.

For the integration of the platform modules, Docker has indeed been chosen. First, it allows to encapsulate each service in its own so-called “image”, thus removing the need of adding dependencies to the platform for each service as well as preventing potential conflicts. In addition, Docker, when combined with Mesos, allows for a more consistent level of parallelism, which will be explained in the next section. Another advantage of having Docker images is that processing modules are easily detachable, so that the toolbox aspect of the platform is further reinforced.

In order to enable the integration of the services, the only requirement is that Docker containers have to expose a REST application. Furthermore, to facilitate the integration, the services will respond using JSON-LD as explained in appendix A.

2.2.2. Mesos as MixedEmotions resource manager

Mesos is an open source resource manager that, in a few words, enables the use of a machine cluster as a single machine, as it distributes memory and CPU load evenly among the machines of the cluster. It is the foundation of many PaaS (Platform as a Service) offerings. Mesos makes "offerings" of CPU cycles and RAM memory and automatically balances load between the servers of the cluster.

In the MixedEmotions platform, Mesos will handle the execution of the different services, using Marathon as a container manager and Mesos-DNS as discovery service. Additionally, having Mesos will let the users use Spark if they will to do so.

2.2.3. Marathon as MixedEmotions containers manager

Marathon is another tool that is used for executing services in Mesos. With Marathon, the user just has to define the number of instances, memory and cpu allocated for a service. Marathon will use Mesos to execute those services in the cluster, using the machines that have more resources. Also, if one of the instances fails, Marathon will automatically start a new instance, making for an environment very fault-tolerant. In case of heavy load, the user can scale up the number of services using the Marathon interface.

Marathon also integrates seamlessly with Docker. The Marathon service configuration allows to use a Docker image URL in a repository as definition of a service. If that is the case, Marathon will download the image and start the corresponding number of instance. There is only one limitation of this approach, and that is that it is not possible to know the ip and port of the service instance beforehand.

2.2.4. Mesos-DNS as MixedEmotions discovery service

For overcoming that gap, the MixedEmotions Platform uses Mesos-DNS, which is a service that can be used as a DNS and a REST service that returns the ip and port of the service instances, as well as serving as a simple balancer. Mesos-DNS is also executed as a Marathon service.

Therefore, the execution of a MixedEmotions Platform pipeline will consist of an execution of the Spark Orchestrator, which is running in Spark. This will ask Mesos-DNS for the ip and port of an instance of the service to be executed whenever an item is to be processed. Then the main orchestrator will send the item to the corresponding service, which is running in Mesos and orchestrated by Marathon.

2.2.5. MixedEmotions pipeline orchestrator

Finally, there is the need to have some component that uses all the services deployed in the microservices architecture. That will be the pipeline orchestrator.

In most cases, the final user will be developing their own orchestrator according to their needs. However, for demonstration purposes, a "default" pipeline orchestrator will be provided.

These orchestrators will not need to delve into the nuances of the Microservice architecture, as they only execute REST requests to the corresponding microservices and parse the results. However, in order to do that, they need the capacity to make queries to Mesos-DNS, the discovery service.

3. Platform Components

In this section, we provide a more detailed description on the elements composing the MixedEmotions platform.

3.1. Processing modules

These are the Natural Language Processing modules in the MixedEmotions Platform. As the platform comprises multiple modules, from different partners and different technologies, is important to devise a way to integrate them together.

The integration paradigm selected for the platform is a Microservice architecture approach, as introduced in the previous section. Further explanation will be found in section 3. In broader terms a microservice architecture is one in which every functionality is provided by an independent service with a single functionality. Those services are usually deployed using containerization. In this project the microservices are deployed using Docker containers. Also, those services use a generic standard of communication, usually HTTP with a REST interface.

In general, NLP functions are expected to accept some text, audio or video, which will be analyzed for the computation of a result related to a given context or problem. The functions addressed in MixedEmotions are the following ones:

- Emotion recognition: the emotions expressed in the data.
- Topic extraction: the topic areas that the data corresponds to.
- Entity/concept extraction: the entities identified in the data.
- Sentiment extraction: the sentiments expressed in the data.

- Suggestion mining: reviewers' recommendations for the brand expressing ways to improve.
- Speech transcription: converts pieces of speech (audio) to text.
- Machine translation: translates texts from Czech, Spain, German, Italian into English.

To integrate all the different modules in the platform, and let the platform be also a toolbox in which the services can be used independently, the modules in the platform will be composed of modular services (hence modules) that are also independently usable. To further facilitate the integration, all of those services will have a REST interface and will return a JSON as output.

A common format for data integration has been defined. This format is JSON-LD (JSON for Linked Data) which is an implementation of NIF for JSON. More details can be found on Appendix A.

Regarding the integration and deployment technique used for the services, there will be two kind of services:

- Docker services: these services will be more closely integrated in the Big Data Platform. More details can be found on section 3.4.
- External REST Services. Finally, another approach to provide the functionality of a module is by means of a REST service. In this case, the MixedEmotions platform must provide the mechanism to call this service and receive its response to the requested analysis of the data.

3.2 Ingestion

These modules are responsible for collecting data from the internet. In the MixedEmotions platform, ingestion modules are the following ones:

- Twitter crawler
- YouTube crawler

3.2.1. Twitter Crawler

The Twitter crawler collects tweets according to the selected keywords. In MixedEmotions, this component is developed in Python and can be deployed as a web service. As such, and because the limitation for retrieving tweets lies in the limited quota of the Twitter API and not in the processing capabilities, this element is to be deployed as a single instance.

3.2.2. YouTube Crawler

The YouTube crawler downloads YouTube videos along with comments and other information attached to videos. In MixedEmotions, this component is developed in Python. Because of the limitation of the Google API, this module is to be deployed as a single instance.

3.3. Graph Analysis: Social Semantic knowledge graph and network analysis

The functionality that provides the social semantic knowledge graph in the MixedEmotions platform will be given by a set of processes which perform the following operations:

- On the knowledge graph side, they extract useful pieces of data from major datasets (e.g. Wikidata, Dbpedia). Then, those are transformed using various approaches like interlinking or ontology matching into a format that can then be used for enhancing the queries and functionalities.

- On the social network side, a social graph will be constructed based on the information extracted from social networks. In addition to the plain data extracted from the network, the graph will be enriched with user and content "metrics", which give further insight on social context. These metrics include calculated parameters such as "centrality", "betweenness" and "influence". Figure 5 depicts the architecture of the modules for social network analysis in the platform.

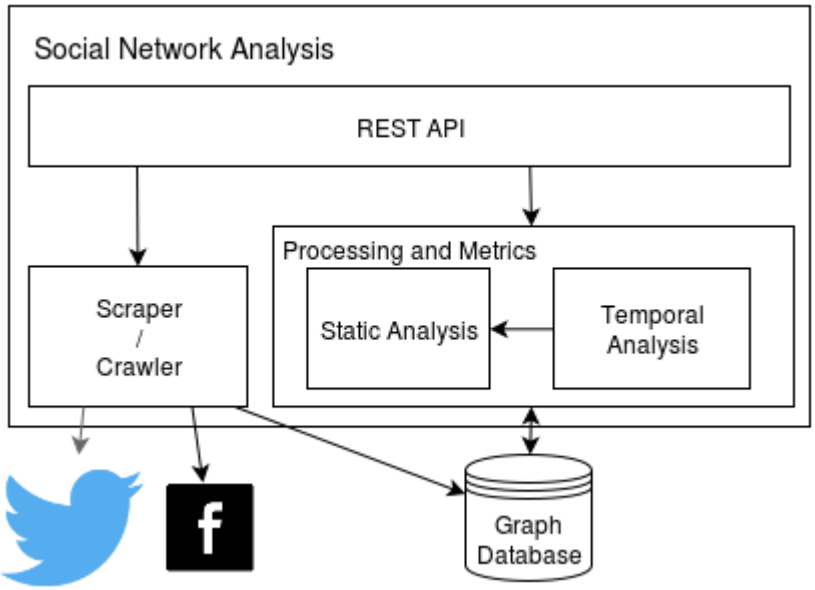

Figure 5. Social Network Analysis architecture

3.4. Analytics and Visualization

In order to be able to appreciate and optimize the results based on the use of the advanced linguistic technologies it is fundamental to be able to represent and visualize the processed data, so it was very important to have a visualization tool included in the platform. To this end, the platform includes a state of the art streaming relational analytics system, Kibi, developed as a fork of Kibana, a visualization tool

for Elasticsearch. Kibi improves Kibana by including the capabilities of representing and using Knowledge graphs for data filtering.

3.5. Pipeline orchestrator

The pipeline orchestrator will be the element responsible for getting the configuration of the MixedEmotions platform from the user and running the software according to the specified requirements. It must take into account the order of the modules in the platform.

The MixedEmotions platform includes an example pipeline orchestrator, that can be used as is or as an starting point for developing a custom orchestrator. This orchestrator reads configuration files in a specified format and can work with REST services or with Docker services deployed with Marathon in a Mesos cluster. In order to do so, it will ask Mesos-DNS about where those services are and will perform some balancing.

This orchestrator can be used also in Standalone mode, where Docker services are deployed by themselves. When doing so, the can be accessed as a regular REST service, the only requirement is to know their ip and port.

4. Distribution of the MixedEmotions platform

The open source modules of the MixedEmotions platform will be available in the MixedEmotions repository in github at https://github.com/MixedEmotions. The platform documentation, detailing the installation and operation instructions will be also made available in that platform at https://github.com/MixedEmotions/MixedEmotions.

Docker images for open source processing modules of the MixedEmotions platform will be available in the Dockerhub repository of images, along with licensing and operation instructions at https://hub.docker.com/u/mixedemotions/.

Most open source modules will be present both in Github and Dockerhub, although there are a couple that do not have a Docker Image readily available, although they provide a Dockerfile and clear instructions on how to create one.

Standalone mode

It is possible to use MixedEmotions in a non Big Data environment, such as in a single machine. In order to do so, the user will just need to download and install a Docker server. Instructions for that can be found in the Docker web page. Then, selected module images can be downloaded from Dockerhub and run independently. As every module uses a REST interface, documented in their source and Docker pages, the integration of these modules should be straightforward enough.

Big Data mode

When using the MixedEmotions platform for Big Data analysis, the main component that orchestrates the Docker images is Mesos. Docker support should be enabled as explained in the Mesos web page. Marathon should be also installed and run on Mesos.

Once Marathon is running, the Docker modules would start by sending Marathon the proper instructions via POST requests. The configuration files and instructions for doing this will also be provided in the MixedEmotions Github repository.

Proprietary modules

A list of proprietary modules will be included in the documentation of the platform, including contact information for users interested in them.

5. MixedEmotions Platform Open Source modules

Table 1 includes the list, description and deployment strategy for the MixedEmotions open source modules. The integration column details the module integration strategy (from those defined in Appendix B). The modality defines the type of input the module needs in order to work. The integration column defines whether the module is available in Dockerhub or only on github. Unless specified, github sources will be available on https://github.com/MixedEmotions, and Docker images will be accessible in Dockerhub on https://hub.docker.com/u/mixedemotions/.

Id	Functionality	Modality	Language	Source	Integration
m1	Sentiment Extraction	Text	EN	Github	Dockerhub
m2	Sentiment Extraction	Text	EN, CS	Github	Github source
m4	Sentiment Extraction	Text	EN, ES	Github	Dockerhub
m5	Emotion recognition	Text	EN	Github	Dockerhub
m6	Emotion recognition	Audio	EN	Github	Dockerhub
m7	Emotion recognition	Text	EN, ES, Multiple	Github	Dockerhub
m8	Entity Extraction	Text	ES	Github	Dockerhub
m10	Entity Extraction and Linking	Text	EN	Github	Dockerhub
m12	Topic Extraction	Text	EN		
m13	Topic Extraction	Text	ES	Github	Dockerhub
m16	Suggestion mining	Text	EN	Github	Dockerhub
m20	Twitter media crawler	Text	n/a	Github	Github source
m21	Fusion	Text/Audio/Video	n/a	Github	Github source
m22	Social Network Analysis	graph	n/a	Github	Dockerhub
m25	Social semantic Knowledge graph	graph	n/a	Github	Dockerhub
m28	Analytics module “Kibi”	-	-	Github	Standalone, Github source
m32	Youtube crawler	Text/Video	n/a	Github	Github source

Table 1. List of modules included in the MixedEmotions platform.

m1. English Sentiment Extraction

This module predicts the polarity for the given sentence. The module uses Neural Network based classification architectures, Long Short Term Memory and Convolutional Neural Networks. The predicted polarity belongs to the classes "positive", "negative" and "neutral". The module is written in Python 3.5 and uses machine learning libraries like scikit-learn[1] and Keras[2]. This module can be found in GitHub:
https://github.com/MixedEmotions/NUIG-suggestion
and in Docker hub:
https://hub.docker.com/r/mixedemotions/01_sentiment_analysis_nuig/.

m2. Czech and English Sentiment Extraction

The sentiment analysis module from BUT uses the Stanford CoreNLP[3], the LingPipe[4] and the VADER Sentiment Analysis[5] tools. It utilizes various classification algorithms including SVM (Support Vector Machine) and Naive Bayes classifier.

The module performs sentiment analysis of tweets as well as general texts in English and Czech language. Having a suitable dataset, models for different domains and languages can be simply trained.
Since most of the tools are implemented in Java, the module is written in Java as well.
This module can be found in github:
https://github.com/MixedEmotions/but_sentiment .

m4. Spanish and English Sentiment Extraction

Spanish and English Sentiment Extraction written in python, using the Senpy framework[6]. As any other Senpy application, the module consists of the senpy core server and a set of plugins that provide the logic for each analysis. There is an official public docker image for the senpy server[7], and several images for different sets of plugins. The dockerfile and image for the open source Senpy plugins are available[8]. The proprietary plugins have been integrated in other private images built using a similar method and cannot be released on the dockerhub due to licensing issues.
This module can be found in github:
https://github.com/MixedEmotions/senpy-community.
It has a Docker image available:
https://hub.docker.com/r/mixedemotions/senpy-community/ .

[1] scikit-learn.org
[2] https://keras.io/
[3] https://stanfordnlp.github.io/CoreNLP/
[4] http://alias-i.com/
[5] https://github.com/cjhutto/vaderSentiment
[6] http://github.com/gsi-upm/senpy
[7] https://hub.docker.com/r/gsiupm/senpy/
[8] https://github.com/gsi-upm/senpy-plugins-community

m5. English Emotion recognition

Categorical emotion recognition module written in Java using the SVM classifier from the Weka machine learning framework. Two models are provided trained on different data sets: annotated news headline data from SemEval 2007 Task 14 (Affective Text)[9] and the Twitter Emotion Corpus consisting of tweets with emotion hash tags as noisy labels. The models produce a single emotion label from Ekman's six basic emotions (joy, fear, disgust, anger, surprise, sadness) for each submitted text. It is currently available via a RESTful web-service.
It has a Docker image available:
https://hub.docker.com/r/mixedemotions/05_emotion_hashtags_nuig/ .

m6. Audio Emotion recognition

Audio emotion recognition module written in C++ and Java, through RESTful web-service. The recognition system is based on Bag-of-Audio-Word models for arousal and valence [1][10] trained on RECOLA french database. Moreover, this module can be used for other languages since audio signals are not restricted to certain languages.
This module has the capability to integrate an automatic speech recognition module, in order to segment audio data into sentence-level chunks and recognise emotions for each sentence. To embed an external ASR you should create three .sh files inside *platform_audio_er* folder:
asr_upload.sh: with the first parameter denoting the location of the audio file and the output should have the following fields in the form of:
{"result":{"info":{"name":"FILENAME.wav"}}}
asr_run.sh: with the first parameters as FILENAME.wav. The output should have the following fields:
{"result":{"info":{"id":"ID","state":"waiting"}}}
checkpending.sh: with the parameter ID and returns:
{"result":{"info":{"id":"ID","state":"waiting"}}}
OR
{"result":{"info":{"id":"ID","state":"finished"}}}
Once ASR completed its task by running again 'asr_run.sh FILENAME.wav' the final output is in the form of:

```
{"result":{"one_best_result":{"segmentation":[
        {"start":1600000,"end":4200000,"word":"<s>"},
        {"start":4200000,"end":11100000,"word":"I"},
        {"start":11100000,"end":14000000,"word":"AM"},
        {"start":14000000,"end":19100000,"word":"FINE"},
        {"start":19100000,"end":22800000,"word":"</s>"}]}}}
```

Where <s> denotes 'silence', </s> is dot (.), and start/end are 0.0000001s → 1600000 = 0.16s
The module source code can be found in:
https://github.com/MixedEmotions/up_emotions_audio.

[9] Bordea, G., Buitelaar, P., and Polajnar, T. (2013). Domain-independent term extraction through domain modelling. In the 10th International Conference on Terminology and Artificial Intelligence (TIA 2013), Paris, France.
[10] Schmitt, M., Ringeval, F., & Schuller, B. (2016). At the Border of Acoustics and Linguistics: Bag-of-Audio-Words for the Recognition of Emotions in Speech. Proc. INTERSPEECH. San Francisco, USA.

It has a Docker image available:
https://hub.docker.com/r/mixedemotions/06_audioanalysis_up/ .

m7. Spanish and English Emotion recognition

Python module for emotion extraction from text, which exposes a REST/NIF API via the Senpy framework[11] framework. Users may choose between two different algorithms to calculate emotions. The first one uses a combination of the ANEW lexicon for recognition (using the <Valence, Arousal, Dominance> model), and a mapping to Ekman's six emotion categories using pre-calculated centroids. The second algorithm relies on the WordNet-Affect lexicon to annotate text with one of Ekman's categories. The module is highly modular and configurable, which allows for the addition of new languages to the module by adding WordNets in other languages. Machine translated WordNet lexicons are available for all 23 official European languages providing emotion recognition capabilities for those languages with the WordNet-Affect based algorithm. Some algorithms have a commercial license and will be deployed in a standalone service and not included in the Dockerhub image. The open source code can be found at:
https://github.com/gsi-upm/senpy.

m8. Spanish Entity Extraction

Entity extraction module written in Python. Exposes a REST interface which returns an array with the detected entities. The entities are based on titles from dbpedia articles in 2012. The user is expected to change or to expand those entities regarding their use case.
The source code can be found in:
https://github.com/MixedEmotions/08_entity_extraction_es .
It has a Docker image available at:
https://hub.docker.com/r/mixedemotions/08_entity_extraction_pt/ .

m10. English Entity Extraction

Java entity extraction module written in Java 8. It requires a trained model to identify the named entities, and an indexed over Wikipedia lexicons to perform entity disambiguation. Lexicons are extracted from Wikipedia snapshot from Oct, 2014. This module is provided as a Jar as well as in docker. It takes a text input in Json and returns a Json file which contains extracted entities, their types and Wikipedia link.
The source code can be found in:
https://github.com/MixedEmotions/entity-linking .
It has a Docker image available at:
https://hub.docker.com/r/mixedemotions/10_entity_linking_nuig/ .

m12. External English Topic Extraction

Topic extraction named Saffron, written in Java. It is based on the use of an automatically constructed domain model to measure the coherence of a candidate term within a domain. See more in [2]
The module is in progress to be made available as open source. The Technology Transfer Office at NUI Galway has been contacted in regard of an open source release of the software. For more information,

[11] http://github.com/gsi-upm/senpy

please visit http://saffron.insight-centre.org/.

m13. Spanish Topic Extraction

Spanish Topic extraction implemented in Scala Spark. It uses a manual taxonomy that is to be defined by the final user. Returns a list of topics detected.
The source code can be found in:
https://github.com/MixedEmotions/13_topic_extraction .
It has a Docker image available at:
https://hub.docker.com/r/mixedemotions/13_topic_extraction_spanish/

m16. Suggestion mining

This module checks if the given sentence contains a suggestion. The module needs a model data that has to be distributed on every node. In the background, this module uses neural network based classifiers which classifies each sentence of a given text into *suggestion* and *non-suggestion* classes.
The source code can be found in:
https://github.com/MixedEmotions/NUIG-suggestion .
It has a Docker image available at:
https://hub.docker.com/r/sapnanegi/16_suggestion_mining_nuig/ .

m20. Twitter media crawler

The Twitter media crawler is a Python application that gets data from Twitter via streaming API. It returns public statuses that match one or more keywords specified by user. The default access level allows up to 400 track keywords. The data is stored in JSON format and can be accessed via API.

There are two types of API requests that can be used to get the data files, the interval and the timestamp. The interval service provides the data corresponding to the interval between two given timestamps and the timestamp service provides the data newer than the given timestamp.

To use this module a new Twitter account has to be registered and the credentials has to be provided to the application. See more details in the module manual at https://github.com/MixedEmotions/twitter_crawlers

m21. Fusion

Multimodal analysis which combines the results of emotion/sentiment recognizers from video, audio and transcription to yield higher recognition performance. This module has been embedded into the audio module repository (but it is independent):
https://github.com/MixedEmotions/up_emotions_audio
After running the module, see http://localhost:8888/er/general for more information.

m22. Social Network Analysis

This module processes social network data, enriching it with contextual information (as described in D4.9 and D4.10). It includes a data crawler.

The service is deployed as a set of Docker containers linked together. In particular, there is: a Scaner container, a celery container with Scaner tasks, a redis container that acts as a message broker between the previous two containers, and an orientdb container.
The source code can be found at: https://github.com/gsi-upm/scaner.

m25. Social semantic Knowledge Graph

This application is meant to be used as a last step after a combination of prior modules. After running the entity extraction module on the data, the recognized entities are passed to this module. Each entity will be match to DBpedia, and its type (Person, Organisation, Event, Location) will be extracted alongside with expanded information surrounding it. All of this will eventually be gathered and combined under the shape of a knowledge graph, given as output.
The source code can be found in:
https://github.com/MixedEmotions/knowledge-graph .
It has a Docker image available:
https://hub.docker.com/r/mixedemotions/knowledge-graph-creator/ .

m31. Analytics module "Kibi"

Kibi is a visualization tool based on Kibana that is a full-fledged project. Its core components are open source, although it also has an enterprise proprietary edition with more functionalities. Kibi has been developed to provide cross Structured and Unstructured data analytics and discovery. In this way, one can browse and discover patterns both in the unstructured text and in the emotional or other information which have been extracted algorithmically or provided by connected/linked data. The source for the open source components of this project can be found at: https://github.com/sirensolutions/kibi. More information is available at https://siren.solutions/kibi/.

m32. YouTube Crawler

The YouTube Crawler is a Python application that collects information about activities and comments belonging to YouTube videos. The application can download video and audio data as well. The module utilizes the YouTube Data API to download the data. The data is stored as JSON files.

The module has two modes of operation. The first one allows users to download all information from a particular YouTube channel and the second one is a keyword based search mode. The search mode can search for all videos related to a keyword given. The module is also able to update the previously downloaded data (e.g. comments or likes) .

Downloading of data from YouTube is limited to 50,000,000 requests per day (operation write needs over 50 operations, read 10-20 operations). Due to the YouTube Data API restrictions, the search mode can retrieve the data for 500 videos in maximum at once.

To use this module a new Google account has to be registered and the credentials has to be provided to the application. See more details in the module manual at:
https://github.com/MixedEmotions/youtube_downloader .

6. MixedEmotions Platform proprietary modules

As stated earlier, is important that the MixedEmotions platform is as versatile as possible. To that end, the MixedEmotions platform permits the use of external services to substitute the modules it has, for example, if the user has its own sentiment analysis tool.

The integration depends on the orchestrators. For the default orchestrator, the requirement is that the external services expose a REST. Usually, that interface that admits a POST containing a JSON-LD and returns a JSON-LD adding new processed data (as defined in Appendix A).

Taking advantage of this platform feature, some proprietary modules have been developed in conjunction with the platform and are going to be used in the pilots. They are to be included in the platform for the users willing to purchase their license, as they have been used along the open source modules. These modules will have professional support and are currently used in the platform.

Id	Functionality	Modality	Language	Integration
m3	Sentiment Extraction	Text	EN, IT	Service
m9	Entity Extraction	Text	IT, EN	Service
m11	Topic Extraction	Text	IT, EN	Service
m15	Entity Linking	Text	IT, EN	Service
m17	Speech to text	Audio	EN	Service
m18	Machine translation	Text	CS, ES, DE, IT	Service
m23	Emotion recognition	Audio	DE, EN, CS	Service
m24	Recommendation engine	Text	EN	Service
m27	Video Emotion recognition	Video	n/a	Service
m29	Age estimation from audio	Audio	-	Service
m30	Gender identification from audio	Audio	-	Service

m3. Italian and English Sentiment Extraction

External service provided with a REST interface. More information can be found in https://developer.cogitoapi.com/docs.

m9. Italian and English Entity Extraction

External entity extraction module. Accessible via REST interface. More information can be found in https://developer.cogitoapi.com/docs.

m11. External Italian and English Topic Extraction

External Topic Extraction module. Accessible via REST interface. More information can be found in https://developer.cogitoapi.com/docs.

m15. External Italian and English Entity Linking

External Entity linking module. Accessible via REST interface. More information can be found in https://developer.cogitoapi.com/docs.

m17. Speech to text

Speech Transcription converts speech signals into plain text. After speech to text conversion, the text can be easily read, edited, searched, processed by text-based data mining tools or archived. This model is trained for English and is accessible through REST interface.

The engine is specially optimized for noisy and colloquial speech. It is based on state-of-the-art techniques for acoustic modeling, including discriminative training, neural network-based features, and speaker and channel adaptation techniques. It is compatible with a wide range of audio sources: GSM/CDMA, 3G, and VoIP-based, Live Broadcast and Recorded speech with emphasis on spontaneous telephony speech. It contains dictionary with 50 thousand words and on our internal test set reaches 72% Word Error Rate.

Phonexia REST server is written in C++ and based on open-source library Poco. It uses asynchronous processing, because the system is relatively slow (it process 100 seconds of audio in 60 seconds of CPU processing time). The system itself is scalable and right now it is set to use 4 cores.

This module can be used in conjunction with the m6 module to separate the video into sentences.
Contact info@phonexia.com for more information about this.

m18. Machine translation

This is a module that can translate from a variety of languages to English. The target languages are German, Italian, Spanish and Czech.

The translation module takes as input a text in a source language and translates it into a target language. Currently, the module covers translations between English and several foreign languages, i.e. German, Spanish, Italian and Czech. The translation models, necessary to translate textual information, were

trained on more than 10 million parallel sentences from different domains, e.g. Europarl (European commission talks), DGT (translation memories of European union law) or Subtitles (subtitles of movies).

The module, which is called through a REST service, uses the widely used phrase based statistical machine translation methodology within the MOSES[12] decoder.

Example for German into English:
http://server1.nlp.insight-centre.org/cgi-bin/ME_transcript_translation_api.cgi?pair=de_en&type=reduced&nbest=10&source="Die Schlepper des Flüchtlings-Lkw auf der A4 sind gefasst. Die drei Männer werden im Nachbarland Ungarn festgenommen."

or Spanish into English
http://server1.nlp.insight-centre.org/cgi-bin/ME_transcript_translation_api.cgi?pair=es_en&type=filtered&nbest=10&source="Más de 70 cuerpos hallados en un camión abandonado en Austria"

Arguments for the API:
-nbest (nbest) = producing n-best translation options, integer value requested
-translation model type (type) = always filtered, for quicker translation approach
-translation direction code (pair)
-source sentence (source)

Translation Direction Code	Description
de_en / en_de	translating from German > English / English > German
it_en / en_it	translating from Italian > English / English > Italian
es_en / en_es	translating from Spanish > English / English > Spanish
cs_en / en_cs	translating from Czech > English / English > Czech

json keys/structure:

key	value
source	source sentence
best_translation	translation of source sentence

[12] http://www.statmt.org/moses/

time	time needed to get the translation
tm_type	translation model type
n_best	list of alternative translations
	<table><tr><td>key</td><td>value</td></tr><tr><td>alternative translation</td><td>probability</td></tr></table>

json output (called from a code):

{
"tm_type": "filtered",
"source": "Die Schlepper des Flüchtlings-Lkw auf der A4 sind gefasst. Die drei Männer werden im Nachbarland Ungarn festgenommen.",
"time": "10 wallclock secs (0.01 usr 0.00 sys + 4.78 cusr 0.43 csys = 5.22 CPU)",
"n_best": {
"the traffickers of refugee @-@ lorries on the a4 are . the three men will be in the neighbouring country of hungary arrested . ": "-109.781",
"the traffickers of refugee @-@ lorries on the a4 are . the three men are arrested in the neighbouring country of hungary . ": "-109.73",
"the traffickers of refugee @-@ lorries on the a4 are . the three men will be arrested in the neighbouring country of hungary . ": "-109.73",
"the traffickers of refugee @-@ truck on the a4 are . the three men are arrested in the neighbouring country of hungary . ": "-109.746",
"the traffickers of refugee @-@ truck on the a4 are . the three men will be arrested in the neighbouring country of hungary . ": "-109.745"
},
"best_translation": "the traffickers of refugee-lorries on the a4 are. the three men will be arrested in the neighbouring country of hungary."
}

Contact http://nlp.insight-centre.org/contact/ for more information on this.

m23. Audio Emotion extraction

Frame level emotion extraction from bottleneck features from audio in English, German and Czech. Contact info@phonexia.com for more information.

m24. Recommendation engine

This module recommends videos, using data extracted from comments from users.

During watching a video, in order to improve the number of videos watched per user, several slots with recommended video will be shown on Apple TV GUI.

The Recommendation Engine (RE) is based on two main things:

1. User behavior. User is represented by an anonymous ID USER and the following related data :
    - List of video/audio items watched by the ID USER,
    - List of the percentages in relation to the entire video
    - List of feedback (Like/UnLike), Emotions (disgusted, fearful, sad, angry, joyful, surprised)

   The properties used to predict the best videos for a given user at a particular moment is divided into:
    - Historical profile: from the info collected in the user's history.
    - Session's profile: from the user behavior in the current session
2. Video characteristics. This could be represented by a set of information already existing in the video database of Deutsche Welle enriched by the semantic tagging provided by semantic analysis of Expert System.

The RE needs:

- A large amount of data on user behavior. This is important to have a story of each user about its behavior.
- Feedback from users about the response to recommendations provided to allow the RE to learn and improve in an automatic way the quality of recommendation.

Result

- Provided that both user behaviour data as well as feedback on given recommendations can be collected, the RE is expected to give quickly improving results with the help of automatic machine learning mechanisms.

Input (ID A/V, ID USER)
Output (list of lists of ID A/V recommended)
Contact http://www.expertsystem.com/ for more info.

m27. Video Emotion recognition

This module extracts per-frame emotion information from video, regardless of language. Additionally it provides estimated age, gender, rough gaze direction, and unique person identities within a video. The contact person for this module is Michal Hradiš (ihradis@fit.vutbr.cz).

m28. Age estimation from audio

This module estimates the age from an audio, regardless of language. Contact info@phonexia.com for more information.

m29. Gender identification from audio

This module identifies the gender of a speaker, regardless of language. Contact info@phonexia.com for more information.

7. MixedEmotions Platform in use: MixedEmotions Project pilots.

The MixedEmotions Platform is being developed in the context of the Pilot scenarios, which are described in detail in deliverable D2.2. Below we present only a brief summary of each Pilot and the modules used in each.

7.1. Pilot 1: Social TV

Pilot 1 provides a real-time event monitoring in particular (but not only) in the context of the broadcast industry. The business case will provide a real-time monitoring support tool for journalists and anchor men which will give them an additional instrument for the management of long interviews and talk shows by regularly checking sentiments and emotions of trending topics on social media platforms.
Additionally, a second scenario for Social TV was defined that enables users to access a recommendation system based on emotion analysis provided by MixedEmotions. The Recommendation Engine (RE) is based on:

1. the assumption that TV users do not necessarily look for content with similar emotional characteristics.
2. the mood management theory: TV users are in principle hedonistic and aim for optimising their mood.
3. a dual process theory of media entertainment:

a. Hedonistic gratification: Enjoyment (a purely pleasurable experience) → "joyful"
b. Eudaimonic gratification: Appreciation (a meaningful/ valuable experience associated with mixed emotions) → "intriguing"

Modules usage.
Pilot 1 exploits the following modules:

1. External CogitoAPI service, gathering together M3, M9, M11 and M15. Pilot 1 analyses video teasers and rss feeds for extracting entities and topics, that are used for searching tweets.
2. M4 and M7 for analysing tweets.
3. External Recommendation Engine, M24, for recommending videos.
4. Social Semantic Knowledge Graph, M25, for exploring additional information regarding extracted entities (by M9 and M15).
5. Emotion extractors M23 and M27, for analysing audio and images of a video.

7.2. Pilot 2: Brand Reputation Management

The objective of the Brand Reputation Management is to obtain information about the position of a certain brand. For that objective, this pilot will obtain data for that brand in some social media and

newsfeeds and analyse it. The analysis will include sentiment extraction, emotion extraction, suggestion mining, concept extraction and topic extraction. The results will be presented in a web dashboard.

As for the data extraction, the pilot will use the Twitter media crawler (m20) and the Youtube crawler (m32). The video analysis will need the Emotion Recognition from video (m27) and the Emotion Recognition from audio (m6) modules. In combination to the latter, the Speech to Text module (m17), a proprietary one, will be accessed as a service.

Concerning the processing of texts, the pilot will make use of the Sentiment Extraction (m4) and the Emotion extraction (m7) modules based on Senpy, although it will use the service version that hosts the proprietary algorithms. Also, to get the most information off the text, it will make use of the following modules: Spanish Entity extraction (m8), the English Entity Linking (m10), English and Spanish Topic detection (m12, m13) and Suggestion mining (m16).

The pilot will also analyze the network of retrieved tweets using the Social Network Analysis Scaner (m22) to trace influential users and emotion propagation. The pilot plans to use the Social Semantic Knowledge Graph module (m25) as well, in order to extract more insight from the data.

Everything will be analyzed and put together using the visualization tool of the platform Kibi (m31).

7.3. Pilot 3: Call Center

The objective of this Pilot is to utilize acoustic emotion recognition from the platform. The obtained data will help with rating of recordings in Contact Centers. Together with other parameters that are already use (especially cross-talks, speech speed, speaker turn count, key-word spotting for emotion words) it will help to detect problematic parts in recordings and scripts or identify challenging topics.
Functionalities, which can help identify emotions in the speech (recordings) and analyze these parts are important for Contact Centers because these are the key moments of unsuccessful calls. Supervisors / quality managers could use the information to improve results and increase success (weaker agents in this area; verbal expressions and phrases that cause these emotions; also the phrases which are useful; in a real time notice for agents about incorrect process which will help them handle the situation and so on).

This pilot currently works with audio data and will mainly use Emotion Recognition from audio modules (m6, m23) and Speech to Text module (m17). We are still experimenting on using the Sentiment Extraction modules (m1, m2, m4) on the top of Speech to Text module. In addition we are also using information from Age and Gender recognition modules (m29, m30) to condition other outputs to offer analysis dependent on these modalities.

7.4. Modules usage

The following table summarizes the use of the Platform components within the Pilots.

		PILOT 1	PILOT 2	PILOT 3
m1	English Sentiment extraction			
m2	Czech and English Sentiment extraction			
m3	External Italian and English Sentiment extraction			
m4	Spanish and English Sentiment extraction	X	X	
m5	Textual English and Czech Emotion extraction			
m6	Audio Emotion extraction from Audio (EN, DE, CZ)		X	X
m7	Textual Spanish and English Emotion Extraction	X	X	
m8	Spanish Entity extraction		X	
m9	External Italian and English Entity extraction	X		
m10	English Entity extraction		X	
m11	External Italian and English Topic extraction	X		
m12	English Topic extraction		X	
m13	Spanish Topic extraction		X	
m15	External Italian Entity linking	X		
m16	English suggestion mining		X	
m17	Speech transcription			X
m18	Machine translation			X
m20	Twitter Crawler		X	
m21	Fusion			X
m22	Social Network Analysis		X	
m23	Emotion extraction from Audio (EN, DE, CZ)	X		X
m24	Recommendation Engine	X		
m25	Social Semantic Knowledge Graph	X	X	
m27	Emotion recognition from video	X	X	
m29	Age estimation from audio			X
m30	Gender identification from audio			X

m31	Analytics module Kibi		X	
m32	YouTube Crawler		X	

8. Conclusions

For the second and final version of the MixedEmotions platform architecture, a major change has been made and that is the inclusion of a microservices paradigm approach for handling the internal modules. This has been necessary as it was evident that modules integration within the platform was an issue that was greatly growing with the inclusion of new modules. This was mainly due to some conflicting dependencies between modules.

The solution that we proposed to that problem is the use of a microservices architecture. The main advantage of this kind of implementation is that every service is isolated from one another. In the present case, each module just needs to be wrapped with a REST interface, preferably using JSON-LD format and be inserted into a Docker image. The new architecture also integrates seamlessly with external REST services, which is perfect for the additional proprietary modules.

Another great benefit of this new architecture is that it can be used for projects of any size. For a personal testing projects, the user will only need to download some Docker images. In a production environment, the whole platform can be deployed in the Cloud, scaling horizontally, with the recommended persistence engines and will still rely in those same Docker images.

To draw a final conclusion, the MixedEmotions platform was able to adapt to the specific needs and modules of multiple partners, and thus is much more flexible now than it was in its original form. The initial idea of defining a monolithic platform before having all the requirements did not succeed. However, the consortium managed to work together on a new and customized revision. The original platform was not adopted in every pilot, and that was a clear signal that the platform was not well defined. If even the involved partners did not want to use the platform, surely the potential customers will also find it not appealing to use. Once reached that point, we changed the strategy, and instead of imposing our vision of a platform of them, we draw from them what make the platform attractive. The platform is now much better, much more flexible and scalable, because it had to take into account the needs of multiple partners.

APPENDIX A: JSON-LD, THE COMMON FORMAT FOR INPUT AND OUTPUT FOR THE PLATFORM MODULES

Introduction

As MixedEmotions is designed as a platform that can have interchangeable modules and even integrate with external modules, is imperative to define a standard exchange format. For this purpose a MixedEmotions schema using JSON-LD has been defined. (More information about JSON-LD here). The complete MixedEmotions schema definition is in its own page here.

NIF (NLP Interchange Format) defines a vocabulary and an API for NLP services. The key concepts to grasp are:

- All the text analysed is a String. Every String is given a URI (unique identifier)
- All strings belong to a Context
- Strings may have have attributes such as: entities, sentiment, lemma...

The NIF specification also defines how the URI should be computed. In a nutshell, URIs are like this:
http://example.org#char=0,40
(In blue is the URI of the Context)
(In yellow is the index of the String within the context)

A NIF document would look similar to this example:

```
<http://example.org#char=0,40>
        rdf:type nif:RFC5147String , nif:Context ;
        nif:beginIndex "0" ;
        nif:endIndex "40" ;
        nif:isString "My favourite actress is Natalie Portman." .
```

In principle, NIF has been created with RDF in mind. The example above uses the turtle notation. A more developer-friendly alternative would be JSON. Or, rather, JSON-LD. JSON-LD documents are JSON documents with some conventions/constraints on the structure and fields they contain, which are used to add semantics to the document.

Using JSON-LD, the NIF example we had before would look like this:

```
{
  "@context":  {
```

```
      “nif”: “http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#”
    },
    "@id": "http://example.org#char=0,40",
    "@type": ["nif:RFC5147String",  “nif:Context”],
    "nif:beginIndex": "0",
    "nif:endIndex": "40",
    "nif:isString": "My favourite actress is Natalie Portman"
  }
```


There are different ways to structure the same data in JSON-LD. The final scheme that we follow in our API is different, as it includes much more information about each context, as well as the sentiment/emotion analysis processes that produced the results.

MixedEmotions Schema

This is the defined MixedEmotions schema.


```
{
  "@context": {
    "topics": {
      "@id": "dc:subject"
    },
    "entities": {
      "@id": "me:hasEntities"
    },
    "suggestions": {
      "@id": "me:hasSuggestions"
    },
    "emotions": {
      "@id": "onyx:hasEmotionSet"
    },
    "sentiments": {
      "@id": "marl:hasOpinion"
    },
    "entries": {
      "@id": "prov:used"
    },
    "analysis": {
      "@id": "prov:wasGeneratedBy"
    },
    "dc": "http://dublincore.org/2012/06/14/dcelements#",
    "me": "http://www.mixedemotions-project.eu/ns/model#",
    "prov": "http://www.w3.org/ns/prov#",
    "nif": "http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#"
  }
}
```


A small explanation of each field.

- **Entries**: The sentence or sentences to be analyzed. Regarding this project, entries will always consist of a single text, but this schema supports multiple entries.
- **Analysis**: parameters and other data of the analysis that has been performed on the entries.
- **Topics**: topics detected in the entry.
- **Entities**: entities detected in the entry.

- **Suggestions:** whether or not this entry contains suggestions.
- **Emotions:** emotions detected in this entry.

The namespaces define the ontologies to be used when filling those fields. Most are standard, but the consortium created a new one for the cases where the existing ones did not suit this project needs.

Examples

These are more concrete example for the MixedEmotions schema, each focusing in a particular field.

Entry

This example covers the basic example in the NIF documentation: http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core/nif-core.html.


```
{
 "@context": "http://mixedemotions-project.eu/ns/context.jsonld",
 "@id": "http://example.com#NIFExample",
 "analysis": [
 ],
 "entries": [
  {
   "@id": "http://example.org#char=0,40",
   "@type": [
    "nif:RFC5147String",
    "nif:Context"
   ],
   "nif:beginIndex": 0,
   "nif:endIndex": 40,
   "nif:isString": "My favourite actress is Natalie Portman"
  }
 ]
}
```


Sentiment Analysis


```
{
 "@context": "http://mixedemotions-project.eu/ns/context.jsonld",
 "@id": "me:Result1",
 "analysis": [
  {
   "@id": "me:SAnalysis1",
   "@type": "marl:SentimentAnalysis",
   "marl:maxPolarityValue": 1,
   "marl:minPolarityValue": 0
  }
 ],
 "entries": [
  {
   "@id": "http://micro.blog/status1",
   "@type": [
    "nif:RFC5147String",
    "nif:Context"
   ],
   "nif:isString": "Dear Microsoft, put your Windows Phone on your newest #open technology program. You'll be awesome. #opensource",
   "entities": [
```

```
    ],
    "suggestions": [
    ],
    "sentiments": [
      {
        "@id": "http://micro.blog/status1#char=80,97",
        "nif:beginIndex": 80,
        "nif:endIndex": 97,
        "nif:anchorOf": "You'll be awesome.",
        "marl:hasPolarity": "marl:Positive",
        "marl:polarityValue": 0.9,
        "prov:wasGeneratedBy": "me:SAnalysis1"
      }
    ],
    "emotionSets": [
    ]
   }
  ]
}
```


Suggestion Mining


```
{
 "@context": "http://mixedemotions-project.eu/ns/context.jsonld",
 "@id": "me:Result1",
 "analysis": [
   {
    "@id": "me:SgAnalysis1",
    "@type": "me:SuggestionAnalysis"
   }
 ],
 "entries": [
   {
    "@id": "http://micro.blog/status1",
    "@type": [
     "nif:RFC5147String",
     "nif:Context"
    ],
    "prov:wasGeneratedBy": "me:SAnalysis1",
    "nif:isString": "Dear Microsoft, put your Windows Phone on your newest #open technology program. You'll be awesome. #opensource",
    "entities": [
    ],
    "suggestions": [
      {
        "@id": "http://micro.blog/status1#char=16,77",
        "nif:beginIndex": 16,
        "nif:endIndex": 77,
        "nif:anchorOf": "put your Windows Phone on your newest #open technology program"
      }
    ],
    "sentiments": [
    ],
    "emotionSets": [
    ]
   }
 ]
}
```


Emotions


```
{
 "@context": "http://mixedemotions-project.eu/ns/context.jsonld",
 "@id": "me:Result1",
```

```
"analysis": [
  {
    "@id": "me:EmotionAnalysis1",
    "@type": "me:SuggestionAnalysis"
  }
],
"entries": [
  {
    "@id": "http://micro.blog/status1",
    "@type": [
      "nif:RFC5147String",
      "nif:Context"
    ],
    "nif:isString": "Dear Microsoft, put your Windows Phone on your newest #open technology program. You'll be awesome. #opensource",
    "entities": [
    ],
    "suggestions": [
    ],
    "sentiments": [
    ],
    "emotions": [
      {
        "@id": "http://micro.blog/status1#char=0,109",
        "nif:anchorOf": "Dear Microsoft, put your Windows Phone on your newest #open technology program. You'll be awesome. #opensource",
        "prov:wasGeneratedBy": "me:EAnalysis1",
        "onyx:hasEmotion": [
          {
            "onyx:hasEmotionCategory": "wna:liking"
          },
          {
            "onyx:hasEmotionCategory": "wna:excitement"
          }
        ]
      }
    ]
  }
]
}
```


Named Entity Recognition


```
{
 "@context": "http://mixedemotions-project.eu/ns/context.jsonld",
 "@id": "me:Result1",
 "analysis": [
   {
     "@id": "me:NER1",
     "@type": "me:NER"
   }
 ],
 "entries": [
   {
     "@id": "http://micro.blog/status1",
     "@type": [
       "nif:RFC5147String",
       "nif:Context"
     ],
     "nif:isString": "Dear Microsoft, put your Windows Phone on your newest #open technology program. You'll be awesome. #opensource",
     "entities": [
       {
         "@id": "http://micro.blog/status1#char=5,13",
         "nif:beginIndex": 5,
         "nif:endIndex": 13,
         "nif:anchorOf": "Microsoft",
         "me:references": "http://dbpedia.org/page/Microsoft",
         "prov:wasGeneratedBy": "me:NER1"
       },
```

```
      {
        "@id": "http://micro.blog/status1#char=25,37",
        "nif:beginIndex": 25,
        "nif:endIndex": 37,
        "nif:anchorOf": "Windows Phone",
        "me:references": "http://dbpedia.org/page/Windows_Phone",
        "prov:wasGeneratedBy": "me:NER1"
      }
    ],
    "suggestions": [
    ],
    "sentiments": [
    ],
    "emotionSets": [
    ]
   }
  ]
}
```


Complete example

This example covers all of the above cases, integrating all the annotations in the same document.


```
{
 "@context": "http://mixedemotions-project.eu/ns/context.jsonld",
 "@id": "me:Result1",
 "analysis": [
   {
    "@id": "me:SAnalysis1",
    "@type": "marl:SentimentAnalysis",
    "marl:maxPolarityValue": 1,
    "marl:minPolarityValue": 0
   },
   {
    "@id": "me:SgAnalysis1",
    "@type": "me:SuggestionAnalysis"
   },
   {
    "@id": "me:EmotionAnalysis1",
    "@type": "me:SuggestionAnalysis"
   },
   {
    "@id": "me:NER1",
    "@type": "me:NER"
   }
 ],
 "entries": [
   {
    "@id": "http://micro.blog/status1",
    "@type": [
     "nif:RFC5147String",
     "nif:Context"
    ],
    "nif:isString": "Dear Microsoft, put your Windows Phone on your newest #open technology program. You'll be awesome. #opensource",
    "entities": [
     {
```

```
      "@id": "http://micro.blog/status1#char=5,13",
      "nif:beginIndex": 5,
      "nif:endIndex": 13,
      "nif:anchorOf": "Microsoft",
      "me:references": "http://dbpedia.org/page/Microsoft",
      "prov:wasGeneratedBy": "me:NER1"
     },
     {
      "@id": "http://micro.blog/status1#char=25,37",
      "nif:beginIndex": 25,
      "nif:endIndex": 37,
      "nif:anchorOf": "Windows Phone",
      "me:references": "http://dbpedia.org/page/Windows_Phone",
      "prov:wasGeneratedBy": "me:NER1"
     }
    ],
    "suggestions": [
     {
      "@id": "http://micro.blog/status1#char=16,77",
      "nif:beginIndex": 16,
      "nif:endIndex": 77,
      "nif:anchorOf": "put your Windows Phone on your newest #open technology program"
     }
    ],
    "sentiments": [
     {
      "@id": "http://micro.blog/status1#char=80,97",
      "nif:beginIndex": 80,
      "nif:endIndex": 97,
      "nif:anchorOf": "You'll be awesome.",
      "marl:hasPolarity": "marl:Positive",
      "marl:polarityValue": 0.9,
      "prov:wasGeneratedBy": "me:SAnalysis1"
     }
    ],
    "emotions": [
     {
      "@id": "http://micro.blog/status1#char=0,109",
      "nif:anchorOf": "Dear Microsoft, put your Windows Phone on your newest #open technology program. You'll be awesome. #opensource",
      "prov:wasGeneratedBy": "me:EAnalysis1",
      "onyx:hasEmotion": [
       {
        "onyx:hasEmotionCategory": "wna:liking"
       },
       {
        "onyx:hasEmotionCategory": "wna:excitement"
       }
      ]
     }
    ]
   }
  ]
}
```

APPENDIX B: DOCKERIZATION

Docker in a nutshell

Docker is a framework that permits to encapsulate services within a "container" so services can be deployed with all of its dependencies and in any environment.

You can find much more information about docker in their official website https://www.docker.com

Installation

Please refer to: https://docs.docker.com/engine/installation/

How to create a Docker image

There are two ways of creating a Docker image: by submitting an existing container into an image or by creating a Dockerfile.

Creating a Dockerfile

For creating images probably the best way is to have a Dockerfile which is a file that defines how an image will be constructed. The Dockerfile should be named “Dockerfile” and exist in the root of the project. Also it might be a good idea to include the Dockerfile in the github repository.

It basically defines how to construct the desired image, using one as a base (defined in the FROM instruction) and executing some commands on it.

A quick example:

```
FROM python:3-onbuild
MAINTAINER Carlos Navarro cnavarro@paradigmadigital.com
ADD src/     /usr/src/app
WORKDIR /usr/src/app
RUN “pip install tornado”
CMD ["python3", "topic_service.py", "2712"]
EXPOSE 2712
```

In this example, we suppose there is some python app under the src subfolder that will be started with the command “python3 topic_service.py 2712” that will listen on the port 2712 and that needs to install a dependency beforehand with “pip install tornado”.

- FROM base image which will be used as a starting point for building the image. There are a lot that can be found here: https://hub.docker.com/explore/ . Some examples are: python:3-onbuild, java:7 and java:8
- MAINTAINER: author
- ADD: copy some local folder to the image folder
- RUN: executes a command
- WORKDIR: change directory
- CMD: command to execute when running an instance
- EXPOSE: Open port (in which the service should be running)

There are many other commands, but these are the basic ones.

Then, for creating a Dockerfile, first find a suitable base image, add files, install missing dependencies, start the process and open the corresponding port

Building images

Once Docker is installed images can be built:

```
docker build -t {name} {basefolder}
```

For example: docker build -t topic_image .

Then the list of available images can be obtained with:

```
docker images
```

If you need to delete an image that is done with

```
docker rmi {imagename}
```

There is another way of creating an image, and that is from an existing container.

Run containers

Once the image is built you can start to create and run containers. There are a lot of ways to do this, but to keep things short just two quick commands.

To run a daemonized container with ports open:

```
docker run --name suggestion_container -d -P suggestion_image
```

To run a terminal inside the container (very useful when developing the future image):

```
docker run --name suggestion_container -t -i suggestion_image /bin/bash
```

Of course, multiple containers of a single images can be run, but each with its own unique name.

Running container can be seen with ‘docker ps’.
All containers with ‘docker ps -a’.
There you can find the port mapping like this:

```
0.0.0.0:32768->2712/tcp
```

Stop and delete images with docker stop {container_name} and docker rm {container_name}

Publishing images

Once an image is created, that image can be pushed into an online repository such as Docker Registry or Docker Hub (which is free, but public).From there images can be retrieved from anywhere.

For that, the image first has to be properly labeled:

```
docker tag topic_image:1.0.0 mixedemotions/topic_image:1.0.0
```

And then pushed into the appropriate repository (which has to have been created beforehand)

```
docker push mixedemotions/topic_image
```

From there the image can be downloaded as

```
docker pull mixedemotions/topic_image
```

MixedEmotions platform regarding DockerHub

A repository for MixedEmotions' Docker images has been created in hub.docker.com/r/mixedemotions

Please note that the repository is public, and it should be in order to allowing the general public to use the platform. However you can always put a disclaimer with the module specific license in the description.

Regarding naming convention, this is the format suggested:

mixedemotions/id_modulename_partner

So in order to upload certain image to the repository you can do, for example:

```
docker tag topic_service mixedemotions/13_topic_service_pt
docker push mixedemotions/13_topic_service_pt
```

Regarding versioning, it is very strongly recommended to have a tag for each past stable version. It is mandatory to have a "latest" for the current version. To create a version tag:

```
docker tag topic_service mixedemotions/13_topic_service_pt:0.1.0
docker push mixedemotions/13_topic_service_pt:0.1.0
```

Once the repository is created we suggest the following structure

Short description: One sentence description of the module
Full description: License, detailed description and how to use the service within the container.