UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER EN TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

DESIGN OF AN EMOTIONAL LIGHTING SYSTEM BASED ON SENTIMENT ANALYSIS OF TWITTER

JOSE FERNÁNDEZ FERNÁNDEZ

2019

TRABAJO DE FIN DE MASTER

Título:	Diseño de un sistema de iluminación basado en el análisis de emociones en twitter
Título (inglés):	Design of an Emotional Lighting System based on Sentiment Analysis of Twitter
Autor:	Jose Fernández Fernández
Tutor:	Carlos A. Iglesias
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MASTER

DESIGN OF AN EMOTIONAL LIGHTING SYSTEM BASED ON SENTIMENT ANALYSIS OF TWITTER

Junio 2019

Resumen

Hoy día existen diferentes proyectos basados en el análisis de sentimientos y reacciones de las personas en las redes sociales. Por un lado, en este tipo de proyectos, que incluyen el análisis de redes sociales, podemos ver cómo se representa la información recogida a través de gráficos para obtener conclusiones sobre la información. Por otro lado, hay muchos proyectos diferentes basados en la iluminación inteligente, el objetivo de este tipo de proyectos es intentar conseguir una forma de automatizar la iluminación basada en la interacción con las personas. Todo ello nos lleva a la representación de los sentimientos y opiniones que se dan en los mensajes de twitter a través de la iluminación, generando colores asociados a los sentimientos recogidos en las redes sociales y utilizándolos para iluminar una sala. Para que los mensajes recogidos sean analizados y representados de una manera nueva y dinámica. Aplicado al campo de la inteligencia artificial, el objetivo de este TFM es desarrollar un caso de estudio que investigue formas de representar sentimientos a través de luces y colores, los cuales cambiarán en función del sentimiento capturado, a través del análisis de los mensajes que los usuarios publican en un hashtag de un twitter. Para ello se va a desarrollar un sistema de hardware y software que cubra los siguientes objetivos específicos: desarrollo de controladores de hardware que actúen sobre luces de color o leds, desarrollo de nubes de etiquetas construidas con temas analizados con LDA y OLDA, desarrollo de una aplicación de software capaz de leer y analizar mensajes de twitter, configuración de una plataforma de automatización de tareas para dotar de funcionalidad IoT al sistema. En cuanto al desarrollo de software, se implementará una aplicación que lea y analice los mensajes para recoger los sentimientos que contiene. Este sistema de lectura y procesamiento de sentimientos se realizará en lenguaje Python y el analizador de texto que se utilizará es senpy, una herramienta cognitiva desarrollada por el laboratorio GSI (UPM). En cuanto al tasker, se incluirá para conseguir una mayor interacción entre el sistema de iluminación y otros actuadores. Para ello se utiliza la herramienta EweTasker desarrollada por el laboratorio GSI. Este tasker podrá gestionar estos parámetros para modificar las luces y pantallas. Por último, se utilizará Elasticsearch para almacenar los resultados, este almacenamiento se realizará utilizando el modelo de datos RDF.

Palabras clave: Ewetasker, Senpy, Sentiment, Emotion, LDA, OLDA, Elasticsearch, RDF, Tag cloud, Light, Twitter.

Abstract

Nowadays there are different projects based on people's sentiment and reactions analysis in social networks. On the one hand in this kind of project, that includes social network analysis, we can see how the information gathered is represented via graphs in order to obtain conclusions about the information. On the other hand, there are many different projects based on intelligent lighting, the target of this kind of projects is to try to obtain a way to automate the lighting based on the interaction with people. All of this brings us to the representation of sentiments and opinions given in twitter messages via lighting, generating associated colors to the sentiments gathered in social networks and using it to light a room. So that the messages gathered will be analysed and represented in a new and dynamic way. Applied to the artificial intelligence field this TFM's objective is to develop a case of study that researches ways to represent sentiments via lights that change colors, making these lights change depending on the sentiment captured via the analisys of messages that the users post in a twitter's hashtag. In order to do this a hardware and software system is going to be developed covering the following specific targets of developing a hardwares controllers that acts over color lights or leds, developing tag clouds builded with topics analysed with LDA and OLDA, developing a software application that is able to read and analyse twitter messages, configuration of a task automation platform in order to give IoT functionality to the system. About the software development, an application that reads and analyses messages will be implemented to gather the sentiments that it contains. This sentiment reader and processing system will be done in Python language and the text analyzer that will be used is senpy which is a cognitive tool developed by the GSI (UPM) laboratory. Regarding the task platform, its intended to include it to get more interaction between the lighting system and other actuators. To cover this it is used EweTasker tool developed by the GSI laboratory. This task platform will be able to manage these parameters to modify the lights and screens. Finally Elasticsearch will be used for storing the results, this storage will be done following the W3C specification using the RDF data model.

Keywords: Ewetasker, Senpy, Sentiment, Emotion, LDA, OLDA, Elasticsearch, RDF, Tag cloud, Light, Twitter.

Agradecimientos

Antes de comenzar me gustaría dedicar unas palabras de agradecimiento a todas las personas que me han acompañado y apoyado en todo momento. Sin ellos habría sido imposible terminar este trabajo.

Especialmente dar las gracias a mi madre y a mi padre, sin ellos no habría llegado hasta aquí. Gracias por enseñarme a continuar siempre hacia adelante a pesar de los momentos de flaqueza. También a mis hermanas Lorena y Paula, que me han empujado siempre que he necesitado ayuda.

Agradecer también los buenos momentos en la escuela a mis amigos. En especial a Dani, Fernando y Miguel, con los que he trabajado y me he divertido durante este máster.

También a los compañeros de trabajo, Victoria y Adolfo, sin los cuales el resultado final no habría sido el mismo.

Por último, dar las gracias a Carlos Ángel, por su guia durante este trabajo, sus consejos y su paciencia.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Nowadays we can find different systems and implementations that exploit the capabilities of Internet of Things technologies. These systems have the ability to control and change different devices that are installed in houses, public places or work centers. These implementations are oriented to achieve more comfortable and personalized places. IoT gives the possibility of change, schedule or configure devices such as temperature controllers or lights. Trying to reach a more connected world. In this project we will exploit the capabilities of this type of technology trying to find new and fancy applications for it.

Also The Artificial intelligence can give us more possibilities to improve this kind of system. The main purpose of AI is to analyze large amounts of data in order to search patterns that allow us to make predictions. But it also can be useful in cases like these ones. AI can understand and modify IoT environments by analysing the data generated in it. This can create IoT enviroments more user friendly that will make it easier for the user to interaction with them. The AI market has grown in the latest years making us question where the limits of its applications are. So this work will combine an IoT environment with AI in order to give a new vision of how to control an IoT environment.

This project deals with one of the fields in which the artificial intelligence has more impact, the social networks. Social networks are a big source of data. In the recent years, we can see multiple applications based on the management of this data. These are usually used in order to obtain more information from social network's users to give them better services, direct advertisements, etc. However this data analysis is not usually presented to them. In order to show how far can AI goes, it is interesting to use it for more visual objectives and represent this data in a more creative way.

Hence, the main target of this project is to demonstrate that currently the limits of the applications we can give to artificial intelligence are completely unknown. In order to achieve this, this works combines different technologies, such as IoT, AI and social networks. Together they offer a combination of elements that have been barely explored so far, with applications that are accessible to a very large public.

To achieve this, it is proposed to create an emotion lighting system for an event. These events usually involve a large number of attendees. They will give their opinions about it with their interactions in social networks. The system will analyze the feelings reflected in the contributions posted by the assistants. In order to visually represent these feelings, there will be two types of representations.

On one hand, colors are going to be used. These are powerful representations of feelings and it are being used to represent them. These lights will vary their color as the feelings vary.

On the other hand we create a tag cloud that complement the lights. A tag cloud is a word representation, this kind of representation draws a cloud formed by words. It is usually created with the most representative words of a text. For this, a more deeply analysis of the information given is done. So the system obtains the main represented topics in the people's opinions. Finally it composes a word cloud with the most representative words. Both representations give a personal environment, that is controlled by AI over an IoT infrastructure.

1.2 Project goals

During the development of this project, a series of objectives are attempted. With them it is possible to create a more resilient and lasting system. To achieving a system more adaptable, scalable and open to be able to be improved in the future.

- **Desing and build of an IoT system.** This means creating a system in which each of the pieces is remotely interconnected. Giving the system the ability to include,

remove or change sensors and actuators without affecting the design of the solution. This allow it to evolve and change. Also it is more adaptable system and fits better with the different environments where it is going to be deployed.

- **Desing and development of the actuators and sensors of this system.** This includes the decisions that are taken in order to create sensors and actuators that accomplish the idea that it could be change or improved without affecting the whole system. Providing the necessary mechanisms to allow a user to include more than one if they want to, or remove the ones that does not fit with each different situation in witch the system could be deployed.

- **Configure the algorithm that are necessary for the analysis.** It is necessary to understand which are the objectives of the system. On the one hand the analysis of emotions and feelings. On the other hand the analysis to extract the main topics. So it is important to understand and compare the tools and algorithms that allow us to obtain the best results in this field.

- **Integrate the system with a task manager.** The task manager will be the core of the IoT system and therefore the core of the whole system. It is necessary to understand how the Tasker works, because it is necessary to modify it. This improves the adaptation of the sensors and actuators, allowing us to connect remotely all the pieces that will compose the whole solution.

- **Integrate the system with the external devices.** If needed the analysis of the input data could be managed by external devices, in cases such as these the system must be able to connect with them either locally or remotely. This is the case where the emotions analyzer is used wich stores the resulsts in the database

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is as follows:

Chapter 1 Introduction, here the context and the goals of the project are explained

Chapter 2 Enabling technologies, in this section all the technologies and tools that are used in this project in order to create the system are going to be explained.

Chapter 3 Requirement Analysis, in this section the functionality of the application, and the requirements that will be necessary to implement in order to satisfy the purpose of this work are going to be explained in detail.

Chapter 4 Architecture, it provides an overview of the system , it also explains all the pieces developed and changes that are implemented in order to create the system with all its functionalities

Chapter 5 Case Study , in this section the real motivation of the project is explained, as well, two real cases are shown.

Chapter 6Conclusion and future work, this chapter summarizes the project's conclusions of the projects and draws the future lines to solve problems that are not covered for the scope of this project. Also, it will show the problems faced during the development of the project.

CHAPTER 2

Enabling Technologies

This chapter provides an overview about the technologies in wich this projects is based. Also it will include the explanation of the principle concepts of it. It also includes an explanation of the operation and the main concepts of this technology, deepening if it is necessary.

The main technologies used for this project are ElasticSearch, Flask, Senpy, RDF, Gensim, EWE and EWE Tasker.

2.1 Ewe Tasker

EWE tasker is a tool developed by GSI in order to solve the problem of exchanging data with IoT devices [11]. EWE tasker solve the task automatization problem by using a rule based task automatization. EWE tasker provides a server and a webclient in order to manage actions and events and also give a human user interface for users to define rules.

On one hand, The server have the engine of the application and it will be listening for events provided by any device which is connected to it, and also it will be ready for sending an action if a certain rule trigger it. On the other hand, the web application provides a service in which users could create their own rules and channels by setting parameters. This way, the communication could be created between the sensor and the actuator, based on the rules that the user set on the web. Rules could be included on the server on the same way

the events and actions are. However the web gives a user friendly human-machine interface, in order to make easier the creation of rules by non technological users.

EWE tasker uses vocabularies for the definition of the channels, this vocabularies are defined with RDF in a n3 file[11]. The channels are the models defined for the input events and output actions. Also, as it is said before, rules are defined in the same way, however they could be created on the web as well as they could be included by defining a n3 file.

EWE Tasker is based on the creation of channels and rules.

Firstly, channels are the connection between the task manager and the sensors or actuators. Depending of the nature of the channel, it could have events, that are received from actuators. Also a channel could include actions, that send commands or other actions that an actuator have to done. Furthermore a channel that implements both of them, events and actions, could be created.

Secondly, rules are the relations between events and actions. Rules simply defines in what cases an event will trigger an action and what action should be set off.

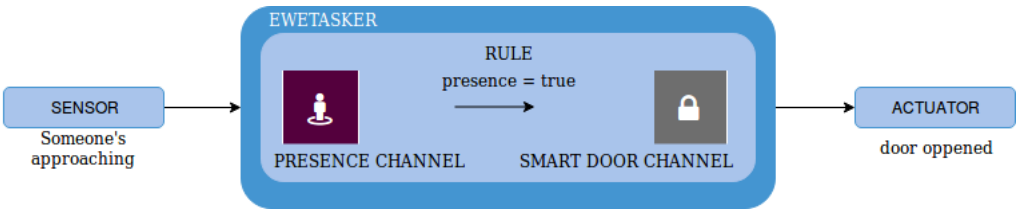

Figure 2.1: Rule Scheme

The figure above shows an example of a rule with the following items:

- A proximity sensor that generates an event when someone is approaching.
- A presence channel defined with the parameter presence.
- A rule that trigger an action when presence is true.
- A smart door channel that implements open and close actions.
- A door actuator that locks and unlocks the door.

Classes

The most important classes that are defined on EWE tasker are channels, events, actions and rules.[10]

Channel

A EWE tasker channel is the piece in EWE tasker which is responsible of generate events or trigger actions. These channels are linked with sensors or actuators. The definitions of these channels are implemented in RDF and they are included on the EWE tasker code. Also this channel could be added after the application deployment if , as in this project happens, it is necessary to create new channels. When a channel is defined then it will appear at EWE tasker webclient interface and it could be used in the webclient to set and define services, devices and rules.

- A service is a channel properly parameterized that trigger an action. Becouse of this, a service could only be created from a channel that have actions.

- A device is also a channel defined and parameterized . In contrast a device could only be created from a channel that have events.

For instance a channel could be a gmail definition with the event "receive an email" and the action "send an email". Other example can be a temperature sensor that defines its event as the value of a measure in degrees.

Event

This class defines a particular occurrence in a process. Events are triggered instantaneously and do not have duration over time. Events will be triggered by an external sensor, if a channel is defined properly then it will catch that event. When an event is processed EWE tasker will make a properly action based on the previously defined rules. Events also have more details that could be used to set the rules when EWE tasker makes a decision, this details are defined as parameters. The events are model with input parameters and output parameters. The difference between them are that the output parameters comes from the sensor but input parameters are defined as a constant when the rules are created.

So with the same channel and the same event, two differents ways to process the same data from a sensor could be defined by fixing an input parameter. For example an output parameter in the case of a temperature sensor is the measure readed. However an input parameter, in the same example, is the ID of the sensor. If more than one rule is created this way, more than one sensor will be managed. This way event definition are not constrain to certain channels but more than one channel may generate similar events.

Action

This class defines the operation that the channel have to provide when a certain event is catched. An action can be resolved by making a decision such as change a light, modify temperature or send an email.This means that actions will be performed directly on an actuator based on decisions. These decisions can only be made on the input parameters. In this parameters would come, for example, the temperature measurement in the case of the temperature sensor. Also in actions, a difference between input and output parameters exists. As before, input parameters will be setted on rules creation and output parameters will be received. Both of them will be used to define the action on the performer action.

Rule

This class is used to define the relation between events and actions. A rule will be defined as one of the events belonging to one channel and as one of the actions belonging to another channel. These channels can be the same or not. In the case of a channel that has actions and events, such as the email channel, can compose a single whole rule.So that, for instance, the email event received can trigger an action email sent.However there is also a condition or set of conditions that a rule have to match. this conditions are defined on rule creation. It is usually a logical condition based on the output parameter coming from the event, such us a value equal to a constant or a value greater than a bound.

To sum up a rule is compose of an event, coming from the sensor, a condition , or set of conditions evaluated in EWE tasker and an action, that will be sent to the actuator.

Important modules

The most important modules defined in EWE tasker, in order to complete the process of task management, are the validator and actuator.

Validator

Validator is he python module designed to analysed an event that comes to EWE tasker. They are used to confirm the crossbar request. Each event has its own validator, and a validator has to fit the parameters defined on the n3 file. Also in the validatior is necessary to specify the values that are necessary, so always will be defined, and the values that are optional parameters, those parameters that only appear in some specific events. So not all crossbar requests will have to implement it.

Performer Action

As well as every event have to be defined in validator module, each action is got to have a definition in the performer manager, in order to compose the message that will be sent to the actuator device. This action define the parameters and actions to perform. Each channel have to define an python actuator module.

Transport Protocols

Two different ways to make the call can be implemented in a sensor in order to share an event with EWE tasker. HTTP and crossbar[10].

HTTP

EWE Tasker can receive events by REST API by using a http request with the post method. The information in the request have to be written using n3 notation. The API is running on port 5050 and use the parameters:

- Username : same user created on the webclient application, in order to create the rules.
- Event: which is written using n3 notation, and have to content all the necessary parameters to trigger the action, as they were defined when the channel event was created.

For instance using the example of the temperature channel:

Listing 2.1: Http event example

```
import requests
url = "http://localhost:5050/evaluate"
f1 = open ('./event.n3','r')
event=f1.read()
payload = {'username': 'administrator', 'event':event}
data_channels = requests.post(url, data=payload)
```

With the n3 file composed as:

Listing 2.2: N3 file for http event example

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
```

```
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.

ewe:TemperatureMeasured rdf:type <http://gsi.dit.upm.es/ontologies/ewe/ns/
    TemperatureMeasured> ;
    rdf:type ewe:Event ;
    rdfs:domain ewe:TemperatureMeasured .
ewe:TemperatureMeasured ewe:hasParameter ewe:TemperatureSensorID .
ewe:TemperatureSensorID rdf:type <http://gsi.dit.upm.es/ontologies/ewe/ns/
    TemperatureSensorID> ;
    rdf:value "ID_NAME" .
ewe:TemperatureMeasured ewe:hasParameter ewe:Degrees .
ewe:Degrees rdf:type <http://gsi.dit.upm.es/ontologies/ewe/ns/Degrees> ;
    rdf:value "TEMP_VALUE" .

{ ?A ?B ?C } => { ?A ?B ?C }.
```

Crossbar

Ewe Tasker can use crossbar as well. The crossbar server is up in a docker container inside the application. So any method can be used to comunicate with EWE Tasker. However, the sintaxis and creation of the JSON object with the parameters is easier, so crossbar improves the simplicity of sensors creation.

Crossbar needs to define the procedure, and a set of arguments that will compose the JSON file. The parameters used with crossbar are:

- User, name of the user created on the webclient application.
- Channel , name of the channel.
- Event, name of the event.
- Parameters of the specific event.

An example of this sintax is shown below:

Listing 2.3: Crossbar event example

```
import requests
import json
Teh url = "http://localhost:8082/call"
data = {}
```

```
data['procedure'] = 'com.channel.event'
data['kwargs'] = {
  "user": "administrator",
  "channel" : "TemperatureSensor",
  "event": "TemperatureMeasured",
  "params" : {
    "TemperatureSensorID":"123.123.123.123",
    "SenpyEmotion" : "marl:Degrees",
    "SenpyIntensity": "30"
  }
}
json_data = json.dumps(data)
data_channels=requests.post(url, data=json_data)
```

2.2 EWE ontology

The Semantic Web aims to give a structure to the meaningful content of web pages. The objective is to create an environment in which artificial intelligence agents can carry out complicated tasks, being able to jump,in a simple way, from one page to another[6].A program of AI that wants to compare or combine information across two sites on the internet has to know how the terms are related, and know if are referring to the same topic if it were.Semantic web try to solve this problem by creating collections of information called ontologies. Ontologies are documents that formally explain the relations among terms. This is amed by assigning properties to classes and allowing subclasses to inherit such properties.

EWE is an ontology, so will describe an scheme to standardize the data necessary to solve the problem of a task automatization service. It will include definitions of rule, channel, actions, events, and other definitions. So the necessary vocabulary, for task creation in EWE tasker, is given by EWE ontology.[9]

An overview of the scheme is shown in the figure 2.2.

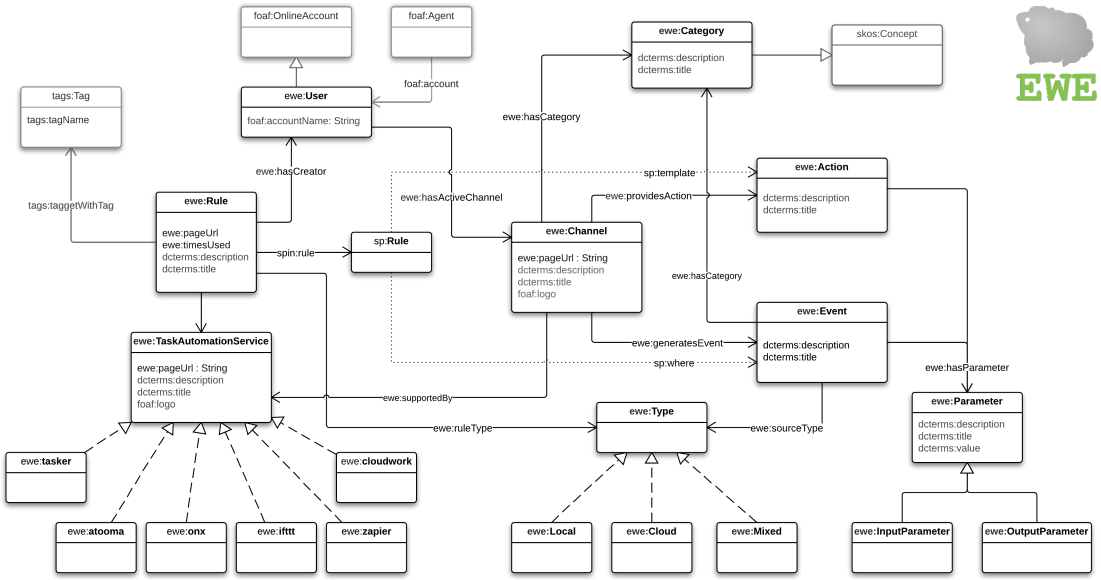

Figure 2.2: Ewe ontology [9]

2.3 RDF and n3

RDF, Resource Description Framework[28] provides a simple way to define data models by expressing it with statements. This statements are compose by triples. Each triple have a subject a predicate and a value. The subject and the predicate refers to an object that can be defined in the same document or can be refered to another resource on the web. On its part, the predicates references the relation among objects[6].

Specific vocabularies are used to write sentences in RDF. These vocabularies are defined by creating a scheme. For this project the vocabulary used is EWE.

Because it will be necessary to create correctly modeled and well hierarchical sentences, RDFSchema is used. RDFSchema (RDFS), will include primitives that will indicate if there are inheritance of classes (Class/Subclass), relations between classes (properties) and restrictions of range and domain, on these properties.

It can also be seen that with RDFS, hierarchies can be described, which gives sufficient power for complex queries and reasoning, while RDF can only be used for the creation of simple triples.

RDF was made to use XML syntax. But, in this case, n3 notation (notation 3) is going to be used. This election is because, this is the notation is used by EWE tasker. In addition this type of notation is more readable for humans, in addition to being a more compact notation.

2.4 Senpy

Senpy is an emotion and sentiment analysis service [15]. Senpy uses Linked Data principles based on the NIF (NLP Interchange Format) specification. It supports different input methods including JSON or text plain. It can be reached via API by making an http request.

In the following figure we can see the Senpy architecture formed by two modules[14]. On one hand it is the senpy core, on the other hand there are different plugins wich are responsible for analyzing the inputs according to the selected algorithm. Depending on the plugin senpy will compose the output that will be based on a linked data model.

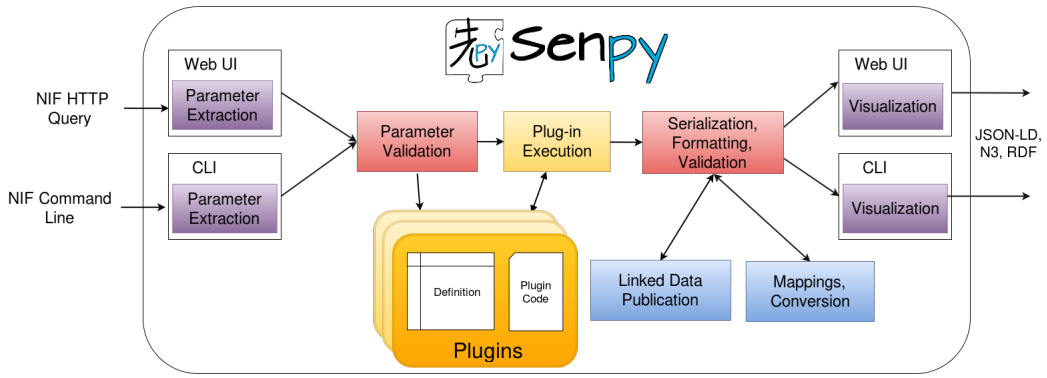

Figure 2.3: Senpy architecture [14]

Senpy has been applied to sentiment and emotion analysis services using the following vocabularies:

- Marl, that is a linked data gsi solution designed to describe subjective opinions expressed on the web. The response come with a polarity that could be positive neutral or negative. [12]
- Onyx, that aims to complement the Marl Ontology by providing a simple means to describe emotion analysis processes and results using semantic technologies.[13]

The main algorithms used in this project are sentiment-140 and emotionwannafect, in order to anylize input data with sentiments or emotions.

2.5 Gensim

Gensim is a python library which provides unsupervised topic modeling with natural language processing by using machine learning. Gensim provides tools to compose dictionaries and corporas, which are necessary to implements the topic modeling.[25]

In order to compose a tag cloud, it will be necessary to get the more relevant words in a speech. To solve this, it can be approached as a natural language processing problem, so that LDA and OLDA algorithms will be useful in the resolution.

This library provides LDA and OLDA algorithms[24] that will be useful during the development of the project:

- LDA, Latent Dirichlet allocation is a natural language processing algorithm. This will be a first tentative in order to try to solve the problem of seeking main topics in a text.[7]
- OLDA, Online Latent Dirichlet allocation is also a natural language processing algorithm. However this will fit better the final solution because it will be more accurate for cases, such as this, where data is obtained by streaming.[16]

2.6 Flask

Flask is a lightweight web application framework written in python. Flask uses python and provides a simple template for web development.[3]

Flask will be useful in order to create both sensor and actuator. On the sensor case, Flask will be useful in order to compose a simple web portal in which users will be able to configure a few basic parameters. However in the actuator it will be use as a simple way to display an interactive tagcloud.

2.7 Elasticsearch

ElasticSearch is a real-time search and analytics engine. It is a distributed tool, it is capable to make analityc queries and it is reached via API by the aplication. ElasticSearch will be useful to store data, this will provide a datastore useful in test cases and also in the post analysis of data.[27]

2.8 D3js

d3.js is a JavaScript library used to manipulate documents based on data. It uses html, css, and svg to create visual representations of data which can be viewed on any modern browser. It also provides some features for interactions and animations. It is used to compose personal dashboards embedded in html pages.[29]

Embedded within an html webpage, the JavaScript D3.js library uses JavaScript functions to select elements, create SVG objects, style them, or add transitions, dynamic effects or tooltips to them. The data that we want to represent is going to be in json format, so it fits the functionality of d3js that can be in various formats, most commonly json, csv or others.

In this project is going to be used in order to compose the word cloud, because d3 allow us to create interactives tag clouds by varying the input data. For the word cloud composition it is going to be used the library d3 cloud that is a wrapper that facilitates the use of the d3 functions.

CHAPTER 3

Requirement Analysis

In this section the functionality of the application and the requirements are shown.

3.1 Use case

It is important for the project to analyze the process that the system will have, in order to clarify it and understand the requirements that will be necessary to properly develop it. Thus, the basic use case is going to be presented, then the basic description of the need that solves the system is going to be shown to finally get the requirements that are going to be implemented in the final project. To give an overview is shown the case use diagram in Figure 3.1.

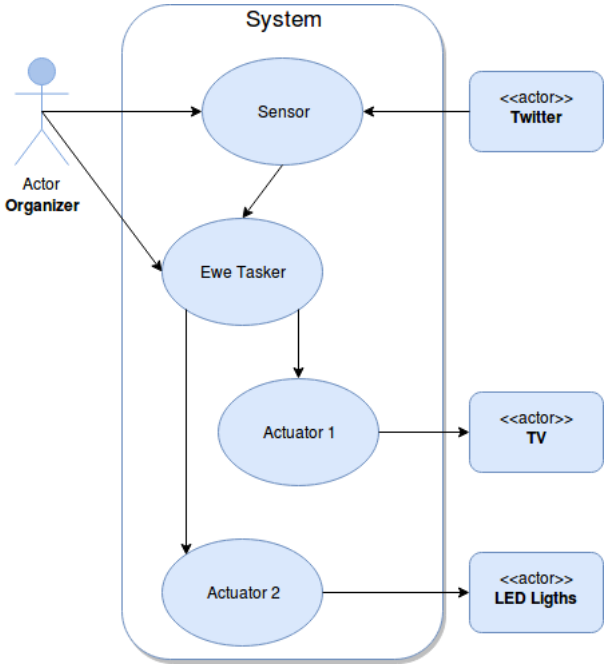

Figure 3.1: Use case diagram

3.2 Description

An event organizer wants to give a more personal atmosphere to an event. In order to do this, he wants to track and discover comments, emotions and feelings around this event. He also wants to move the feelings to get a more personalized environment that enhances the experience of those who attend to the event. In this case, the event organizer needs to represent positive, negative and neutral feelings as well as representing the topics discussed by composing a tagcloud.

Then, once the event has started, the event organizer logs in the sensor web client. This way he can select which hashtag, related to the event, he wants to track in order to follow and analyze the messages posted on it. He also sets the emotion analysis with the two possibilities emotion and sentiment.

Once this is set, the sensor will track tweets. The messages will be analyzed in two different ways. First, it runs the process to get the emotion and then, it runs the process to get the main topics. Regarding this, each tweet and its following analysis will be stored for possible further analysis.

Then EWE Tasker sends the corresponding action to the actuators that are listening. Both of them will respond to this, one of them by changing the lights to the corresponding color and the other one by refreshing the tagcloud that is presented on a screen.

3.3 Actors

To achieve a correct functionality for this System the actors around The system has to be the following:

- Primary actor:
  - event organizer, the event organizer/s who control the system. His/their function is to select the proper analysis (emotion or sentiment), and the proper hashtag to track, he/they also decide/s when the system starts and stops.
- Secondary actors:
  - Twitter API, to retrieve the necessary messages with the opinions of the attendance.
  - TV, a screen where the tag cloud will be projected.
  - Led Lights, led lights prepared to display colors.

3.4 Requirements

With all the above, the system requirements are defined divided into functional and non-functional requirements.

3.4.1 Functional requirement

The system has to track a conversation posted on a social network.

This is the main requirement, the purpose of this application is the development of artificial intelligence for social networks and the lighting of an event, for which the first step is the analysis of interactions in a social network. The most common social network to exchange opinions is Twitter, so that it is going to be the social network chosen to get the opinions. Because of that, the application has to implement all the necessary features in order to have a fluent interaction with the twitter API.

The system has to analyzed sentiments and emotions.

In order to get more than one implementation of the analysis, the system has to be able to change between sentiment analysis and emotion analysis, this will provide more choices in order to give a more completed representation, allowing the user to change the motivation of the analysis depending on its purpose.

Each sentiment or emotion have to get associated with a color.

The system will have the option of changing the colors that are being represented. This way, the system will allow the user to change the colors that are going to be used give a more custom experience. However, there are a lot of studies about how the colors represent different sentiments, so the system must have a predefined set of colors for each type of representation, emotion and color. This way, the user will be able to choose among the predefined configuration or a custom configuration.

The system has to get the main topics.

The other way the system can analyze the input is via natural language processing, so the system has to implement a module prepared to analyze the texts using natural language processing.

The system has to implement a simple interface to control its processes.

A simple interface to control the start and stop of the main functionality has to be implemented. Also, it has to provide the functionality of hashtag selection and other simple functionality for the system control.

The system has to implement a IoT design.

In order to link the resulting data from the analysis with the actuators, the system has to be designed as an IoT environment. This will provide a scalable environment with the capacity of evolving the system by including new devices.

Actuators have to be IoT actuators.

Two actuators have to be developed. They have to be developed in the same way as the whole system by implementing IoT. So that a correct representation could be done, there have to be two different actuators, one capable of representing the main topics and the other capable of representing emotions and feelings.

3.4.2 Non-functional requirement

Sentiments and emotions to be represented.

The sentiment and emotions will be the ones that senpy is capable of extracting. On one hand the system has to get neutral, positive and negative when sentiments are being analyzed. On the other hand, the system has to get joy, disgust,etc. in the case of emotions.

Storing of the data.

The result of the tracked data and their analysis have to be stored in order to get additional information in further analysis. For this, the system has to implement a data base capable of managing big data.

The emotions and sentiments will use lighting.

For the representation of emotions and sentiments via colors, one of the most powerful way is by illuminating a room using led lights that can represent a big variety of colors.

Main topics will use word clouds to be represented.

Wordcloud is useful for quickly perceiving the most prominent terms and for locating a term alphabetically to determine its relative prominence. It is widely used in media and well understood by the public. So, for the representation of the main topics is very useful the usage of tag clouds that will change their size and variety depending of the words used in the main topics.

3.5 Design decisions

For the development of this solution there is no requirement that requires the use of a specific programming language. However, there is an important set of libraries on python that provide functionality with big data, artificial intelligent and internet of things. Also, Ewe tasker, that will be a big important piece of the whole system, is developed using python. So, the best option in this case is python.

However, there are parts of the project that will be done using java script, in the case of the representation of the information, sometimes representing the data with javascript libraries will be a better option. To be able to use the d3js library, that provide a big functionality representing data, the front end of the wordcloud actuator will be developed using javascript.

Another design decision is which social network is chosen to analyze. The most common social networks that are used to exchange opinions are Facebook, Instagram and Twitter.

First, Facebook is a social network that launched to the public in 2004. It currently has more than 2200 million users worldwide, making it the largest social network in the world. Communications on Facebook are made through posts written on profiles board or home pages that belong to the users' contact profile. For the scope of this project, the information we could analyze, regarding all that is posted on facebook, is the information written by the users, discarding all the other information that is there such as pictures or multimedia

content. As it is said before, it is necessary to track a conversation. Facebook as well as Twitter can tag commentaries with hashtags. But the main problem using Facebook it is that we cannot get all the commentaries from a conversation because facebook has higher levels of privacy, that allow users to configure their profiles, so they are not visible to the users whom they do not interact with. Despite facebook providing an API in a similar way twitter does, Facebook is not a good option giver for the scope of this project.

Secondly, Instagram is another social network to consider. It is one of the most famous social network and one that have more activity. It is also property of Facebook and it started in 2010 and stands out for its ease of content sharing. The main kind of information we can find on instagram are pictures and its comments, but here we cannot find big threads of conversation around topics in general. The value of its content comes practically only from images, so for the study of feelings the amount of text we could use to analyze it would be quite scarce or unrepresentative. Especially considering that the scope of this project is based on the natural language processing. In addition, like Facebook, Instagram has a series of APIs that allow access to information shared on this social network, but unlike Facebook, images shared in Instagram are the property of the person who has shared that image, the access to Instagram APIs is subject to a review of the purpose for which it is requested in addition to requiring the address of a reference website in which we would have to implement the functionalities for which APIs are needed.

To sum up, Facebook and Instagram give us content that provides information with low value content, because this system will be based on natural language processing. Also this information is very restrictive due to privacy policies from both companies. This is a big barrier of entry and dealing with that is out of the scope of this project. Finally, the last social network to consider is twitter, that will be the best option for the developing of this project. Twitter is a social network in which the users interact among them by posting short messages (constrained to 280 characters). Twitter also has low levels of privacy allowing every user to access to other user's information. This is because of the nature of this social network, that is to share public messages to everybody who wants to read them.

As said before, the main way to exchange opinions on Twitter is by short messages. Also the user can exchange gifs, pictures or links, but the differences with Facebook or Instagram it is that the main content is shared by typing messages, and this feature fits well with the target of this project.

As the other social networks, twitter provides an API, but in this case, it is much easier to use it as developer. However, it has different levels of users that includes a paying option. But this is only a limit to use the API by only tracking one tweet per five seconds. This does not disturb the target of this academic project so it fits well for the scope of the system. Comparing and testing the three social networks that are mention above, we can

see that Twitter is the social network that better fits this project. Facebook offers the same functionality but the users are less reachable than in Twitter and Instagram generates other type of content that will not be useful for natural language processing. As conclusion for the development twitter will be used.

Other important design decisions have been made, which are not going to be explained in more detail in this section that only intends to give an overview of the most basic decisions. However, every technology used in this project is described deeply in other parts in this document.

CHAPTER 4

Architecture

4.1 Overview

This section shows a design overview. This projects includes software and hardware. The software that have been developed includes the processes necessaries in order to gathered data and analyses it. The hardware development includes the development and configuration of the devices. These devices are in charge of representing the results of the analyzed information. The system is designed as an IoT environment. A project like this is composed of three layers such as devices, edge gateways and cloud.

The first layer is composed by all those devices that include networked things. Sensors and actuators are in this layer. All these devices interact with the environment by taking measures, but, also with people who interact with it. The second layer is composed by gateways. These have the functionality of connect actuators and sensors. This way information is redirected to any other device in the network. The third layer is the cloud that is the piece in charge of managing the information. for instance the analysis of data.

In Figure 4.1 the diagram that has been followed to build the system can be seen.

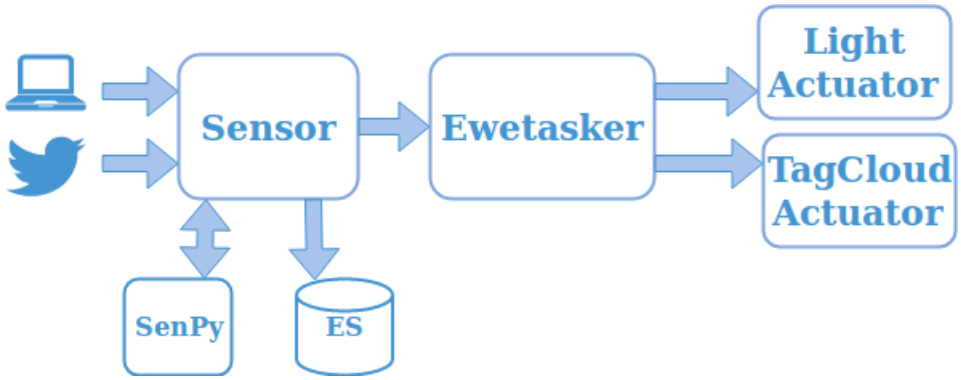

Figure 4.1: Architecture overview Diagram

Each piece shown in the Figure 4.1 it is composed as a unique and independent application. This way, in terms of evolve, the project could change and improve by modifying every single piece, keeping the main functionality. For example, if a new sensor is developed to track commentaries in another social network, that piece could be added and the whole system could continue working undisturbed.

The purpose of the project is to get an opinion from a social network, analyze it and represent the results in two innovative ways such as lighting and tagclouds. So the system has to implement different pieces that will solve the problems of, data gathering, data analysis, and communication and representation of the data. For this, the principal pieces are represented in the Figure 4.1. This pieces are the sensor, Ewetasker and both actuators.

The first piece will be the sensor. It is a piece that will combine the capacity of reading data as well as processing and analyzing it. Sensor will also have the task of being the interaction between the user and the system, bringing the capacity of choosing the hashtag and also the analysis type, between sentiment or emotion analysis. Sensor is divided in three different pieces including flask server, controller and OLDA analyser. Firstly, server provides the basic human machine interface. Secondly, controller has the functionality of coordinating the other pieces. Finally, OLDA provides the analysis to get the main topics in the data. Furthermore, sensor has been developed to be able to connect with two external pieces senpy and Elasticsearch. Senpy gives the emotion and sentiment analysis and data will be stored in Elasticsearch. Also sensor sends tasks to Ewetasker, so the controller is connected to Ewetasker by using crossbar.

The second piece in this architecture is Ewetasker. It is configured to be able to communicate with the sensors and actuators. Ewetasker manages the tasks and communicates with the sensors and actuators.

Finally, both actuators receive orders from Ewetasker and execute them. Light actuator does this by changing lights and Tag cloud actuator by changing the tag cloud. The first

one is composed by an arduino connected with the led lights. These lights have the capacity to represent colors defined by RGB codes. So Ewetasker will send different orders to a Light actuator public IP depending on what color has to be represented. The tag cloud is a device that implements a flask server in order to use it as an API REST to manage the orders from Ewetasker. This actuator also provides an output that prints the tag cloud of each iteration.

In terms of functionality, the system is implemented in such way that every single tweet is going to be analyzed. So, every time the sensor detects a new input, the data is going to be analyzed as a single document. It is going to be important in both ways the data is analyzed.

First, when we are analyzing emotions, the target is to be able to get the result of the analysis of each commentary, because of the nature of data. Every single tweet is completely different from one user to another, and the system does not have the functionality of group the input that is similar or somehow related. This way, there is no point in putting together a subset of tweets and then get the whole emotion of this subset.

Secondly, in terms of language processing, it is possible to create a subset of tweets to analyze them together as a unique one document. However, it is not the best option to get a dynamic representation of the main topics. It is achieved a powerful and dynamic representation by displaying the topics in a tag cloud. This representation is useful because it allows us to check in a glance how the terms represented are changing with time, how the most relevant will become larger and how a few will lose weight in representation, or how, the most irrelevant, will disappear from the figure. This streaming analysis and representation will have special features, so it will be important to select an algorithm of natural language processing that fits the solution. How this algorithm is going to be selected will be describe later in this section.

Once each tweet is analyzed it will be stored in a proper way at the same time an event is triggered on Ewetasker sending the proper data to it. Then Ewetasker will trigger two actions on each actuator.

In the following sections the functionality of each piece is presented in detail, explaining each of its components, how they have been developed, and how they work.

4.2 Sensor

The sensor it is developed as an independent application As it has been said before, it is composed by different pieces that are going to be explained further in detail. The function of this piece will be to describe a pipeline that will track data from twitter and then analyze it. Once the data is processed this piece will store it and send an action to Ewetasker. The

modules are interconnected and work in the way shown in Figure 4.1.

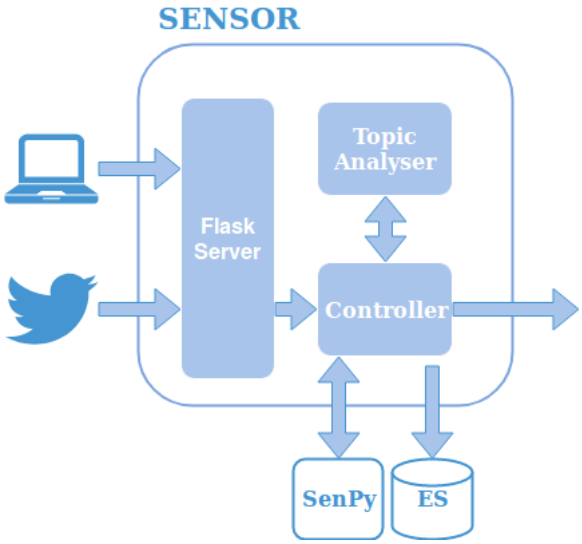

Figure 4.2: Sensor architecture

The sensor is composed of flask **Server**, **topic analyser** and **controller**. Also implements conections with **Senpy**, **Ewetasker** and **ElasticSearch**.

4.2.1 Controller

The controller is the main code in the sensor. Its function is to provide the core of the application. This means that it will manage the data obtained from the source and connect it with all the other pieces that will analyze and store the data. The controller will work as shown in the figure.

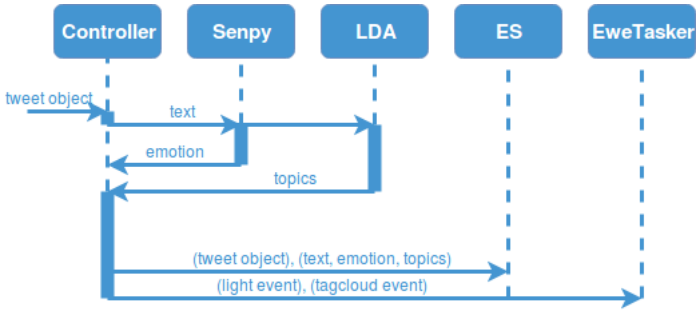

Figure 4.3: Controller flow diagram

As it is shown in the figure, the system performs the following actions:

- It receives the tweet object as input
- It sends text contained in the tweet object to both senpy and LDA analyser

- It recieve the response and compose the json object that are going to be stored into Elasticsearch.
- It triggers two new Ewetasker events.

The input is received from the flask server, here we have a raw tweet object. The controller have to get the most important information from the tweet object,because it is full of information that will not be used in this project. Thus, for the development of this project it is important to get the tweet id, the text and the language.

- Tweet id is a unique number that unequivocally identifies each tweet. It is an integer and will be useful to store the data in elasticsearch, this way, if a tweet is duplicated elasticsearch will discard it automatically.
- Text field contains the full text from twitter, this is the most important information arising from the source. Because is the part of the tweet object that is analyzed.
- Language, for the scope of this project only English tweets are going to be analyzed, because of that, the controller discards every non-English tweet.

Once the important data from a tweet is obtained, the controller discards non-English texts. Then, each text is prepared for processing in two different ways because of the needs of each analyzer are different . The first one is the process necessary to do the sentiment analysis that is going to be explained in the senpy section. The second is the same cleansing process but for the language process that is going to be explained in the algorithm section.

Once the data is analyzed it will be stored on elasticsearch. For this it iscreated one index that will be composed by the important tweet information, the result of the sentiment analysis and the corpora. Also it includes the main topics and words obtained. in Listing 4.1 is shown how a tweet object is stored. Also it is shown the JSON object in Listing 4.2.

Listing 4.1: Elasticsearch object 1

```
es.index(index="index", doc_type='tweet', id=storage['id_tweet'], body=
    storage)
```

Being storage a json object like this

Listing 4.2: Elasticsearch object 2

```
{
  "tweet": text,
  "id_tweet":int,
```

```
  "polarity_value": float,
  "has_polarity": polarity_string,
  "json2Cloud": json_object
  "date": datetime,
  "timestamp": int
}
```

For the tag cloud actuator to understand the data from OLDA, data must be parsed to a json object. This object is the same that is stored under "json2Cloud"(Listing 4.2) and it looks like is shown bellow.

Listing 4.3: LDA response

```
[[{"text": "#avengersendgame", "size": 32}, {"text": "joker", "size": 28},
    {"text": "much", "size": 27}, {"text": "ticket", "size": 24}, ...]]
```

Finally an event will be send to ewetasker. This event is composed by using crossbar that will that facilitates the sending of tasks. This avoids having to compose a complex http message. The crossbar code have to be composed as it is explained in enable technolies section, for this project it is as shown in Listing 4.4 and Listing 4.5.

Listing 4.4: Crossbar tagcloud event

```
# tag cloud actuator event
def tagCloudEWE():
    url = "http://localhost:8082/call"
    data = {}
    data['procedure'] = 'com.channel.event'
    data['kwargs'] = {
      "user": "administrator",
      "channel" : "TagCloudReciever",
      "event": "Receptor",
      "params" : {
        "TagCloudFlag":"1"
      }
    }
    json_data = json.dumps(data)
    data_channels=requests.post(url, data=json_data)
```

Listing 4.5: Crossbar light event

```
# Lights actuator event
```

```
def lightEWE(inputSent):
    url = "http://localhost:8082/call"
    data = {}
    data['procedure'] = 'com.channel.event'
    data['kwargs'] = {
    "user": "administrator",
    "channel" : "Senpy",
    "event": "SenpyEmotionWanaffect",
    "params" : {
      "SenpyPublicIP":"123.123.123.123",
      "SenpyEmotion" : inputSent[0],
      "SenpyIntensity": inputSent[1]
    }
    }
    json_data = json.dumps(data)
    data_channels=requests.post(url, data=json_data)
```

the controller triggers the event on Ewetasker. It is necessary to implement this in Ewetasker as it is explained in Ewetasker section. Also we have to notice that crossbar works via using an http post request, including some json structured data. As it is shown in the figure we have to indicate the url in wich Ewetasker is going to be hosted. In this particular case, and for the further tests, Ewetasker is going to be deployed in the same host as sensor is. However the system is created and configured in order to have these pieces splitted in different hosts, but in this particular case, Ewetasker domain is going to be localhost, running on the port 8082, that is the port prepared to attend crossbar requests. Also, in this figure, we can check the name of the user channel and event, as well as the params of each one, that are going to be explained in more detail in the Ewetasker section.

4.2.2 Server

The sensor incorporate a server. This server is a simple server developed with python using flask, it is created in order to allow the user to change, in a simple way, the hashtag and also to start and stop the application. This server presents a web front end in html. The web page is composed by a text box to type the hashtag to track and also an alert that will notify the user when the application is gathering tweets. Furthermore it is composed by a selector to choose what algorithm the users want to use and three buttons, two of them to start and stop the application and the other one to shut the lights of in case it is wanted to finish the representation.

Also have the functionality of present a preview of the tag cloud and the result of each tweets analyzed.

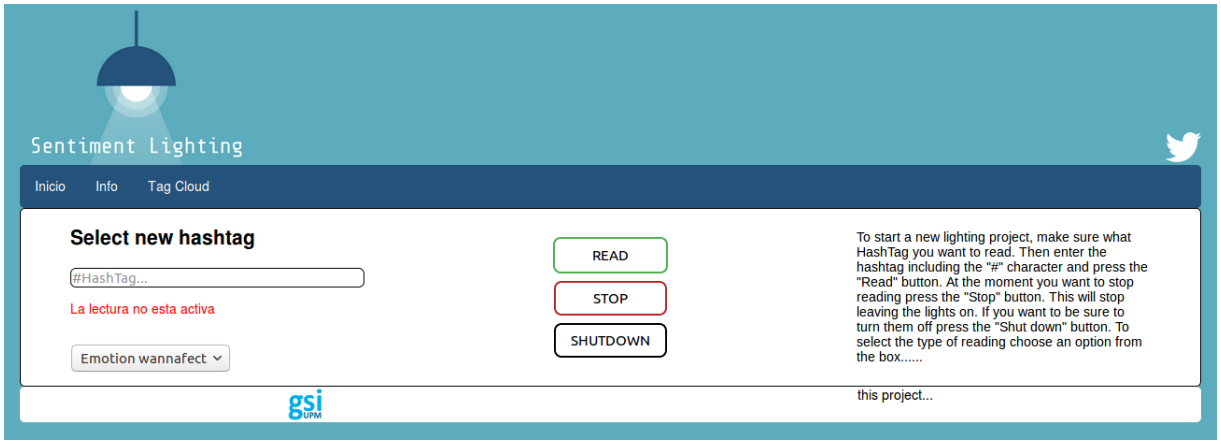

Figure 4.4: Server view

As said in the requirements analysis Twitter is the social network selected to use in this project, so this server will provide the connection with the Twitter API.

For the connection with twitter the tweepy library is going to be used. This is a wrapper that provides access to the entire twitter RESTful API methods. There are two types of authentication in order to access the twitter API. The first one is the application-only authentication, that provide read only functionality. The second one is the application-user authentication that provides more functionality, allowing users to build features that would post tweets. Tweepy implements the second one authentication so we will need to create an application in the twitter API in order to obtain the credentials that allow us to use the API. The credential needed are the followings:

- Consumer key
- Consumer secret
- Token key
- Token secret

On one hand, the consumers credentials identify the account in which the application is going to be connected, what means that it will identify the account that will be used to read and write. On the other hand, tokens, identify the session used to connect to the account, we can manage the users that have access to this account by creating and revoking the API tokens. But in the scope of this project just one sensor is going to be used to read twitter, so it is necessary to generate only one token key and its corresponding token secret.

Once authenticated, the tweepy allow us to track in streaming the desired word or group of words. This way we can follow an entire conversation or a hashtag. So, once the user

provide the basic information for the analysis then the tracker begins to send each tweet catched to the controller in order to be analysed.

The response from the API is the tweet object that is the basic block that compose twitter. Tweets have a long list of attributes such as, “id”, “created at” or “text”, these represent a unique number identificator, the creation date and the basic text, respectively. Also have deeper components that could relate tweet objects between them, for example, “user”, “entities” and “extended entities”. These fields of the object will provide information about the user or other related tweets, for instance we can obtain data from the user, who published the tweet, such as name or screen name, it description, its profile configuration such as background color or other data.

For the scope of this implementation just the plain text of the tweet object is actually useful, however, in order to provide raw material for further investigations or analysis, the whole tweet object of each tweet catched will be stored at Elasticsearch. So, in the server tweets are sent to the controller as it comes from the API.

4.2.3 Topic Analyser

The topic analyzer is connected to the controller. It receives a tweet object and process it. The process of cleaning data is explained in the LDA section. Once cleaned, the data is analysed and then it composes a JSON object as was shown in Listing 4.3.

In next sections LDA and OLD are explained further in detail, in order to know what algorithm to use. Also the parameters are discussed to optimize the algorithm.

4.2.3.1 LDA

Latent Dirichlet allocation is a natural language processing algorithm that provides a generative statistical model that allows a set of observations to be explained by unobserved groups that share words with the data that is going to be explored.[7] In other words, LDA gives a thematic summary of a set of documents. This summary will be discovered by analyzing the different topics that could appear in the text that we provide to the algorithm. The algorithm will discover for us the proportion of this topic as well the main words around the analyzed topics.

To understand LDA we must know that the input data is divided in different input documents. These documents are composed by a word sequence. And, in order to get the topics, LDA composes those documents in a corpus, which is a collection of the documents that are going to be analyzed. LDA will get the different topics in the corpus and will get the different main topics in the whole word sequence via seeking a probabilistic model of a corpus, assigning probabilities to the members of the corpus, and also assigning probabilities

to the different documents that compose the corpus.

The LDA algorithm is based on the idea of a document being analyzed and represented by a set of random latent topics.

The first assumption in LDA is the dimensionality of the problem, this are the k topics that are going to be looked for in the input data. In LDA it is assumed that k is known and that it is fixed in the whole problem resolution.

The graphical model representation of LDA is:

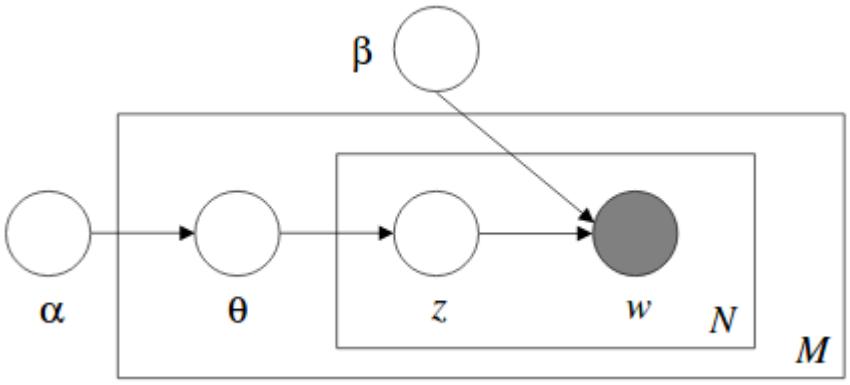

Figure 4.5: LDA diagram [7]

In this picture we can see the three levels of the LDA representation. The parameters alfa and beta take effect in a corpus level and they will be sampled when the corpus is generated. Theta is a document level parameter generated per document. Finally, z and w are parameters in a word level so are generated once at word appears.

The most important variables to understand the in the algorithm are:

- k: the number of topics
- V: number of unique words, which are the basic unit in the input of this model.
- w: a word is represented as a vector 1,...,V, this vector will take value 1 when the word appears and 0 for every other position.
- N: the number of words in a document (w1,w2,...wN)
- M: the number of documents in a corpus.

With the previous information obtained via input data the parameters in the figure are explained below:

- w is the word variable

- z is the topic variable defined as p(zn = i — theta) =theta i
- Beta is a k x V matrix, whose rows represents the multinomial distribution of each topic. It represent topic-word density.
  - This will be calculate with k and V.
- Alpha: Is the representation of the document-topic's density.
  - This also will be calculated with k and V. In gensim it could be set with a prior belief of each topic probability.
- Theta is the topic distribution (of length k) for a document, drawn from a uniform Dirichlet distribution with parameter alfa.

Testing of the algorithm:

LDA will be the model on practice, getting the documents and selecting the number of topics and words in order to obtain the values of the mathematical parameters.

On one hand, the documents will give the input data that is going to be analyzed. With these documents, the corpora will be settled and used to compose the model. On the other hand, the number of topics and words needed will be set beforehand, and as it is mentioned above, they will model the alpha, beta and theta values of the LDA.

Analyzing twitter, we will retrieve a set of tweets by tracking a hashtag in the same way that it will be done in the final project. For this goal, a pipeline is created, and the tweets are stored in elasticsearch. Once the data is obtained it will be dumped into a document. The raw data contained in this document will have to be cleaned for the subsequent analysis.

This whole method is developed using python, a jupyter notebook will be used in order to get conclusions to improve the result of the algorithm by modifying the parameters. This will help get a better result when the algorithm is implemented in the final solution.

Data cleasing:

The data preprocessing uses the nltk python library, as well as other cleansing techniques based on the nature of the data.

First of all, since the data comes from twitter, urls and other similar character strings, have to be removed, the following steps are: tokenizing the text, removing capital letters and lemmatizing the words because it is necessary to analyze it as single items. Ultimately, it is important to remove the stop words, punctuation symbols, and, as we saw earlier, due to the nature of the data, it is important to eliminate the words of the twitter slang such as "rt" or other similar phrases. It is also important to remove the hashtag we are tracking

because it will be a word that every single tweet will have, without adding any relevant information.

It is important to realize that in this part of the project only the number of words and different topics that appear in the input text are important, and no so much the emotion or sentiment that is going to be analyzed via other techniques.

Document:

Once data is cleansed, the next step is to identify which is the document in this problem. The size of each document will be settled. For this paper the best option is to take each tweet as a unique document, since we do not have any evidence of the relation between tweets, we cannot infer what is related between them and is conforming a thread or other kind of twitter document. The other option could be to take the whole dataset as a singular input, but with this decision, we lost a whole level in the algorithm, so that, the results obtained from the analysis will be biased. So, because of this, each tweet will be a single document.

Dictionary:

Once the data is cleansed and the document settled, a dictionary will be created. A dictionary is the reduction of the whole input in order to get the unique tokens that are composing the input.

Parameterization:

The gensim library is going to be used, with the data preprocessed. The next step it is to create the corpora, and then we model the input. With this LDA algorithm using gensim we have these important parameters:

- number of topics
- number of words
- chunkside
- passes

On one hand, the two first parameters are directly related to the way the algorithm will be modeled. On the other hand, chunkside and passes are related with how the algorithm will be trained because, as it is mentioned above, the LDA is a supervised algorithm.

Thus, the chunkside will model the number of documents used and the passes are the number of times the corpus is passed during the training.

So the model is setted and it will not be change.The result will be a set compose by topics represented with a set of words, with this words associated with a weight in function of the association of every single word with its topic.

Results:

The algorithm is going to be parametrized. To get conclusions about this different hashtag have been tracked. The results that are going to be presented on this sections corresponds to two different set of tweets, one of them obtained from the presentation of an sport event ("#oppeninday") and the other from a twitter user timeline (Elon Musk timeline).

First of all in order to not overfit the model the chunkside and passes will be set on 10.

Secondly, the number of words will be set, it is important to understand what means a greater number of words. The more number of words the bigger is the field that topic covers, so it will be most difficult for the algorithm to fit a topic in a document and also there will be more words and the confident of the algorithm will be more disperse. So that we will have a greater number of words with a very low level of confidence, that is to say that many of the words will appear represented (because they are part of that topic) but nevertheless they will not have to appear in many of the documents. Because of this, noise could be added, that means a lot of words that really does not represent the real documents but the topic. So it will be better, for a good representation in the final tagcloud, to get less words. However, if it was necessary to get more items for a better representation (that is to say that the representation is formed by more words not that there are more words per topic), more topics could be added but not more words.

Also in this specific case our basic unit to define what is a document will be a tweet, ie, a tweet is a document. Because of this, to assume documents of more than 240 characters, which are about 45 or 50 words (size of a tweet), is directly absurd. To be more precise when assigning the most important topics the number of words should be approximately between 10 and 15. Therefore for the development of the project and the proper functioning of the algorithm will be set at 10.

Word variation for sport event:

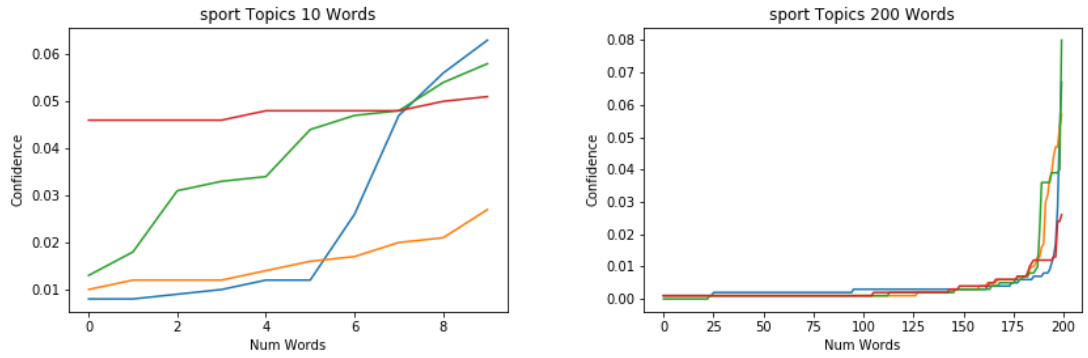

Figure 4.6: LDA: word variation 1

Word variation for Elon Musk timeline:

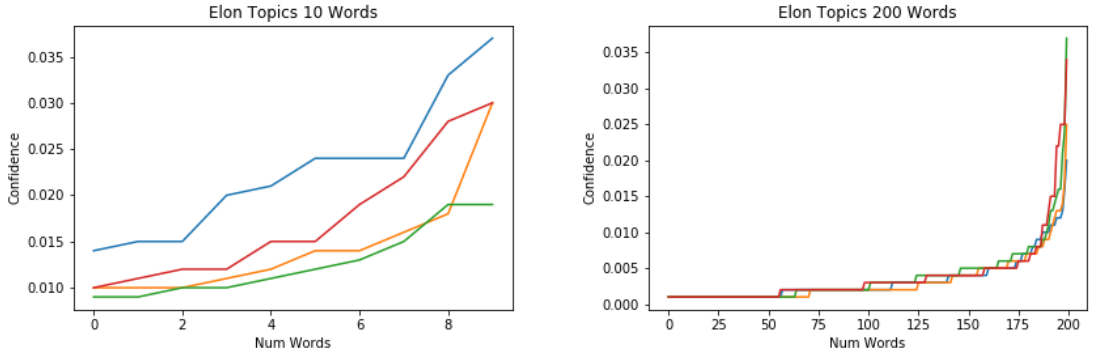

Figure 4.7: LDA: word variation 2

In this figures we can observe that increasing number of words only give us much more words that are almost irrelevant in the document. So that is better a lower number of words.

Finally, we have to set the number of topics for a good representation. The main problem here is the overlapping of topics. As we can see in the next figures if we increase the number of topics it will be overlapped this will cause two situations, the words will be repeated in more than one topic and also the confusion when assigning the dominant topic to a document will increase. This make sense because the topics related to a hashtag are the main topic and a few more.

Topic variation for sport event:

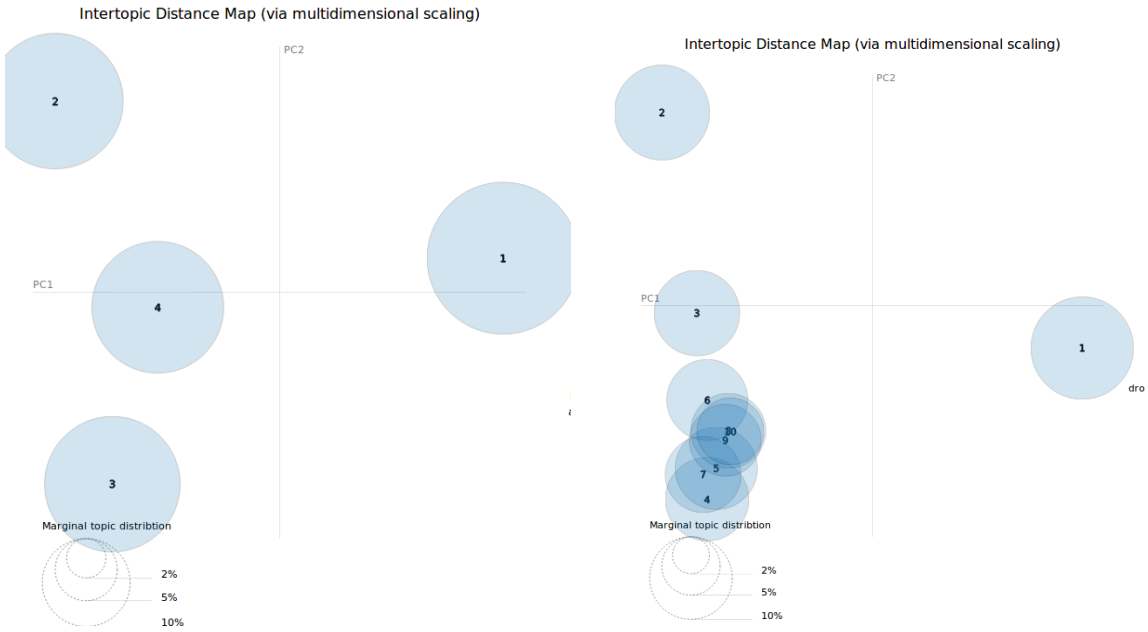

Figure 4.8: LDA: topic variation 1

Topic variation for Elon Musk timeline:

Figure 4.9: LDA: topic variation 2

Due to all the above is deduced the number of topics should be 4 and the words 10. this parametrization will give us the possibility to compose tag clouds such as this:

4.2.3.2 OLDA

The main problem of the LDA is that it cannot be used in streaming aplications, due to the corpora and dictionary must be defined before in order to compose the model. So for the resolution of the problem in the final project OLDA "Online LDA" is going to be used. This algorithm is used in cases that not every data is obtained before the analysis begins and for large datasets. In this project, data are obtained in streaming, so it is going to be a more accurate solution because some of the assumption of the previous algorithm (LDA) cannot be satisfied.

However the parameters and the reasoning to select it also fits this algorithm. This is because the number of words and the number of topics do not depend on the input text to analyze, and also when using online LDA, must be given in advance.

Both LDA and OLDA are based on the assumption that the problem is solved with variational bayesian methods. This methods are a family of techniques for approximating intractable integrals arising in Bayesian inference and machine learning.

So OLDA is an online LDA that is based on online stochastic optimization [16] . This optimization produce the parameters by estimating it. This make the algorithm faster so it is better both to large datasets and streaming data. Due to this, it is not necessary to store the data and the algorithm will pick each document that arrives and discard it once it is used. Furthermore OLDA converges and do its faster than LDA for large datasets.

Example of implementation:

Assuming that data cleansing are the same as in the previously analyzed algorithm. And givin the same parametrization (understanding parametrization as fixing the number of topics, number of wordsand chunkside and passes) an example of the implementation of this algorithm is shown in Listing 4.6 . The results are the same that the obtained with LDA.

Listing 4.6: OLDA implementation

```
# To compose the model
dictionary = corpora.Dictionary(array_words) # array_words contain a
    set of tweets
corpus = [dictionary.doc2bow(text) for text in array_words]
NUM_TOPICS = 4
NUM_WORDS = 10
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics =
    NUM_TOPICS, id2word=dictionary, passes=15)
ldamodel.save('modelOLDA.gensim')
```

```
# For each tweet only upload the data, without recalculate de model
new_corpus = [dictionary.doc2bow(text) for text in array_words] #
    array_word contains a tweet
lda.update(new_corpus)
```

4.2.4 Senpy

Senpy is a gsi project developed in order to allow users to have an own sentiment and emotion analyse server. Senpy is deployed as a service running in a docker container. This will allow us to have an own server evaluating the controller requests. Senpy provides an API so it can be reached via http requests. In this piece of code is going to be used the request python library in order to have a simple connection with the Senpy server due to Senpy does not have a library that wrap the functions of its API.

The request, that is made by the controller, gives as input data a json object that will be sent to Senpy, that object is composed by three fields:

- "algo" that correspond to the algorithm selected to analyse the input message. There are different algorithms implemented as plugins in Senpy. In this project there been selected two, one for each different analysis implemented. The algorithm "Sentiment 140" will correspond to the sentiment analysis and the "Emotion wannafect"
- "i" that corresponds to the input data to senpy as plain text. This is going to be the text to analyse.
- "lang", this corresponds to the language selected for the analysis of each input text. Senpy implements multiple languages, however, this project discard non-english tweets. So language param will be fixed with "en" value that corresponds to english.

As mentioned above, Senpy provide multiple algorithms, but for the scope of this project, we will use two, sentiment 140 and Emotion wannafect.

Sentiment 140 algorithm provides a sentiment analysis that give back a json response compose by the fields that fit the Marl Ontology. The most important properties of this response are the following:

- "marl:haspolarity" that provide the name of the polarity in which the input text has been clasified. Polarity could take three diferent values neutral positive and negative.
- "marl:polarityvalue" that give us a confidence number resulting from the analysis.

As well as "Sentiment 140" algorithm uses Marl ontology, the response of "Emotion wannafect" will be defined by Onyx ontology. The response, when using the emotion analysis,

will return us a field with the most relevant data of the answer. This field is the one corresponding to the property "onyx:hasEmotion" of the Onyx ontology. This property contains an object for each of the emotions that the algorithm analyses that are anger, joy, negative-fear, sadness and disgust. Each of these five objects have two properties as in the "sentiment 140" case.

- "onyx:hasEmotionCategory", that will be one of the previously mentioned (anger, joy ...). In this case every emotions will appear within each response.
- "onyx:hasEmotionIntensity", similar to the confidence number but in this case taking values from 0 to 100. However this is not a confidence number but it is a number of the percentage with which it is shown the amount of this emotion that the algorithm retrieve from the input data.

4.2.5 ElasticSearch

Elasticsearch will be deployed in an individual container in the same way as senpy. This will give persistence to the system allowing the user to store every data analysed in order to use for further applications. Allowing the user to implement post analysis tools or graph tools in order to explore the data, as well as giving the possibility of use the stored data as training dataset for further algorithms configuration improvements.

Thus, Elasticsearch is going to be configured in order to use two different index in which there is going to be stored different information.

- "index", this will be the main index of the system. It will store the full information post analysis as shown in controller section in this document.
- "preIndex", this will store the raw information of each tweet retrieved from the API twitter with the complete tweet information.

4.3 Ewetasker

Ewetasker will be used in this project as a task manager in order to give to the project an IoT structure. Ewetasker gives the possibility to create rules and channels in order to manage tasks. This way, Ewetasker will trigger an action when certain events arrives to the platform. Ewetasker is deployed as an individual piece of the whole project as is shown in the Figure 4.1. It will be deployed by using docker with the package provided by gsi. However in order to be able to incorporate the functionalities, that are proposed in this project, the source code of Ewetasker have to be modified.

First of all it is going to be explained the basic modules used in this project and an overview of the Ewetasker flow. Also, in this section are going to be explained the changes that have to be done in order to incorporate the task manager to the whole system.

4.3.1 Architecture

Ewetasker have two principal modules that provide the whole functionality to the tasker, the server and the web client. The server provides the engine of the system and the webclient provides a web in which we are going to create the rules and channels, once we have defined the necessary changes in the server. This modifications will be done over server modules, such as the actuator, vocabularies and validator.

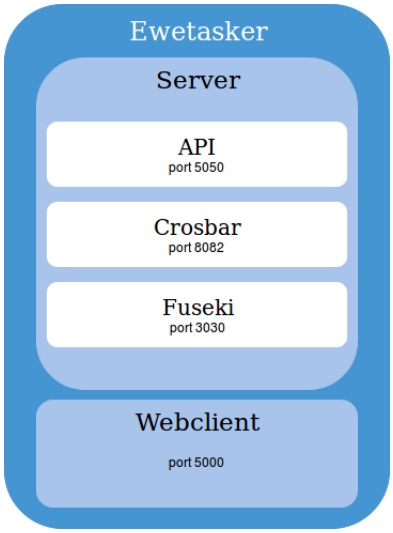

Figure 4.10: Ewetasker modules

Ewetasker is built with semantic web, because of that it needs apache jena (Fuseki) that is a java framework that will provide an API to Ewetasker in order to create graphs with semantics syntax. Here is where we will modify and add vocabularies, to create new classes.

So different components interconnected among them, are needed to be deploy in the same host, every of them will be reach by different ports. A components summary of Ewetasker can be seen in the next figure

Now we will see what happens in Ewetasker, having all the performers, vocabularies and validators defined, when an event arrived. The definition of this flow is in the figure 4.11 that represent this steps:

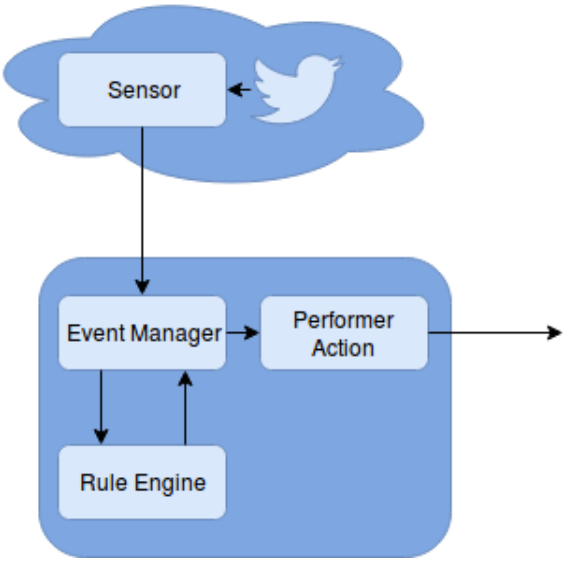

Figure 4.11: Ewetasker flow chart

- Event management: first of all the event arrive via crossbar to the validator, in this module is going to be check the crossbar message if it fits with any of the validators defined, there will be a positive response and the message pass. If not then it will discard the message and response the request with an error.

- Rule engine: with a correct message that match a class the system will check if there are a rule defined if so then the system applied that rule. As a result, if the message match the conditions defined in that rule, it will trigger an action.

- Action triggered: the action is checked in the performer manager where are defined every actions, then it goes to the specific code defined for this action and execute it.

4.3.2 Channels

Ewetasker will trigger an action when certain events arrives to the platform. For this, it will be necessary to create four different channels, two of them to manage the lighting and the sentiment messages and the others to manage the LDA messages and the television that will display the tag cloud. This channel are going to be Senpy, Tag cloud receiver, Smart light and Remote screen.

In order to satisfy the needs of the application, we will define the different characteristic that all of these four channels have to implement.

- Senpy: this is a channel that was previously defined in Ewetasker, but this channel did not cover all the needs of this project, so it was redefined. This channel have to

get two events and no one action. The events will be to receive a emotion analysis and to receive a sentiment analysis. Both of them will have a parameters that will indicate the ip or domain direction of the actuator to active. However they have differences. The emotion detection will have the parameters emotion detected and its intensity, while the sentiment detection will implement the parameters sentiment detected and its confidence.

- Smart lights: this channel has been done in order to manage a smart light, so it has been done in order to satisfy not only this project situation but other possibilities that could involve smart lighting. For the scope of this project, it would be enough to change the light colors but other actions has been implemented. The actions defined in this channels are set RGB color, switch on, switch off and set Brightness.

- Tag cloud receiver: This is the most simple implementation of a channel. It only will have a "set tag cloud" event, so it will be listening when the rule is configured. The only one parameter it is going to implement is a flag parameter so each event of this type recieved will triggered an action.

- Remote screen: Only have an action to send a message to the screen to change. The parameters defined here will be the remote screen IP or domain name, and the IP or domain name where the data to represent will be stored.

4.3.3 Vocabularies

The vocabularies are defined in rdf by using notation3. The vocabularies have to be written by following the channel needs that we defined before. Once vocabularies are defined, the ingestion of data by Fuseki must be carried out. For this, we will use the web portal that Fuseki provides that is the simple way to update the data. Then the channels will appear in the ewetasker portal. In this figure we can see not only the channels defined for this implementation but the whole set of channels that Ewetasker provides.

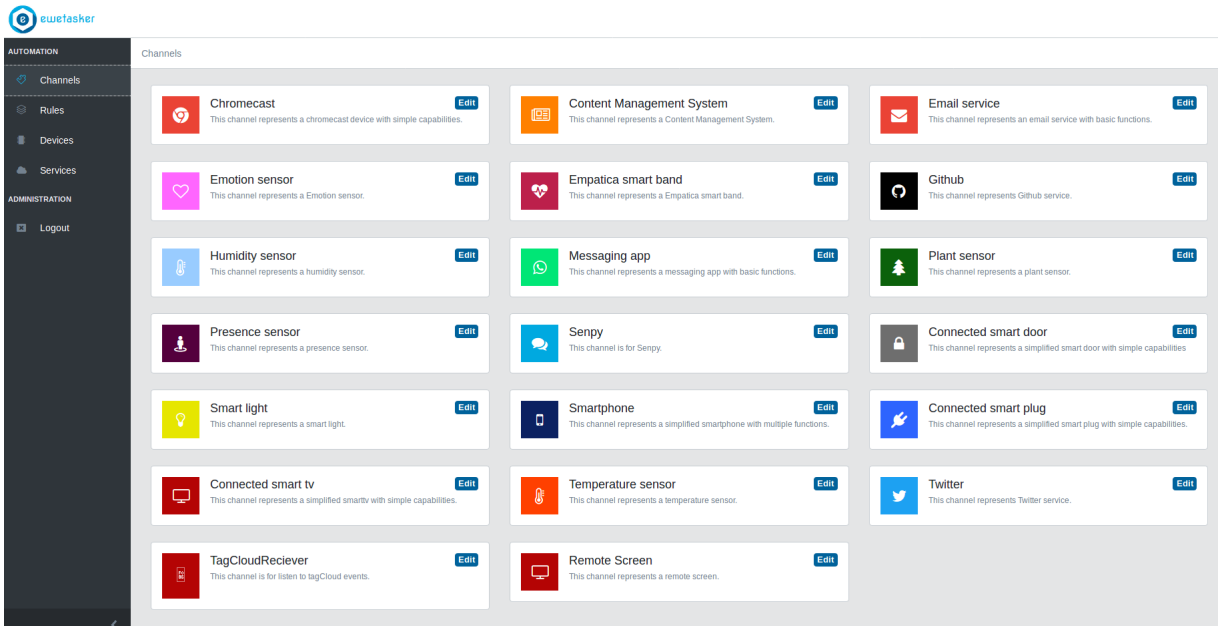

Figure 4.12: Channels represented in Ewetasker

For the definition of the vocabularies there are some prefixes in common that share all the vocabularies defined, including rdf, math, foaf and other useful ontologies that are necesary in order to compose the vocabularies for Ewetasker. Also there will be added Onyx or Marl if necessary.

The example of how the channel is defined in these vocabularies are shown bellow.

Listing 4.7: Senpy

```
ewe:Senpy a owl:Class ;
    rdfs:label "Senpy" ;
    rdfs:comment "This channel is for Senpy." ;
    foaf:logo "fa fa-comments" ;
    dbo:colour dbo:GsiBlue ;
    ewe:generatesEvent ewe:SenpyEmotionWanaffect ;
    ewe:generatesEvent ewe:SenpySentimentVader ;
    ewe:hasCategory ewe-service:Service ;
    rdfs:subClassOf ewe:Channel .
```

Listing 4.8: Tag Cloud receiver

```
ewe:TagCloudReciever a owl:Class ;
    rdfs:label "TagCloudReciever" ;
    rdfs:comment "This channel is for listen to tagCloud events." ;
```

```
    foaf:logo "fas fa-comments" ;
    dbo:colour dbo:Red ;
    ewe:generatesEvent ewe:Receptor ;
    ewe:hasCategory ewe-service:Service ;
    rdfs:subClassOf ewe:Channel .
```

Listing 4.9: Remote Screen

```
ewe:TagCloudSender a owl:Class ;
    rdfs:label "Remote Screen" ;
    rdfs:comment "This channel represents a remote screen." ;
    foaf:logo "fa fa-tv" ;
    dbo:colour dbo:Red ;
    ewe:providesAction ewe:SetTagCloud ;
    ewe:hasCategory ewe-device:Device ;
    rdfs:subClassOf ewe:Channel .
```

Listing 4.10: Smart Lights

```
ewe:SmartLight a owl:Class ;
    rdfs:label "Smart light" ;
    rdfs:comment "This channel represents a smart light." ;
    foaf:logo "fa fa-lightbulb-o" ;
    dbo:colour dbo:Yellow ;
    ewe:providesAction ewe:SwitchOnLight ;
    ewe:providesAction ewe:SwitchOffLight ;
    ewe:providesAction ewe:SetBrightnessLight ;
    ewe:providesAction ewe:SetRGBLight ;
    ewe:hasCategory ewe-device:Device ;
    rdfs:subClassOf ewe:Channel .
```

In the same way the channels are defined, the event actions and parameters of each of theme have to be defined in the same way by using RDF.

4.3.4 Rules

It is necessary to define the rules. For this, we have to know what are the parameters that the system are going to check. Also it is necessary to previously set the channels. Once these parameters are known and the channels are defined, we are going to implement a rule in Ewetasker. There are three different types of channels. First we have the channels that implements actions, secondly there are channels that only implements events and finaly,

both actions and events can be immplemented in the same channel. In this case, there are two channels with only events, that are Senpy and Tag cloud reciever, and also two channels which only implements actions, that are Remote screen and Smart light.

The rules creations is made by using the graphic user interface that provides Ewetasker. So we have to log in the tasker and then go to rules here the tasker shows every channel that is configured via RDF and stored in Fuseki, as have been done before with the channels of this project. By dragging the boxes as is shown in the figure the rules are composed. the system only allow us to use actions as actions and events as events. When we drag a event the system will ask us what are the thresholds that the system have to check in order to take a decision. As well as, when including an action, the system ask us what are the parameter to pass to the performer manager.

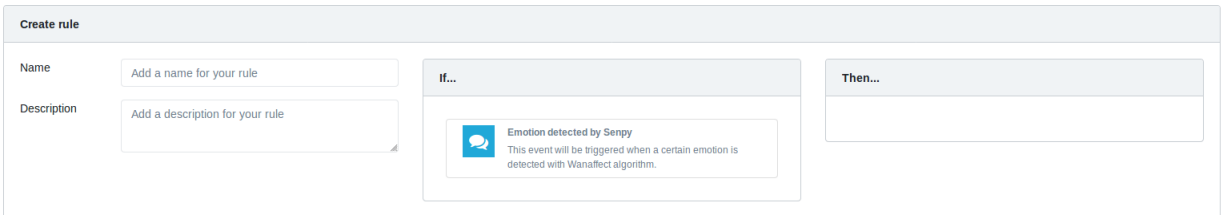

Figure 4.13: Ewetasker rule creation

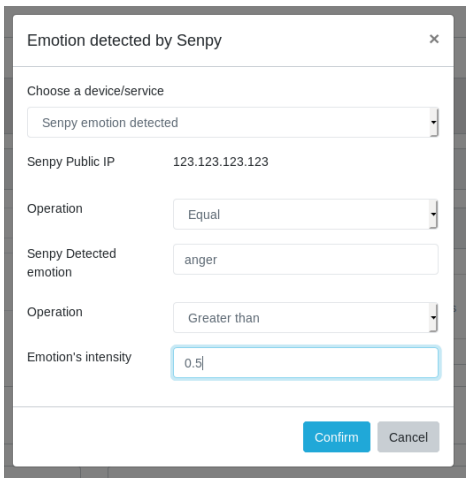

Figure 4.14: Ewetasker thresholds selection

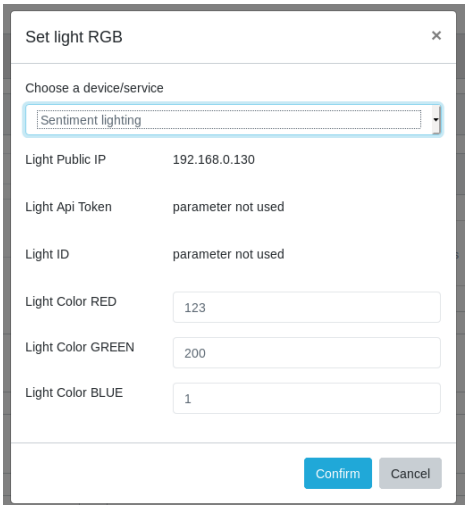

Figure 4.15: Ewetasker action parameters selection

There will be a unique rule in order to implement the tag cloud. This rule is always executed so there is not a threshold. Instead of that, every time an event like this arrive, the system will trigger the action. But this action needs to define an IP or domain as a parameter to send the action. This way,if more than one screen is going to be used, the it

has to be configured a similar rule but with different IP when the system ask for the action parameters.

The case of the tag cloud there are going to be similar rules, the only one difference among rules are going to be when using more than one screen. However the sentiments and emotion are different because is when creating the rule when we select the colors that are going to be represented. This way, it is necessary to create one rule per sentiment and one rule per emotion, selecting the colors, that will represent each sentiment and emotion,at this moment. If more than one actuator is going to be configured we have to include a new set of rules for each new actuator. This is because the sistem select here the colors how made the system more complex when configuring the enviroment, but more flexible when configuring diferent actuators.

4.3.5 Validator

This is the first submodule that a message faces when arrive to Ewetasker. We need to modify it in order to make ewetasker able to receive events. Here it is necessary to create two python modules, one for each channel we will need for this project, in which will be defined a class named as the service they are going to be attending. Here we will indicate the schema that will have the message.

It is important to create the messages with all the params that have the channel so this schema have to match with the params we define when creating the action in the vocabulary.

For every module defined here we have the mandatory parameters channel, event, user, and params. However inside the param parameter we will find the exclusive parameters of each different actions. Also these params could be inclusive, that means that every action of this channel have to implement it or exclusive that will describe those params that could be defined or not depending of the action.

Now, it is been shown how is defined the validator of senpy and tag cloud.

4.3.5.1 Senpy

Listing 4.11: Senpy Validator

```
class Senpy:

    validate = Schema({
    Required('channel'): str,
    Required('event'): str,
    Required("user"): str,
```

```
    Required('params'): {
        Inclusive("SenpyPublicIP", "SenpyPluginGroup"): str,
        Exclusive("SenpyEmotion", "SenpyPluginGroup"): str,
        Exclusive("SenpySentiment", "SenpyPluginGroup"): str,
        Exclusive("SenpyIntensity", "SenpyMeasureGroup"): str,
        Exclusive("SenpyConfidence", "SenpyMeasureGroup"): str
    },
}, extra=ALLOW_EXTRA)
```

4.3.5.2 Tag Cloud

Listing 4.12: Tag Cloud Validator

```
class Tagcloudreciever:

    validate = Schema({
    Required('channel'): str,
    Required('event'): str,
    Required("user"): str,
    Required('params'): {
        Required("TagCloudFlag"): str
    },
}, extra=ALLOW_EXTRA)
```

4.3.6 Actuator

The actuator is the submodule in Ewetasker that provide the functions that will perform the corresponding actions when a rule is evaluated and its reaction has to be triggered. Once the rule is accomplished, ewetasker select what is the properly method to execute. For this there are two modules the Performer manager and the performers, both of them have been done by using Python.

4.3.6.1 Performer Manager

The performer manager has to select what is the performer that has to be triggered. Here there are listed every performer that is able to be used. So here we have two add the code to execute corresponding to the channels that we previously defined that incorporate actions. In the case of this project we have to incorporate and define actions for tag cloud and remote screen.

4.3.6.2 Performers

Finally here we will incorporate the code to execute.

On the one hand we have the lights, here the performer have to get the properties defined in the rules, that properties will be three numbers to define a RGB code and the public ip for the lights. With this information the performer composes a http request, because the lights implements an API so this is the way to change the lights. An example of this implementation is shown bellow.

Listing 4.13: Senpy Performer

```
def change_lights(username,parameters):
    R=parameters['LightColorRed']
    G=parameters['LightColorGreen']
    B=parameters['LightColorBlue']
    url="http://" +parameters['LightPublicIP'] #192.168.0.130
    color="/color?r="+R+"&g="+G+"&b="+B
    request_string =url+color
    requests.post(request_string)
    log.warning(request_string)
```

In the tagcloud class the implementation is more simple because only will need the ip of the actuator that the action have to notify.An example of this implementation is shown bellow.

Listing 4.14: Tag Cloud Performer

```
def change_tag(username,parameters):

    requests.post(parameters['tagCloudIP'], data={'data':"dame datos"})
    requests.post(parameters['screenIP'], data={'data':json.dumps(json.
        loads(r.text)[0])})
```

4.4 IoT actuators

In order to achieve an innovative and creative way to represent data, there will be two different actuators that will plot the data.

First of all it is wanted to represent sentiments and as it is explained in this document a way to representing it are colors. Also, there are common the lighting projects, such as building lighting, furthermore lights are an important tool to create uniques environments

to specific events such as concerts, catwalks and more. So, we can represent sentiments via color lighting by changing colors when new sentiments are detected.

The other option,that is going to be used in this project, to get a different representation is by composing a tag cloud. For this the main topics analysed and the words that composed it (with it corresponding weights) are sent to the tag cloud actuator. So the words are going to be displayed in a screen. For this a server it is going to be deployed with a web page that will be opened on a screen.

4.4.1 Lights

This is the actuator that will represent the sentiments. For this it is necessary an IoT device that have to be able to receive orders from Ewetasker and change lights when the system requires it.

This actuator is a device composed by two basic modules, so there is an compatible arduino board and then, connected to it, a led strip that can represent RGB colors.

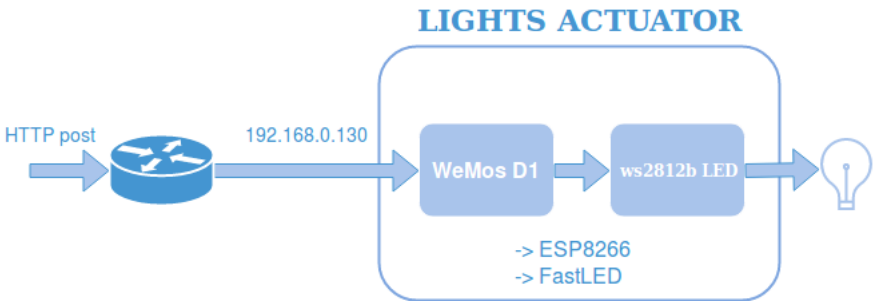

Figure 4.16: Lights actuator diagram

So there will be a “Wemos D1” microcontroller board, that is compatible with arduino, connected to a wifi router by using ESP8266 library. ESP8226 brings support for Arduino environment chips. It will allow us to write sketches using familiar Arduino functions and libraries, and run them directly on ESP8266, with no external microcontroller required.This will provide a wifi connection to the board so it will be provisioned with a static IP. Secondly the lights are connected to the board and also it is installed the fastLED library. This library will allow us to define an API, this way the Leds will be able to be reached via HTTP. FastLED it is not a specific library for the “ws2812b LED strip”, that is the one selected for the resolution of this project, but it is a library for easily efficiently controlling a wide variety of LED chipsets.

To sum up, this will define a device that will provide an static IP and an API with a set of simple action that the actuator will implement. The most important actions are :

- IP/color?r=¡R¿g=¡G¿b=¡B¿, in otder to set the RGB values of the color selected to

be represented

- IP/off to turn off the lights
- IP/on to turn on the lights (using the last color and value)

To sum up this actuator its going to receive a HTTP post with 3 values of RGB, sent by Ewetasker. Then it changes the color of the LED strip. This way the sentiment is going to be represented by the color in the LED strip

4.4.2 Remote Screen

The last representation is the tag cloud representation, it will be composed by a blank web page that will be filled with words when the OLDA analysis is completed. Ewetasker will notify the actuator in order to get it know that a new set of words is generated. Then the actuator implements a method to retrieve the full set of words to represent from ElasticSearch. Finally the words will be represented and the tag cloud will vary with the next set of words that arrives.

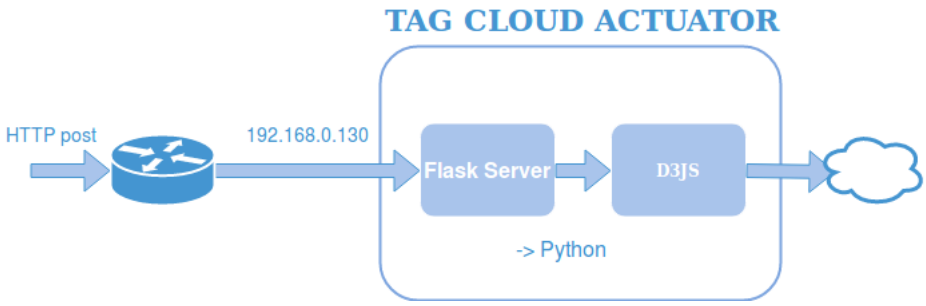

Figure 4.17: Tag cloud actuator diagram

The basic application is developed with python and uses flask to set up a server that will provide an API to contact the actuator. The action that provide is:

- IP:5002/postJSONData

Then it will use d3 cloud library to make the representation of the words and also the transitions when a new set arrives because this will make possible the visualization of what of the words gains importance and what of them disappear.

CHAPTER 5

Case study

5.1 Introduction

In this section it will be presented selected cases of study or bussines cases, in witch the system developed during this project are be able to have a real use.

The main utility for this technology are the moments in which we want to get a unique atmosphere in an event, defined by the public that are related with that event . May exist a lot of cases in which we want to know the opinion of a particular group of people who surround a particular topic, for instance there are a lot of applications oriented to know the intention to vote in elections, or other type of events where people's opinion s important to make a decision. By gathering information in social media, graphs and maps that expresses opinions are made. This kind of application combines the technologies used in big data analysis and the representation of this information via techniques to make data understandable for humans.

Tracking a social media resource, such as a hash-tag or picture's comments, becomes crucial when the opinion of people is the main issue to take decisions but it is also important when we want to customize the experience for the attendants. Customization can change the experience of people.

The first information we can get by analyzing coments it is the opinion and as a result we can obtain the summary of the opinions and impressions that attendants can get at that

particular event in that particular moment. But by the use of sentiment analysis next step is achieved. The feelings transmitted by the situations that are happening in every moment can be gathered so we can know how the event is flowing.

For instance if data is gathered during a symposium, enough information can be obtained about the speakers. For example, what are the moments of the speech that got more expectation and by contrast when people is getting bored or angry with this information we can know when and why people are more receptive or what are the parts of the speech when the attendance are more receptive.

Every this data is directly refer to people's opinions, we obtain here what are they strictly saying about whats happening, but there are more information under the surface. By measuring the feelings stemmed from opinions, more sincerely information is obtained. The same post analysis that is done for text analysis can be done in this case. This way we obtain a sentiment analysis.

On the one hand we have the sentiment analysis, this way data about the attendasnce's feelings is gathered, and we can measure it and represent it in a dashboard, with barcharts, maps and other methods as always.

On the other hand, we want to achieve the personalization of an event and for this we can match that feelings with sounds or music, but for the scope of this paper, feelings and colors will be matched.

So the unique atmosphere we want to achieve can be obtained via lighting.

Colors represent basic sentiments so we can easily match the result of sentiment analysis whit the color that sentiment represents. Also, opinion can be obtained and represented via basic colors such as red, green and white representing bad opinions, favorable opinions and neutral opinions respectively.

Colors have been selected in this work in order to get a more powerful representation of the statisticals sentiments and emotions retrieved from social media. This is because of colors primary relations with sentiments and emotions. Also, it is known that throughout the history color schemes has often been used to define different feelings, seasons and ceremonies depending on one's culture and origin and besides colors can affect how we feel subconsciously.

Colors have psychological effects on individuals. According to the various researches, the color that surrounds us in our daily lives has a profound effect on our mood and on our behavior. Not only the natural light, but the colors of clothes, buildings and lighting have the power of changing people's mood.

The design of an environment through a variety of means such as temperature, sounds, layout, lighting, and colors can stimulate perceptual and emotional responses in people and affect their behavior.

In order to understand the psychological Properties and Symbolism of color. We have to realize that there are two main concepts over this:

- Symbolism of color: on the one hand, colors may be used as symbols in order to express messages, emotions, or other kind of feelings or signals. The colors may be used to represent ideas or concepts. The symbolistic use of color usually may change depending on the culture, but also there are cultural and universal basis. For instance red is usually used to represent power, love or passion, and we unconsciously associate this ideas when seeing it.

- Psychological properties: on the other hand, colors may be used to condition people's behavior. because primary colors are strongly related to the body, mind, feelings and emotions so there are effects that colors produce to humans. For instance red, which have the longest wavelength, is a strong and powerful color that have the capacity to grab first the attention and also have the capacity to get people more energy or get it more stress.

Based on this two definitions we know colors may be used in two different ways.

Firstly, we can create compositions based on people's sentiments that aim to create fully colored spaces that represent that feelings. Including all the feelings we can retrieve such as anger, joy or others, and represent it via cognitive lighting showing the color palette based on emotions we obtained. Also rejecting or approval atmospheres may be created, retrieving the emotion and representing it via basics colors, red and green for instance.

However, other execution may help us to use other cognitive lighting, via forcing the results to alter people feelings. We can move the general sentiment via using colors based on the information we retrieve, for example if the tone of a conversation is going to become more and more anger, we would change colors and use them in order to get more relaxing environment to the situation.

To sum up we may represent the sentiments that we know people are feeling or try to change them based on this data, via using of color.

In order to know, what colors fit better with what emotions, there are many studies talking about that issue but the most simple representation is the Robert Plutchik wheel of color. This theory consider that there are eight primary emotions in a nature environment that may be represented by eight primary colors. Plutchik relate this colors to a evolutionary level in order to know how the colors affect on a psychological way.

This wheel is represented in this figure.

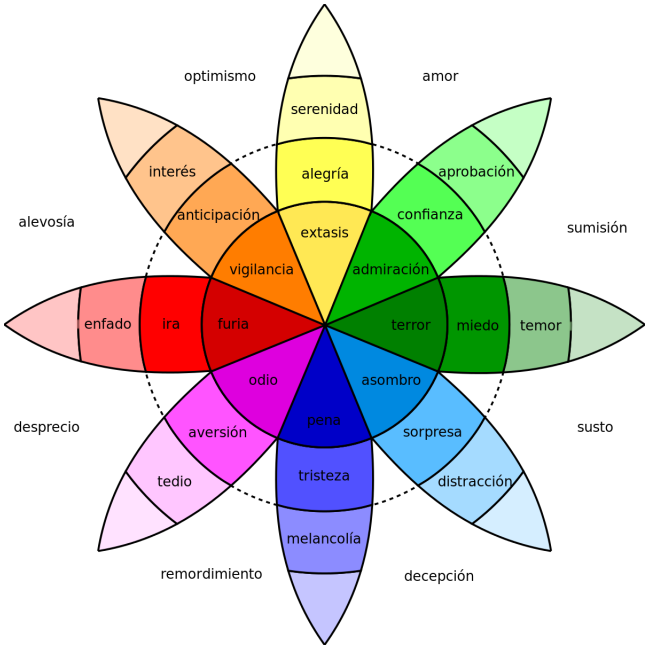

Figure 5.1: Sentiments and colors

We can easily match the results of this research with the results that we could get of any emotion analyzer based on AI. This color wheel could be used to influence on people's behavior but also could be used to represent that feelings. However emotion representation do not have to use a pattern like this and other possibilities may be used, because in that case it will be a personal decision about what color may be use, but not in the case of influencing on people behavior in which we have to understand why this colors change human perception.

For the scope of the project colors must be used only to represent opinions and emotions, and it will allow users to pick the colors they want, however the basic configuration will include this, and it is recommended to follow it, to do the colors makes sense.

5.2 case of study

A set of case of study is going to be presented in order to give a vision about how the final solution would look like in real cases. For this, the cases are going to be presented, as well as each particularity, presenting the possible changes that could be implemented in case it was necessary, in order to develop these cases.

Here, the real application of the technically solution is going to be presented, being the most simple and the first the case quot;Room atmospherequot;, that is the one that better fits the solution.

5.2.1 Room atmosphere.

A room is going to be used to host an event. The user has to configure every item necessary to start with the lighting, in this case let's assume that the user wants to represent the opinions that people post about a new film recently announced to be released in theaters.

Then the user will select what colors he wants to use. As it is explained before, there are many possibilities to obtain a correct set of colors. The user only wants to represent opinions so colors will not represent emotions therefore a more simple combination of colors can be used.

So, the user decides to use the white color to represent neutrality, red color to represent negativity and green color to represent positivity. Now the user has to implement this by using this system. For this the user describes the necessary rules in Ewetasker, one for each different color represented.

Also, the user needs to prepare the place where the event will take place by setting up the actuators and the sensor.

5.2.1.1 IoT deployment

This case is intended to give the perfect atmosphere to a single room in which an event is taking place. Also, this approach is the basic case that fits better with the whole solution.

For the representation it is necessary:

- Deployment of the Ewetasker server.
- Deployment of both actuators, screen and lights.
- Deployment of the sensor

This case will allow us to understand the deployment of the whole system and the real interconnection of each piece. Each one of the pieces define bellow will be deploy in separated hosts that will have whole connectivity between them.

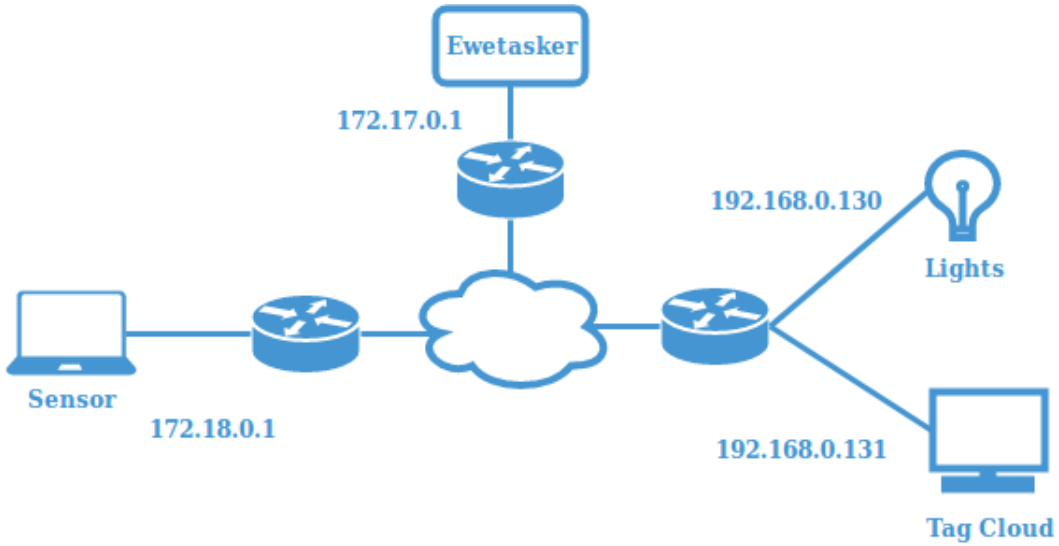

Figure 5.2: Example of deployment

Every one of the pieces that have been shown above, will be dockerized in order to make the deployment of the system easier. Also, this structure will allow the deployment of the pieces in cloud with the proper connectivity but, for the example that is being shown, it will be deployed locally.

5.2.1.2 Services and Devices

First, Ewetasker needs to define services and devices. These are the logical representation of physical sensors and actuators. These logical representations are created on the channels defined previously in the architecture, and will come from channels, but it will have its parameters properly defined in order to be linked with the devices that will trigger the events and fulfill the actions physically.

As it is explained in the architecture, to cover the full requirements of the system, there are four channels. Two of the themes correspond to the sensor, these are the smart light channel and the screen channel. Also, there are two channels that will correspond to the sensor, these are the tag cloud receiver channel and the senpy channel. For the point of view of Ewetasker these are going to behave like two different sensors, however they are implemented in the same piece of code because they will analyze the same input data, even if its contributions to Ewetasker are completely different.

So, in ewetasker there will be defined as services, the channels that provide event and that will correspond with tag cloud receiver channel and the Senpy channel. Also, they will be defined as devices he smart light channel and the screen channel. So, the next services must be created in ewetasker:

- Senpy, due to the nature of the information that is going to be retrieved by this service there could be two different types of information, sentiments and emotions. In this case of use we make the assumption that the user wants to represent sentiments. Because of that the channel must be configured like this:
  - The base channel must be selected.
  - It will be named properly in order to be readily recognizable to work with it in Ewetasker. The name selected for this device is "Senpy sentiment detected".
  - We have to set the sensor ip, that is, asit is shown in the Figure 5.2, 172.18.0.1.

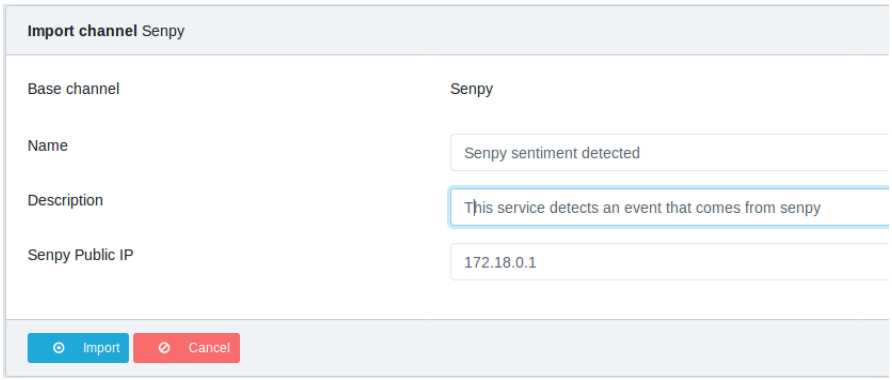

Figure 5.3: Senpy service

- Tag Cloud receiver, that it was defined simpler, because of its needs. So it will not have specific parameters and only will have a name,"Tag Cloud Receiver", and a description.

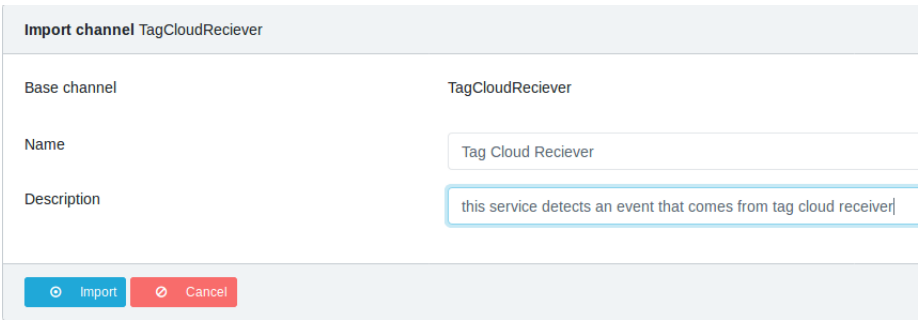

Figure 5.4: Tag cloud service

Finally, the next two devices have to be created:

- Remote screen device that have the parameters:
  - As all the devices and services, it has a name and a description. the name is going to be "Remote Screen".

- On one hand it has to be set the IP where is the data. In this case the IP to get the data is 172.18.0.1
- On the other hand, we have to set the IP where the remote screen is allocated. In this case it will be 192.168.0.131

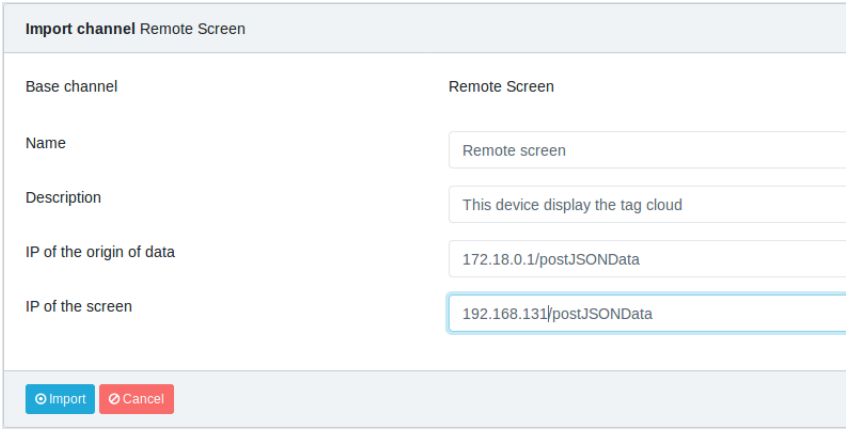

Figure 5.5: Remote screen device

- The last one item to configure are the lights, it will also have a name and a description. The name, this time, will be "Sentiment lighting" and we also have to provide The IP of the device that will be 192.168.0.130.

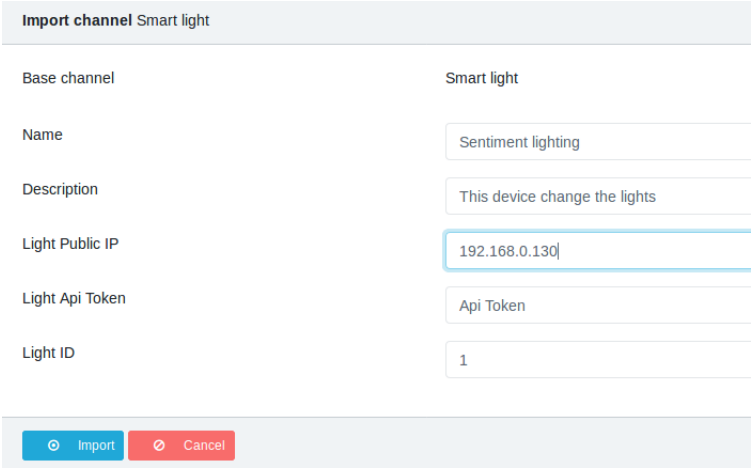

Figure 5.6: Lights device

5.2.1.3 Rules

Here we can notice that there is going to be two different set of rules. On one hand there will be one rule in order to create the tag cloud that will be composed by two channels, the tag cloud receiver and the remote screen channels.

On the other hand, there will be rules to set up the lighting system. The rules necessary for this case will be composed by two channels, Senpy and Smart light.

Furthermore, when selecting the channels, it will be necessary to select the devices and services created before. Then, the user has to pick the new rule option and then create each rule following the next steps:

- Name and description, it has to be set in order to have a most clear set of rules in Ewetasker. These fields are not mandatory but strongly recommended in the rule's creation.

  For the two types of rules that concern us there will be different ways to create it. So, for tag cloud rule, we will put a name and description that refers to which rule it is, without the need to make a clear reference to its operation, since it is a single rule and will be easy to identify.

  Nevertheless, for lighting rules we need to be more specific. The name of each rule will be the emotion detected, and the description will be a text clarifying what emotion detects the rule and what color it uses to represent.

  So the name of the rules will be:

  - When detecting a new tag cloud:
    - Name: Tag cloud detected
    - Description: New tag cloud detected.
  - When detecting Neutrality
    - Name: Neutral sentiment detected
    - Description: sentiment neutral and light white.
  - When detecting Positivity
    - Name: Positive sentiment detected
    - Description: sentiment positive and light green.
  - When detecting Negativity
    - Name: Negative sentiment detected
    - Description: sentiment negative and light red.

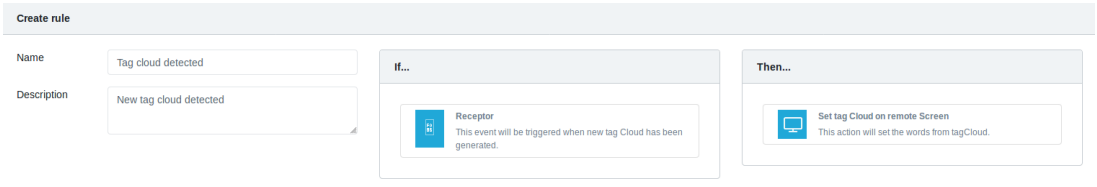

Figure 5.7: Rule creation with Tag cloud

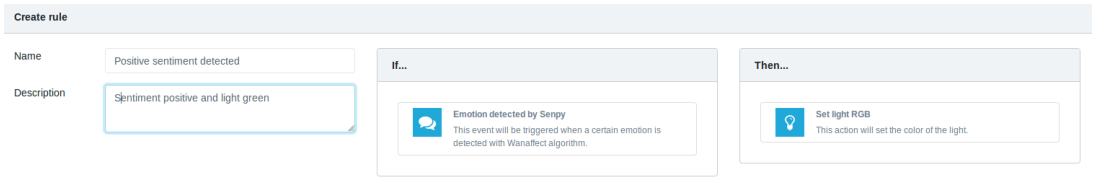

Figure 5.8: Rule creation with positive sentiment

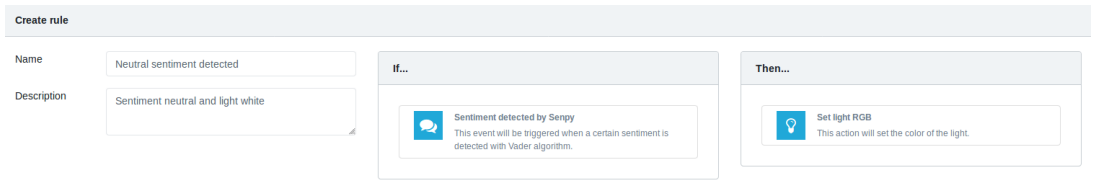

Figure 5.9: Rule creation with neutral sentiment

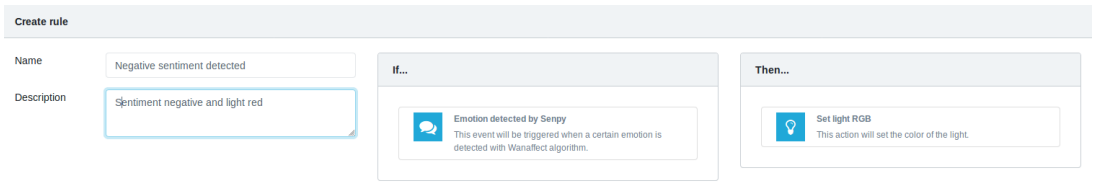

Figure 5.10: Rule creation with negative sentiment

Furthermore, the description could be more specific, for instance, giving information about the specific actuator that will cover the resolution of the event. but this will not be necessary in a case like this in which there will be only one actuator of each one of them.

- If box, the "if" box has to be filled with the channel that triggers the action, it means to say that the channel is responsible for the event management. So, in the case we concern, it is going to be used "emotion detected by senpy" and "Tag cloud receiver". When using "Tag cloud receiver" we only have to put the rule as equal to one, because this service works as a flag. But when using "emotion detected by senpy" we have to configure the specific thresholds whose accomplishment triggers the response. For this, we have to fill both of the following fields:
  - Senpy detects emotion: we have to simply indicate which emotion triggers each rule. So, we have to text respectively neutral, positive and negative
  - Confidence: in order to get only the messages that the algorithm clearly identifies if it is a good option to select a big confident, in this case 0.75 that will preserve the system to change the lights when it is not clear that an emotion is cached.

Also it is important to select the service (sensor) created before.

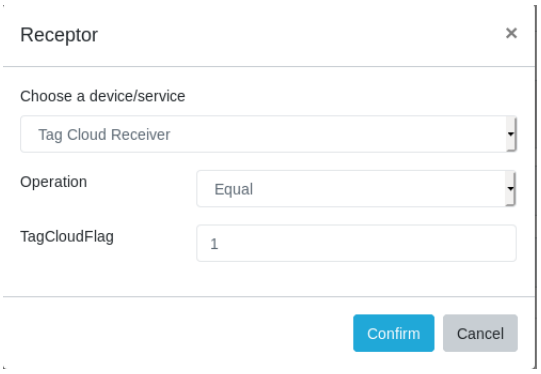

Figure 5.11: Example with Tag cloud

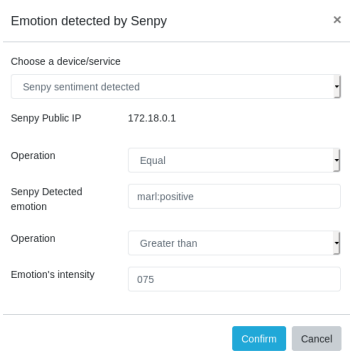

Figure 5.12: Example with Positive

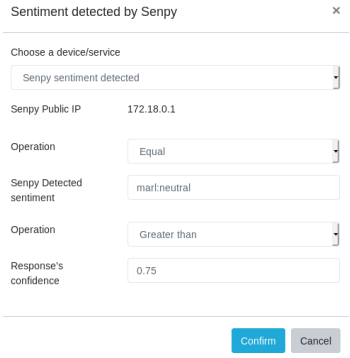

Figure 5.13: Example with Neutral

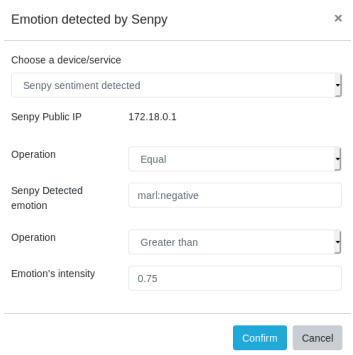

Figure 5.14: Example with Negative

- Then box, the "then" box will correspond with "Set tag Cloud on remote screen" and "Set light RGB".the first one does not have any field to configure in this step but not the second one. "Set light RGB" is the channel that has been developed to communicate with the lights' endpoint. When deploying it, it will ask the user what the color is he wants to represent for each emotion. So, it is here when the user has to decide what colors he wants to use and in case he wants to change it, he will have to delete the proper rule and then create another one with the new color. Also, this will allow the user to change, without constrains, the color he needs and do it without stopping the application, however in the case that this occurs, every event that they destroy rule the cover will not have any effect while until new rule is up. So, as far as this case concerns, the configuration of this box will have following parameters:
  - Positive rule: with R as 0, G as 255, and B as 0.
  - Negative rule: with R as 255, G as 0, and B as 0.

– Neutral rule: with R as 255, G as 255, and B as 255.

Also it is important to select the services (actuators) created before.

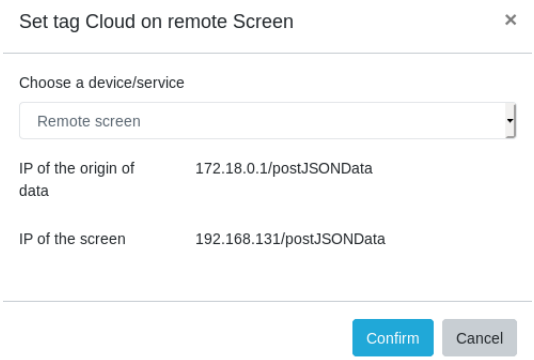

Figure 5.15: Example with Tag cloud

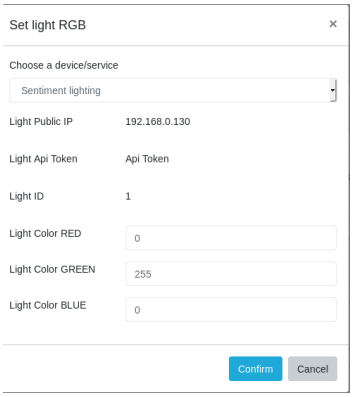

Figure 5.16: Example with Positive

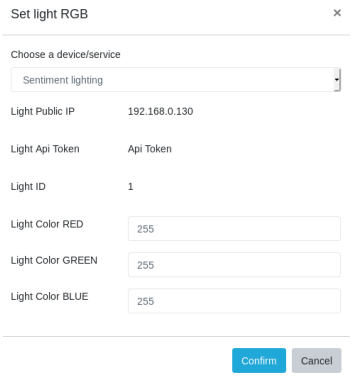

Figure 5.17: Example with Neutral

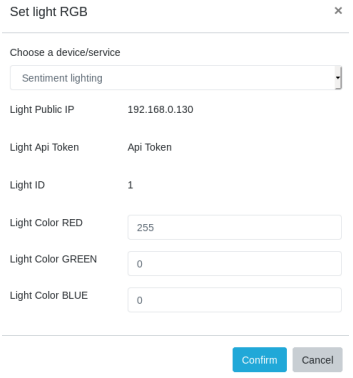

Figure 5.18: Example with Negative

5.2.1.4 Activation of the system

Now that the actuators, sensors and Ewetasker are set up and configured to fit the necessity of this particular event, the user has to start the system, that will automatically set the words and colors detected in each situation.

For this, the user has to connect to the sensor and start tracking an event. In this case it is required to follow the release of a new film. The user has to know when the movie will be announced and what is the starting hour in order to begin with the reading.

When the event starts then the user types the "hashtag" in the text box, selects "sentiment140" and presses the start button. At this moment the sensor is catching tweets, processing it and send it to Ewetasker.

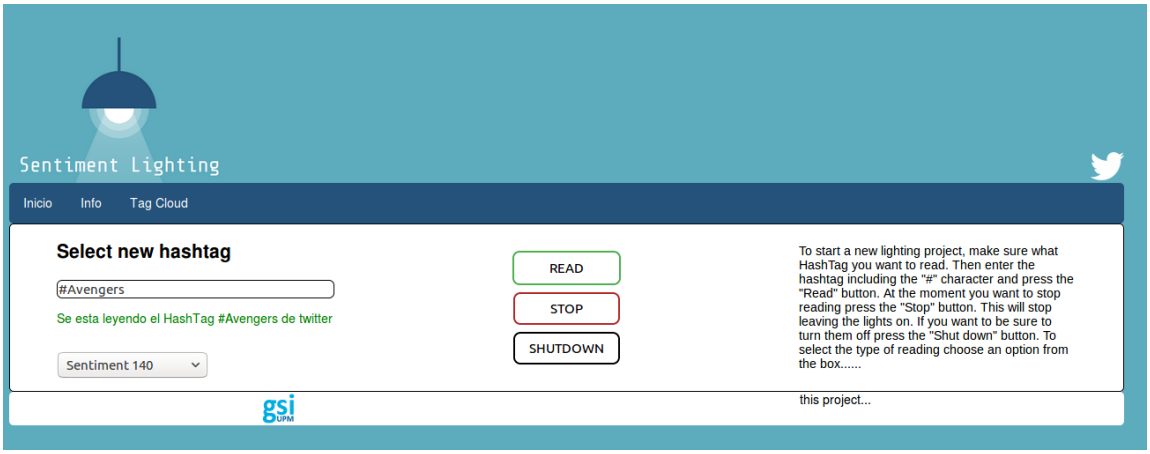

Figure 5.19: Sensor configured to read a film using sentiment 140

5.2.1.5 Results

With this configuration, the lights will be changing every time a tweet arrives, but the presentation of the tag cloud will take a gap, because first of all it has to train the model and create the dictionary to represent. Once the first tagCloud is created it will be refreshed when a new tweet arrives, this will create a dynamic representation combining the lights and the tag cloud that will vary with time.

Figure 5.20: Example with Positive

Figure 5.21: Example with Neutral

Figure 5.22: Example with Negative

In the pictures above, there are shown three different states of the system, defined for this case of study. Each one of them are representing one different sentiment as were defined. These lights will change with each tweet that arrives and will give a fancy environment to understand the feelings with a different powerful representation.

Furthermore, we can see the full representation that is composed by the lights and also the tag cloud. This case was defined to track a hashtag talking about fills so the cloud will contain words related with it and as said before, these words will vary its size depending on how much it appears in the four main topics founded in the whole four topics.

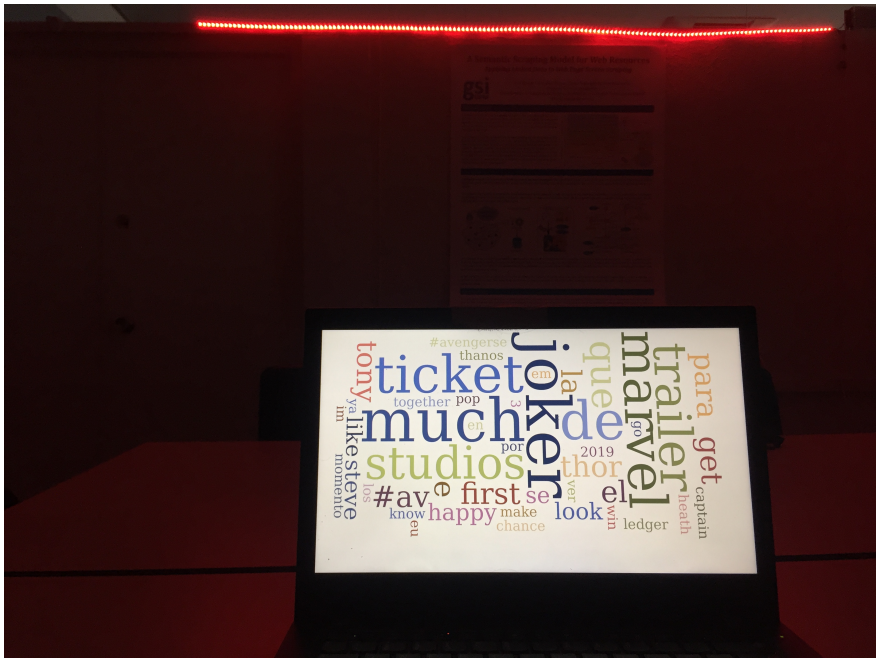

Figure 5.23: Static Tag Cloud

Figure 5.24: Tag Cloud transition

In these pictures we can see also the static word cloud representing the most representative words and the moment of the transition between the representation of one wordcloud to another.

5.2.2 Concerts or other events.

There are certain types of events that have been created to arouse emotions in the people who attend them. These events usually try to obtain shocking performances with light and sounds. This usually creates a sentiment storm in the attendance. In order to prove the power of IA, it could be an option to manage these emotions in order to obtain more personal representations.

We can find cases like this in events such as concerts or fashion shows. So, a fashion show is one of the more representative situations, for this deployment, because this kind of event are created to shock the attendance and get opinions about the clothing pieces presented on it.

This case will allow the event organizer to fit the lights to the sentiments catch from the audience.

5.2.2.1 IoT deployment

In this type of events, additional complications have to be faced due to the lighting systems that are used commercially. Typically, when lighting an event, the lighting technician, in charge of the lighting of the event, uses a command wing. This is a table with the capability of control and program a light session. This will provide a higher control to manage the multiple light sources that are disposed at an event like this.

A professional command wing is thought to be managed by a technician and most of them do not have IoT functionality. However, this kind of system usually has to provide mechanisms for remote control allowing users to connect tables among them. In order to solve this connectivity problems, the two principle algorithms that this table implements are:

- DMX is an electronic protocol used in lighting technology for the control of professional lighting devices, allowing communication between light control equipment and the light sources themselves.
- MIDI that is a technical standard that describes a communications protocol, typically used in electronic instrument but also in lighting systems.

For the scope of this case, it is easier to understand and implement the MIDI protocol. The MIDI format has 8 groups of commands that can be sent or received by devices, such as note off, note on, polyphonic key press, control change, program change, monophonic key press, tone adjustment, system exclusive commands. These will be interpreted by the command wings as specific actions that have to be previously configured in its memory. A

MIDI message usually has two or three numeric fields. The first defines the command to be executed and can have any value between 128 and 255. For instance, the 144 it is used to turn on or off a note. The second and third numbers include the data necessary for the execution of the command defined with the first number and depending on which command they accompany, they can mean the number of the key pressed, the intensity of the key pressure, the program number, the control value, etc. For the scope of this project we have to know that a lighting table will interpret these messages in a different way. To explain this case deeply it will be supposed that is going to be used a table like the "grandMA2" that is widely extended professional command wing. This table can manage different channels these will represent other devices that the table have to connect with. So, the first number defined before have to be the number of the channel used, the second number represent the color stored on the table memory and the third one will represent the intensity of these color.

Furthermore, an IoT interface has to be developed. This interface has to include a server listening in the same way that the two principal actuators that have been defined in architecture section. But with the difference that this interface has to be deployed over a computer that has installed the official program that controls that controls the command wing. Also, to connect the interface with the program it has to be defined a virtual MIDI channel. Finally, the computer and the command wing are connected via usb. The final solution will look like this:

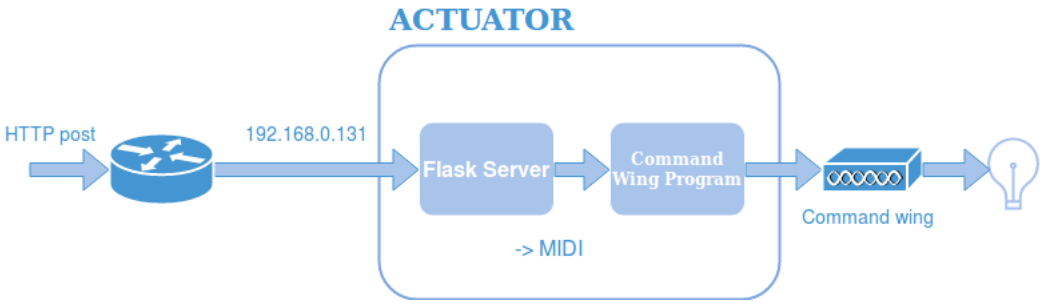

Figure 5.25: Actuator schema

With this new actuator the final user will have to prepare the same deployment as it was deployed in the first case, that means that the user has to select a hashtag for his event, this way the attendance will post its comments there and the system will follow the same routine as it is described in the previous section. In addition to the previous case, the command wind has to be configured, for this it is important that the colors defined in the specialized program matches the colors defined in Ewetasker. This means that the light technician has to know what are the MIDI channel that the interface developed uses, and fits the session stored at the lighting table memory in order to make the system chose the

appropriate colors.

5.2.2.2 Results

Finally, by connecting the table with the actuator, as said before, we will get a system like the one that is shown below.

Figure 5.26: Tag Cloud transition

And the party could be illuminated using sentiment posted in social media. The results of this case of study implementation will be like in the pictures shown below.

Figure 5.27: Example with Positive

Figure 5.28: Example with Neutral

Figure 5.29: Example with Negative

5.2.3 Other opinion resources and events.

Also there are a lot of projects in witch this kind of technology could fit. For instance projects where buildings are illuminated with diferent colors. There are projects that usually provides static lighting but also projects where buildings are light based on the interaction that people have with panels, remote controllers or other direct devices. This kind of projects are getting more and more visibility nowadays. And also with sports events, nowadays, it is more and more inportant to provide a personal configuration with lights,sounds and others, as we can see with the construction of large stadiums covered with panels that change color or large colored spotlights that illuminate them.

Finally this is not only a system thought for big event. But it develops better with small places in which the illumination, as well as the nature of the information, is controlled. In this way it would be given in private parties or for instance a lecture or presentation.

CHAPTER 6

Conclusions and future work

In this chapter the conclusions extracted from this project, thoughts about future work and problems faced while developing the project are described.

6.1 Conclusions

In this project we have developed a complete end-to-end application with added functionalities, that allows users to interact with them in a simple way. The solution offers a complete system of information extraction from a social network, followed by an analysis of the feelings reflected by the extracted information. With the result of the analysis and the selected colors, two different and intuitive ways of representing this information have been composed. The lights and the tag cloud.

The way in which the system treats the data has been split, providing the whole application with an IoT behavior. In this case, the sensor has been responsible for the connection with the chosen social network, which has been twitter because of the multiple facilities it offers when extracting its information. This sensor has also been equipped with the system's intelligence. This sensor is the one that has the functionality of analysis and storage of the data, leaving the information clean to travel through the network.

This decision has been taken because one of the objectives of the project was to be mod-

ular and that every piece developed could be extracted if desired, even though the power of the application is in this sensor. This will allow future users to evolve this idea by capturing different types of information and not just those that have been initially proposed in this paper.

During the development of this project, multiple decisions have been taken. One of the most important was that the data analysis is done in the sensor. This made the system simpler than if the analysis was done in cloud. Also, this will allow the development of new modules which could have different analysis. This provides the system with the capacity of easily evolving. Another important decision is the inclusion of the task manager, which gives more complexity to the project. However, this is the most important piece in the system, because it provides the engine to manage the tasks. This gives us a much more modular system. Furthermore, it allows the connection between items. The inclusion of rules also gives us the capacity of changing the scenario easily. It makes the system more adaptable to any environment. It has been possible to demonstrate that the project is suitable for real life situations. This could seem complicated at first, but this kind of implementations are very eye-catching for the general public. Furthermore, the implementation of this technology is very economical, and it can make it very interesting for the advertising sector.

One negative aspect found in this project is that it is very complicated to achieve an accurate analysis of the data input. This will be improved when the maturity of the sector is be greater. However, the error margin of the analysis is small enough for the scope of this project. Important limitations in the system can be also seen. Such as the limitation on accessing social networks. The reason why we have only taken twitter has been discussed during the work. However, it would be interesting to have access to more sources of information. Although, as we have seen in Design Decisions, trying to include other social network brings up various problems. In addition, there is currently an important awareness of the data usage. This could cause rejection from some users.

To sum up, this project fulfils its objective as an AI project over an IoT environment. In addition, it can be implemented in final environments as shown in the “Case Study” section of this project. Despite it being useful as it is, this system can be improved, and the possibilities that it offers to various sectors are almost endless.

6.2 Problems faced

During the development of this work, some problems has been faced. This has made us to make some decisions that finally has been implemented in order to get the better result possible. Some of the most importants are going to be listed in this section:

- Connection of multiple and different devices: This is one of the main problem faced in this project. In order to solve it Ewetasker have been use to provide an IoT architecture.

- Properly managed of data: In order to get a better analysis from the data obtained, there is important to clean properly the data. Here has been face to models of cleansing. On one hand a most simple way to clean the data for the Senpy's analysis. On the other hand it had to be implemented separately the cleansing of data to OLDA. These proposal of cleansing has been deeply explained in the architecture section.

- algorithm election: When in this paper the data are analyzed to obtain the main topics, it is done in order to know the weight of the words that are going to be represented later. For this reason different algorithms have been evaluated. LDA and OLDA have been the two most suitable for the project and have their differences in the way these algorithms work. However, the main difference is the way in which they are able to analyze streaming data in which OLDA has been proved to be a better option. However, it is complicated, and still not resolved, the fact of having a complete dictionary of words before the analysis begins. To solve this fact, the algorithm has to spend several cycles compiling this dictionary with new entries to later consolidate a model.Once the model is created the OLDA starts, it will be updated and the variation in the topics represented has been shown in the tag cloud.

- Design and development of the different pieces: As it as shown before, there are notable differences among the structure and design decision of each one of the pieces that are been developed for this project. This is because each one of the pieces are thought to be deployed in very different endpoints. That can prior seems to be a problem but thanks to the IoT structure all these pieces can communicate thakns to the task manager, that will manage the communication among theme.

- Ewetasker adaptation. Due to Ewetasker only implements a few basic channels, to give the full implementation of this system multiple channels have been created or modified to properly define and conect services and devices.

6.3 Achieved goals

This work has achieved the goals proposed or it that are, deploy an IoT system, desing and develop its actuators andsensors, configuration of the algorithms and tools necesaries in order to analyse text to get topics and emotions and integration of the system with a task manager.

All of these points in order to obtain a system that is able to provide cognitive lighting and environment.

6.4 Future work

- Find other ways to represent data: The principle line that could be evolved in future releases are the creation of new actuators that provide different ways to represent data including, for instance sound system that change the music and rhythms depending on the input data. Also more traditional ways to represent data, composed by barcharts, piecharts or others, could be implemented in order to check and evaluate on a simple way the results.

- Evolve the existing representation of data: As it is explained in the architecture section, the representation of the topics is made via one screen that brings together the four topics. This could be improved in order to show the four topics separatly. On one hand it could be achieved via tagging tha data this way the actuator could get the posibility to represent each topic with diferent colors. On the other hand the topics could be separated in four screen for this the ewetasker channel must be changed but not the actuators, this is more expensive and complicated but also more visual.

- Develop an engine to recognize and automatically get the colors: get the colors to represent the emotions depending on some input photographies such as brand logos. Matching the most representative colors with the sentiments or emotion that most accurate represent.

- Integration with more than one social network: For the scope of this project only twitter is tracked for the reasons explained in sections before. However it is interesting to find other ways to obtain important information from other social networks.

- Integration with methods that analyse non-written data: Ewetasker can incorporate other kind of sensor such as proximity sensors based, for instance, on beacons. Also Other ways to obtain measures for the place we want to light such as facial recognicion in order to get emotions and feelings from people faces.

APPENDIX A

Social, Ethical, Economical, and Environmental Impact

This appendix shows the social, economic and environmental impact, as well as the ethical and professional responsibility.

A.1 Social Impact

In the last few years we have seen IoT and Artificial Intelligence projects have been gaining notoriety. The number of the products that are able to connect to a network has been growing exponentially. In 2020 it is expected that 10 billion devices are going to be connected to a network and the amount is going to be duplicated by 2015 [18]. This gives us a vision on how these kinds of projects are welcome by society and are having an important acceptance. This is because these projects make our society a more optimum place and they make people's day to day simpler. The same cases take place with AI [17], whose market's financing is growing the most compared to other markets that are stagnating. In this case, even if there are data cession problems, the users accept giving away their data in order to obtain better results in fields like publicity or other recommendations that companies can make to them, and other similar projects.

Generally speaking, the society usually adapts easily to these kind of projects, and

the one presented in this project should not be an exception. Facing a public that is more adapted to technology compared to previous generations, the search of new ways to use technology could have a good welcome among the society, and specially with younger generations. Specifically speaking about an illuminating project, like the one explained in this paper, that has been implemented thought the years in all kind of events, like concerts or football matches, etc. This project has just taken this simple idea of illuminating a room and has taken it a bit further by implementing AI and IoT in other to improve the experience of the final user.

A.2 Economical Impact

Economically speaking, this project is not expensive when looking at the great analysis and impact that is has on final users. Companies could really benefit from it as people would enjoy the experience and it would give the company a lot of great publicity. When looking at the impact that this could have on the labor force, this would be a complementary job from the main activity of the company, for example, if we use it during a fashion show, by using the light and the analysis it gives the designer and the director an instant idea of the thoughts that people have about the collection, letting them know, more or less, how well or bad it is going to sell.

A.3 Ethic Impact

The most important ethical question that has been faced during the development of this project is how to approach the management of the data that is used thought the project. Since it is a project that contains both AI and Iot, the treatment that the data receives is one of the main activities of the project. Because of this, the GDPR has to be taken into account [21]. In order to meet this standard the following steps have to be taken into account:

- Anonymize the data, so none of the text processed can be directly linked to the person who wrote it.
- Taking only data that is useful for the project's development, in this project only the message is needed, it does not matter who wrote it, hence the extra data that we do not need is not going to be stored.
- To not concede the data stored to third parties. This data could be used to improve the system but they cannot be given away or used outside of the academic scope of the project.

Everything is being looked at from an ethical point of view, since Twitter is the entity that is directly responsible for the data, and who agrees on with the users what data can be diffused and used by third parties through Twitter's development API. Nevertheless, during the development of this project this ethical use and treatment of the data has been considered very important.

A.4 Environmental Impact

The development of this project does not have a major impact on the environment, since it has had a small scope. Hence, the environmental impact that it has could be reduced to the impacts derived from the energy consumption of the devices and the environmental cost of the production of the devices used such as LED lights, the Arduino board, the screens and the laptop used. Besides, it is important to highlight that, if the advice that is given in the project is followed, modern devices such as LED lights have been used in order to reduce the energy consumption compared to older versions that have a much higher energy consumption.

APPENDIX B

Project Budget

In this appendix we can see an overview of the project's cost. In order to approach it, we are going to propose two different visions, the first one is the cost of the development, and the second one, is the cost of implementing it.

B.1 Project's development

For the development of this project we have to take into account the facilities that have been provided by the GSI laboratory. However, the cost that some of these devices have can be estimated. The laboratory has disposed the network, the led lights and the Arduino board. Additionally, for the software development and the deployment of the testing devices a laptop with the following characteristics has been used: 1TB hard disk, corei5, 3.20GHz and 8GB of RAM. There fore these are the approximate costs:

- LED lights 10 Euros
- Arduino board 20 Euros
- Laptop 800 Euros
- Network cost 50 Euros/month
- Labor 1500 Euros/month

Assuming that the time extension for the complete development of the project would be approximately a month, we can estimate that the total cost would be around 2380 Euros.

B.2 Deployment of a real-life case

Following the example used in the "Use Cases" section, we are going to calculate the cost of a deployment for a room environment. We are taking into account that the deployment is made on a Cloud in order to avoid a high initial cost.

The following hosts are needed, we have given them prices that are the ones from AWS:

- Host sensor: 1CPU and 2 GB of RAM 18.97 Euros/month
- Ewetasker host: 2CPU and 4GB of RAM 37.94 Euros/month

A person that would be in charge of the deployment, it will take that person around 8 working hours that would suppose a cost of 80 Euros. Assuming that, to have a professional deployment, more than one actuator like the one in the previous case, are going to be needed. In this case we are going to suppose 8 actuators with an individual cost of 30 Euros, a total of 240 Euros.

Finally, the total cost would arise, approximately, to an initial investment of 350 Euros and a monthly cost of 56.91 Euros.

Bibliography

[1] . MA lighting Manual. http://help2.malighting.com/Page/grandMA2/grandma2/en/3.7, 2019. [Online; accessed 18-June-2019].

[2] Andrew Morton. MIDI: MIDI.js. https://www.npmjs.com/package/midi, 2019. [Online; accessed 18-June-2019].

[3] Armin Ronacher. Flask server Documentation. http://flask.pocoo.org/docs/1.0/, 2019. [Online; accessed 18-June-2019].

[4] Avoleoo. Color meaning and psychology. https://graf1x.com/color-psychology-emotion-meaning-poster/, 2019. [Online; accessed 3-May-2019].

[5] Barry Babin, David Hardesty, and Tracy A Suter. Color and shopping intentions. *Journal of Business Research - J BUS RES*, 56:541–551, 07 2003.

[6] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[8] Yin Zhou Quan Zhao Xin Geng Deyu Zhou, Xuan Zhang. Emotion distribution learning from texts. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 638–647, 2016.

[9] GSI. Ewe ontology. http://www.gsi.dit.upm.es/ontologies/ewe/, 2017. [Online; accessed 18-June-2019].

[10] GSI. Ewetasker repository. https://lab.gsi.upm.es/ewe/ewetasker_webclient, 2017. [Online; accessed 18-June-2019].

[11] GSI. Ewetasker webpage. https://ewetasker.readthedocs.io/en/latest/ewetasker.html, 2017. [Online; accessed 18-June-2019].

[12] GSI. Marl ontology. http://www.gsi.dit.upm.es/ontologies/marl/, 2019. [Online; accessed 11-January-2019].

[13] GSI. Onyx ontology. http://gsi.dit.upm.es/ontologies/onyx/, 2019. [Online; accessed 11-January-2019].

[14] GSI. Welcome to senpy's documentation! https://senpy.readthedocs.io/en/latest/, 2019. [Online; accessed 10-June-2019].

[15] GSI. What is senpy? https://senpy.readthedocs.io/en/latest/senpy.html, 2019. [Online; accessed 10-June-2019].

[16] Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864, 2010.

[17] Jacques Bughin, Eric Hazan, Sree Ramaswamy, Michael Chui, Tera Allas, Peter Dahlström, Nicolaus Henke, Monica Trench. Artificial intelligence the next digital frontier? https://www.mckinsey.com/~/media/McKinsey/Industries/Advanced%20Electronics/Our%20Insights/How%20artificial%20intelligence%20can%20deliver%20real%20value%20to%20companies/MGI-Artificial-Intelligence-Discussion-paper.ashx, 2017. [Online; accessed 18-June-2019].

[18] Knud Lasse Lueth. State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/, 2018. [Online; accessed 18-June-2019].

[19] PS Lokhande, Fankar Aslam, Nabeel Hawa, Jumal Munir, and Murade Gulamgaus. Efficient way of web development using python and flask. 2015.

[20] Mike Bostock. D3: Data-Driven Documents. https://github.com/d3/d3, 2019. [Online; accessed 18-June-2019].

[21] Official Journal of the European Union. on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN, 2016. [Online; accessed 18-June-2019].

[22] Miguel Ricardo Pérez Pereira and Anderson Sebastián Salinas. Control de sistemas de iluminación dmx (digital multiplex) basado en señales audibles. *Revista Vínculos: Ciencia, tecnología y sociedad*, 14(1):45–54, 2017.

[23] Robert Plutchik. The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. *American Scientist*, 89(4):344–350, 2001.

[24] Radim Řehůřek. Gensim Library. https://radimrehurek.com/gensim/, 2019. [Online; accessed 18-June-2019].

[25] Radim Řehřek and Petr Sojka. Gensim—statistical semantics in python. *statistical semantics; gensim; Python; LDA; SVD*, 2011.

[26] Francis Rumsey. *MIDI systems and control*. Butterworth-Heinemann, 1994.

[27] Shay Banon. Elasticsearch Documentation. https://www.elastic.co/guide/index.html, 2019. [Online; accessed 18-June-2019].

[28] w3. Resource description framework (rdf). https://www.w3.org/RDF/, 2019. [Online; accessed 14-April-2019].

[29] Nick Qi Zhu. *Data visualization with D3. js cookbook*. Packt Publishing Ltd, 2013.