

PROYECTO FIN DE CARRERA

Título: Diseño y desarrollo de un servicio de análisis de sentimientos basado en una infraestructura de Big Data

Título (inglés): Design and development of a sentiment analysis service based on a Big Data infrastructure

Autor: David Moreno Briz

Tutor: Carlos A. Iglesias Fernández

Departamento: Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente: Mercedes Garijo Ayestarán

Vocal: Tomás Robles Valladares

Secretario: Carlos Ángel Iglesias Fernández

Suplente: Marifeli Sedano Ruíz

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



PROYECTO FIN DE CARRERA

**DESIGN AND DEVELOPMENT
OF A SENTIMENT ANALYSIS SERVICE
BASED ON A BIG DATA INFRASTRUCTURE**

David Moreno Briz

Septiembre de 2014

Resumen

Esta memoria pretende llevar a cabo la descripción de un proyecto que ha consistido en la combinación de una gran variedad de tecnologías con el objetivo de ofrecer un servicio de análisis de sentimientos y emociones.

El servicio de análisis de sentimientos y emociones desarrollado en el proyecto, denominado Sentiment and Emotion Analysis Services (SEAS), ha seguido la especificación NLP Interchange Format (NIF) y es compatible con los vocabularios semánticos Marl y Onyx, que se intercambian en formato JSON for Linking Data (JSON-LD).

Para hacer uso de SEAS, se ha desarrollado un cliente denominado Sentiment and Emotion Analysis integrated in GATE (SAGA), un plugin de GATE (una herramienta de análisis de texto basada en anotaciones), desarrollado para realizar análisis de texto basandonos en los resultados ofrecidos por SEAS así como por otros servicios web de análisis de sentimientos y emociones que respeten el mismo formato, como son los ofrecidos por Eurosentiment. SAGA será usada junto con otros recursos de procesamiento ofrecidos por GATE.

Por otro lado, se ha explorado el uso de SEAS desde infraestructuras Big Data. Por ello se ha llevado a cabo el uso de herramientas Big Data, de computación distribuida y procesamiento de datos como son Flume, Hadoop y Pig, para desarrollar un sistema de extracción y procesamiento de información proveniente de Twitter, aprovechando las ventajas que nos ofrece el uso de servicios de análisis en la nube como SEAS.

Con esto, se han presentado las conclusiones extraídas del trabajo, las posibles líneas de continuación del proyecto, así como los siguientes pasos en cuanto a desarrollo y aprovechamiento de la plataforma.

Palabras clave: Tecnologías semánticas, Big Data, NIF, Marl, Onyx, Sentimientos, Emociones, Java, GATE, Servicios Web, Flume, Hadoop, Pig, Eurosentiment, MongoDB

Abstract

This thesis aims to undertake the description of a project that involved the combination of a variety of technologies to provide a sentiment and emotion analysis service.

We provide a web service called SEAS, which aims to present different sentiment and emotion analysis services that follow NIF's API and returns its results according to Marl (Sentiment) and Onyx (Emotions) ontologies in JSON-LD format, among others.

To use SEAS, a client called SAGA has been developed. SAGA is a GATE (an analysis tool based on text annotations) plugin developed to provide text analysis based on the results offered by SEAS as well as other web based sentiment and emotion analysis services with the same result format as SEAS, like the ones Eurosentiment offers. SAGA will be used along with other processing resources offered by GATE.

Finally, the use of SEAS from Big Data infrastructures has been explored. We have combined Big Data, distributed computing and data processing tools, such as Flume, Hadoop and Pig, to develop a system for extraction and processing of information from Twitter, using the advantages offered by the use of analysis services in the cloud as SEAS.

With this, we have presented the conclusions drawn from the work, the possible lines of continuation of the project and the next steps in terms of development and progress of the platform.

Keywords: Semantic technologies, Big Data, NIF, Marl, Onyx, Sentiment, Emotions, Java, GATE, Web Services, Flume, Hadoop, Pig, Eurosentiment, MongoDB

Agradecimientos

A mi familia y amigos.

Contents

Resumen	V
Abstract	VII
Agradecimientos	IX
Contents	XI
List of Figures	XVII
List of Tables	XXI
List of Acronyms	XXIII
1 Introduction	1
1.1 Context	3
1.2 Master thesis description	4
1.3 Master thesis goals	6
1.4 Structure of this Master Thesis	6
2 Enabling Technologies	9
2.1 Overview	11
2.2 Opinion mining	11
2.3 Marl: An Ontology for Opinion Mining	12
2.4 Onyx: Describing Emotions on the Web of Data	13

2.5	Eurosentiment	15
2.6	NIF 2.0	16
2.7	GATE	17
2.7.1	GATE Developer	17
2.7.2	GATE Embedded	18
2.8	Hadoop	19
2.8.1	HDFS: Hadoop Distributed File System	20
2.8.1.1	HDFS daemons	20
2.8.1.2	HDFS blocks	20
2.8.2	Hadoop MapReduce	20
2.9	Flume	22
2.10	Pig	23
2.10.1	UDFs	24
2.10.2	Elephant Bird	24
2.11	Conclusions	24
3	Requirement Analysis	25
3.1	Overview	27
3.2	Use cases	27
3.2.1	SEAS	27
3.2.1.1	Actors dictionary	27
3.2.1.2	Use cases	29
3.2.1.3	Sentiment analysis	30
3.2.1.4	Emotion analysis	31
3.2.1.5	New services	32
3.2.1.6	Test and demo	33
3.2.2	SAGA	34

3.2.2.1	Actors dictionary	34
3.2.2.2	Use cases	34
3.2.2.3	Sentiment and emotion annotations calling SEAS	36
3.2.2.4	Opinion annotations	37
3.2.2.5	Sentiment and emotion annotations calling other NIF services	38
3.2.2.6	Update SAGA	39
3.2.2.7	New PR	40
3.2.3	SEAS-Hadoop	41
3.2.3.1	Actors dictionary	41
3.2.3.2	Use cases	41
3.2.3.3	Get data	42
3.2.3.4	Sentiment and emotion analysis over data	44
3.2.3.5	New script	44
3.2.4	Conclusions	45
4	Architecture	47
4.1	Introduction	49
4.2	SEAS: Sentiment and emotion analysis services	50
4.2.1	Input format	51
4.2.2	Sentiment analysis	52
4.2.2.1	Output format	53
4.2.3	Emotion analysis	55
4.2.3.1	Output format	56
4.3	SAGA: Sentiment and Emotion Analysis integrated in GATE	59
4.3.1	PR: Processing resources	60
4.3.1.1	Predefined Sentiment Annotation PR	60
4.3.1.2	Sentiment And Emotion Analysis Calling SEAS PR	61

4.4	SEAS-Hadoop: Sentiment and emotion analysis over a Big Data infrastructure	65
4.4.1	Flume	65
4.4.2	Pig	66
4.4.2.1	UDFs	67
4.5	Conclusions	68
5	Case study	71
5.1	Introduction	73
5.2	SEAS	74
5.2.1	Call SEAS using a command line shell	74
5.2.2	Call SEAS using Eurosentiment playground	76
5.2.3	Call SEAS using the demo available at GSI	77
5.2.4	Call SEAS to analyze videos in real-time	79
5.3	SAGA	81
5.3.1	Corpus	82
5.3.2	Finance sentiment analysis calling SEAS	82
5.3.2.1	Validation of the sentiment analysis	87
5.3.3	Other sentiment analysis services calling SEAS	90
5.3.4	Emotion analysis calling Onyxemote	91
5.3.5	Sentiment analysis calling Eurosentiment services	93
5.4	Hadoop for financial analysis	95
5.4.1	Using Flume to obtain data from Twitter	95
5.4.2	Data processing and sentiment analysis using Pig	97
5.4.2.1	Hadoop vs GATE	100
5.5	Conclusions	101
6	Conclusions and future lines	103

6.1	Conclusions	105
6.2	Achieved goals	105
6.3	Future work	106
A	Installing and configuring SEAS	109
A.1	Installation	109
A.1.1	Requirements	109
A.1.2	Installation steps	110
A.2	User manual	111
A.2.1	Command line interface	112
A.2.2	Using Java	113
A.2.3	Web service interface	114
B	Installing and configuring SAGA	117
B.1	Installation	117
B.1.1	Requirements	117
B.1.2	Installation steps	117
B.2	User manual	118
B.2.1	Sentiment and emotion analysis calling SEAS and Eurosentiment . .	118
B.2.1.1	Example of use - Sentiment analysis over a finance domain	120
B.2.1.2	Example of use - Emotion analysis using Onyxemote	121
B.2.1.3	Example of use - Eurosentiment services	122
B.2.2	Predefined Sentiment Annotation	123
C	Installing and configuring SEAS-Hadoop	127
C.1	Installation	127
C.1.1	Requirements	127
C.1.2	Installation steps	128

C.2	User manual	129
C.2.1	Using Flume to obtain data from Twitter	130
C.2.2	Sentiment and emotion analysis using Pig	130
Bibliography		133

List of Figures

1.1	System general picture	5
2.1	Class and Properties Diagram for the Marl Ontology [1]	13
2.2	Marl, example of use [1]	13
2.3	Class and Properties Diagram for the Onyx Ontology [2]	14
2.4	Onyx, example of use [2]	15
2.5	NIF Core Ontology [3]	17
2.6	Example of configuration of a PR	18
2.7	GATE Embedded APIs [4]	19
2.8	HDFS architecture [5]	21
2.9	Hadoop master/slave architecture [6]	22
2.10	Flume architecture [7]	23
3.1	SEAS use case	29
3.2	SAGA use case	35
3.3	SEAS-Hadoop use case	42
4.1	General Architecture	50
4.2	Sequence diagram for sentiment analysis	53
4.3	Sequence diagram for emotion analysis	56
4.4	Runtime parameters configuration for negative annotations	61
4.5	Sequence diagram for Predefined Sentiment Annotation PR	62
4.6	Runtime parameters configuration for sentiment annotations	64

4.7	Runtime parameters configuration for sentiment annotations	64
4.8	Sequence diagram for Sentimen And Emotiont Analysis Calling SEAS PR .	64
4.9	Sequence diagram for Flume	66
4.10	Sequence diagram for Pig	67
5.1	The Eurosentiment playground	76
5.2	The Eurosentiment playground with a POST request	77
5.3	SEAS demo	78
5.4	Real-time video sentiment analyzer options	80
5.5	Real-time video sentiment analyzer performance	81
5.6	Sample of documents inside a GATE corpus	83
5.7	Sample of a document loaded in GATE	84
5.8	Sample of Sentiment and emotion analysis calling SEAS PR configuration .	84
5.9	Sample of Sentiment and emotion analysis calling SEAS PR configuration .	85
5.10	Example of an analyzed negative document	87
5.11	Sample of Annotation Set Transfer PR configuration	88
5.12	Sample of Predefined Sentiment Annotation PR configuration	89
5.13	Example of an analyzed negative document and its real polarity	89
5.14	Negative corpus Quality Assurance	90
5.15	List of sentiment analysis services	91
5.16	Sample of Sentiment and emotion analysis calling SEAS PR configuration .	92
5.17	Sample of Sentiment and emotion analysis calling SEAS PR configuration .	92
5.18	Sample of an analyzed emotion document	93
5.19	Eurosentiment services	93
5.20	Runtime Eurosentiment parameters 1	94
5.21	Runtime Eurosentiment parameters 2	94

A.1	Web service interface	114
A.2	Web service interface in action	115
B.1	New processing resource	119
B.2	Finance example 1	120
B.3	Finance example 2	120
B.4	Finance example result	121
B.5	Emotion example	121
B.6	Emotion example results	122
B.7	Eurosentiment services	123
B.8	Runtime Eurosentiment parameters 1	123
B.9	Runtime Eurosentiment parameters 2	124
B.10	Eurosentiment results	124
B.11	New processing resource	124
B.12	Runtime parameters configuration for negative annotations	125

List of Tables

3.1	SEAS - Actors list	28
3.2	SAGA - Actors list	34
3.3	SEAS-Hadoop - Actors list	41
5.1	Execution enviroment	73
5.2	Execution times in GATE and Hadoop	101

List of Acronyms

API	Application Programming Interface
GATE	General Architecture for Text Engineering
GSI	Group on Intelligent Systems
HDFS	Hadoop distributed file system
HTTP	Hypertext Transfer Protocol
J2EE	Java 2 Platform, Enterprise Edition
JSON-LD	JSON for Linking Data
NIF	NLP Interchange Format
NLP	Natural Language Processing
OWL	Web Ontology Language
PR	Processing Resource
SAGA	Sentiment and Emotion Analysis integrated in GATE
SEAS	Sentiment and Emotion Analysis Services
RDF	Resource Description Framework
REST	Representational state transfer
UDF	User Defined Function
URI	Uniform resource identifier
UML	Unified Modeling Language
UPM	Technical University of Madrid
XML	Extensible Markup Language

Introduction

This chapter provides an introduction to the problem which will be approached in this project. It provides an overview of the benefits of sentiment and emotion analysis of information, as well as the use of Big Data infrastructures for these matters. Furthermore, a deeper description of the project and its context is also given.

1.1 Context

Sentiment and emotion analysis techniques have become very popular in the last few years [8] and they are starting to be used in more and more applications every day. These techniques allow us to develop more complex analysis tools in which the most important goal is not to discover what the information is about, but about the feelings expressed in it.

This growth of popularity has been possible because of the exponential growth of the available information to be analyzed, which is bigger every day. This phenomenon is called Big Data, which is the direct consequence of the desire of processing large amounts of data due to the evolution of information technologies and communications. It is a solution to data processing when these data are composed by very large sets, as traditional methods of analysis represent high costs, both in time and resources. Therefore, the problems we will try to solve are the capture, storage, search, analysis, sharing and viewing of these sources of information.

These large amounts of information, which will be used in this project, comes from the web and social networks. This, as well as the tendency of human beings to communicate with each other and the world, has resulted in a new source of rich information resources.

Besides, this new way of treating the data opens the door to a new world of services, such as the implementation of new business models, fight against organized crime, the study of the spread of diseases and so much more, thanks, for example, to the messages that are written every day on social networks like Twitter.

This master thesis is developed as a part of Eurosentiment¹ project, which is an European project that aims to develop a large shared data pool for language resources meant to be used by sentiment analysis systems, in order to bundle together scattered resources. The project specifies a schema for sentiment analysis and normalize the metrics used for sentiment strength.

In addition, this master thesis is also developed as a part of Financial Twitter Tracker² project, which aims to enrich content with information extracted from financial social media Twitter and detect financial demand for new content on certain topics.

¹<http://www.gsi.dit.upm.es/index.php/en/component/jresearch/?view=project&task=show&id=78>

²<http://www.gsi.dit.upm.es/index.php/en/component/jresearch/?view=project&task=show&id=80>

1.2 Master thesis description

This project will provide a sentiment and emotion analysis service which aims to process a big volume of data that is obtained from different web sources.

To do so, a web service based on J2EE [9] server technologies will be developed. This web service will provide different sentiment and emotion analyses using a REST API, so POST requests will be send to this service in order to perform the different semantic analyses that it will contain.

Also, we will focus in two different end users. In one hand, we will develop a client oriented to linguistic engineers that want to combine the sentiment and emotion analyses provided by the service with other analysis tools such as the plugins and processing resources provided by GATE. On the other hand, we will use these analysis services using a Big Data platform as Hadoop in order to have the data obtaining, processing and analysis in the same platform.

So, the main purpose of this master thesis is to develop a standardized sentiment and emotion analysis service that is available via a REST API and can be used by different kinds of end users or clients. In order to achieve that, the project is divided in different modules as depicted in figure 1.1.

Inside this project we distinguish the following modules:

SEAS is a set of Sentiment and Emotion Analysis Services according to NIF. The NLP Interchange Format is an RDF/OWL-based format that aims to achieve interoperability between Natural Language Processing tools, language resources and annotations. NIF consists of specifications, ontologies and software, which are combined under the version identifier "2.0", but are versioned individually. The results of the service provided by SEAS will have the JSON-LD format and will follow Marl and Onyx ontologies, which are used to describe sentiment and emotion resources. This service is used via POST requests, which should contain the text to be analyzed and the analysis algorithm as parameters, additionally the corresponding NIF parameters.

SAGA (Sentiment and Emotion Analysis integrated in GATE) is a set of processing and linguistic resources, written in Java, developed to run sentiment and emotion analysis over text using GATE platform. Because of the nature of GATE, the text format should be plain or XML. This plugin will use the results provided by SEAS to make sentiment and emotion classification of the analyzed text via annotations. Also,

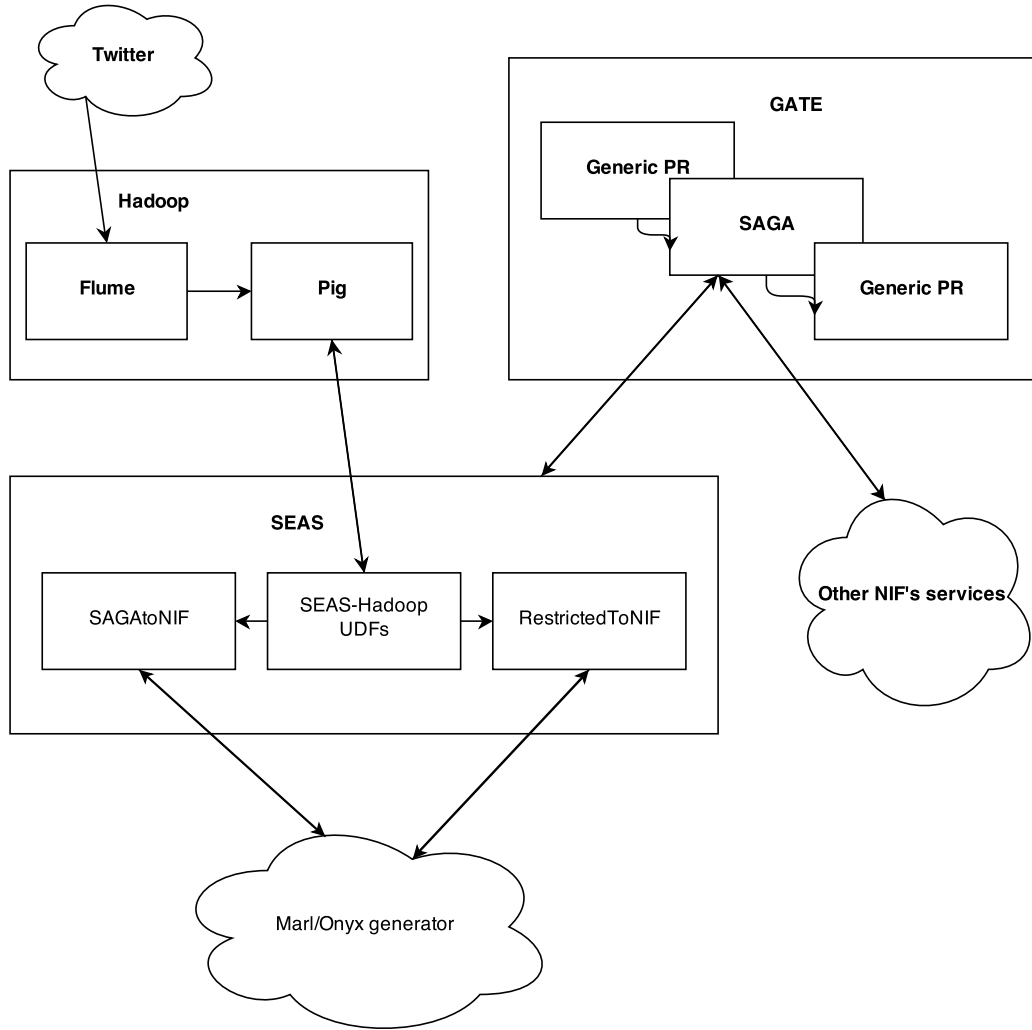


Figure 1.1: System general picture

the plugin allow us to configure different endpoints to another sentiment and emotion analysis services according to NIF, such as the provided by Eurosentiment.

SEAS-Hadoop will allow to integrate SEAS with the distributed processing platform called Hadoop. Over Hadoop we will execute Flume, a tool which retrieves tweets from Twitter about the topics we configure and stores them in HDFS (Hadoop distributed file system), and then a Pig script will be executed to process these data using a UDF (User defined function) that calls SEAS.

1.3 Master thesis goals

The main purpose of this project is to develop a **web service** that provides several **sentiment and emotion** analysis alternatives to process text. This service should be **interoperable** and **easy to use** from other platforms. To achieve that we will use **NIF** as a standard.

Then we should develop different tools to prove this interoperability and the advantages of use web distributed services. On one hand, we will create a plugin for a platform that process texts in a pipeline one by one. On the other hand, we will create a software that works over Hadoop, to create a distributed system that process data using the sentiment and emotion analyses provided by SEAS.

1.4 Structure of this Master Thesis

In this section we will provide a brief overview of all the chapters of this Master Thesis. It has been structured as follows:

Chapter 1 provides an introduction to the problem which will be approached in this project. It provides an overview of the benefits of mash-ups of semantic technologies. Furthermore, a deeper description of the project and its environment is also given.

Chapter 2 contains an overview of the existing technologies on which the development of the project will rely.

Chapter 3 describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language. The result of this evaluation will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

Chapter 4 describes the architecture of the system, dividing it into 3 groups and differentiating front-end and back-end modules.

Chapter 5 describes a selected use case. It is going to be explained the running of all the tools involved and its purpose. It is based on how to retrieve tweets from Twitter, stores them, analyzes the using SAGA o SEAS-Hadoop and present the results.

Chapter 6 sums up the findings and conclusions found throughout the document and gives a hint about future development to continue the work done for this master thesis.

Finally, the appendix provide useful related information, especially covering the installation and configuration of the tools used in this thesis.

Enabling Technologies

This chapter introduces which technologies have made possible this project. Because of its purpose, this project uses a lot of data processing oriented technologies. These technologies will cover five steps on the traditional path of data treatment, which are: capture, process, share, representation and storage of information and the resources associated with them, such as dictionaries, algorithms or configuration files.

2.1 Overview

As we approach the Web 3.0 [10], the availability of semantic technologies and related web services is growing every day. There are a lot of tools to discover the meaning of information, especially those which perform sentiment and emotion analysis. Also, there are a lot of web services to perform these kinds of tasks and a lot of processing technologies to treat big amounts of data.

However, developers face some difficulties when they want to make interoperable tools that can use different sentiment and emotion web analyzers over the same platform because there is no standard in the communication interface between them.

First, it is a difficult task to develop a generic call function that could be use to make a request to any web analysis service, because every service accepts its own parameters.

Second, every web service returns its result with different file formats and semantic structures, making even more difficult to develop a standardized processing tool.

Third, most of this analysis services are not ready to be used with a large amount of data or over distributed processing systems.

This master thesis describes the development of standardized sentiment and emotion analysis web service in terms of request and response handling, which is prepared to perform these analyses over any platform that can perform HTTP request.

The main goal of this project is to show the interoperability of this web service with any other processing tool. To do so, first we will develop a plugin for a non distributed corpus pipeline based processing tool like GATE [11] and then, a processing script that will work over a distributed processing system like Hadoop.

2.2 Opinion mining

Opinion mining [12] is a type of natural language processing which is oriented to discover the perception about the entities inside the information. In other words, what is thinking the person that have written this information about the entities he or she is talking about. This is a useful way to discover what people think about certain products, companies or events and it is pretty related with what we know as Big Data processing.

We can differentiate two kinds of opinion mining:

Sentiment analysis is a specific type of analysis inside opinion mining. It usually classifies data in three categories or polarities: positive, negative or neutral. These polarities are also specified with a numeric value that represents the intensity of the opinion and usually goes from -1 to 1.

Emotion analysis is a specific type of analysis inside opinion mining. It usually classifies data into one or more categories of emotions such as happiness, sadness, anger, fear... These categories are also specified with a numeric value that represents the intensity of the emotion and usually goes from -1 to 1.

2.3 Marl: An Ontology for Opinion Mining

Marl [1] is a standardised data schema (also referred as "ontology" or "vocabulary") designed to annotate and describe subjective opinions expressed on the web or in particular Information Systems. Marl is not a complete model to address the problem of describing and linking opinions online and inside information systems. It mainly defines concepts that are not described yet by the means of other ontologies and provides the data attributes that enable to connect opinions with contextual information already defined in metadata created with other ontologies.

The goals of the Marl ontology to achieve as a data schema are: first, enable to publish raw data about opinions and the sentiments expressed in them. Second, deliver schema that will allow to compare opinions coming from different systems (polarity, topics, features). Third, interconnect opinions by linking them to contextual information expressed with concepts from other popular ontologies or specialised domain ontologies

The Marl class diagram presented in Figure 2.1 shows connections between classes and properties used for describing opinions.

A very basic example of use depicted in Figure 2.2 shows a single opinion annotated with Marl metadata.

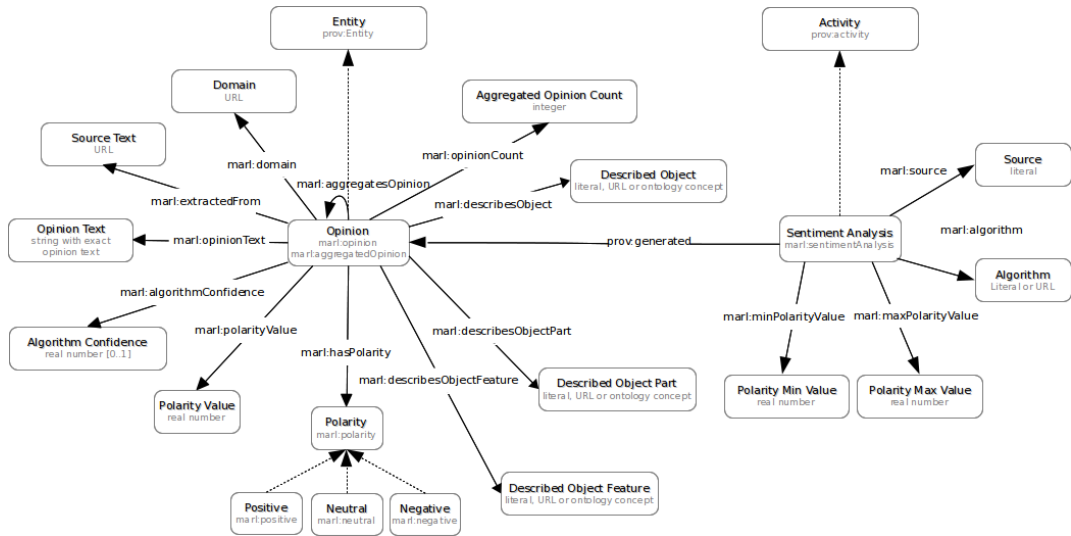


Figure 2.1: Class and Properties Diagram for the Marl Ontology [1]

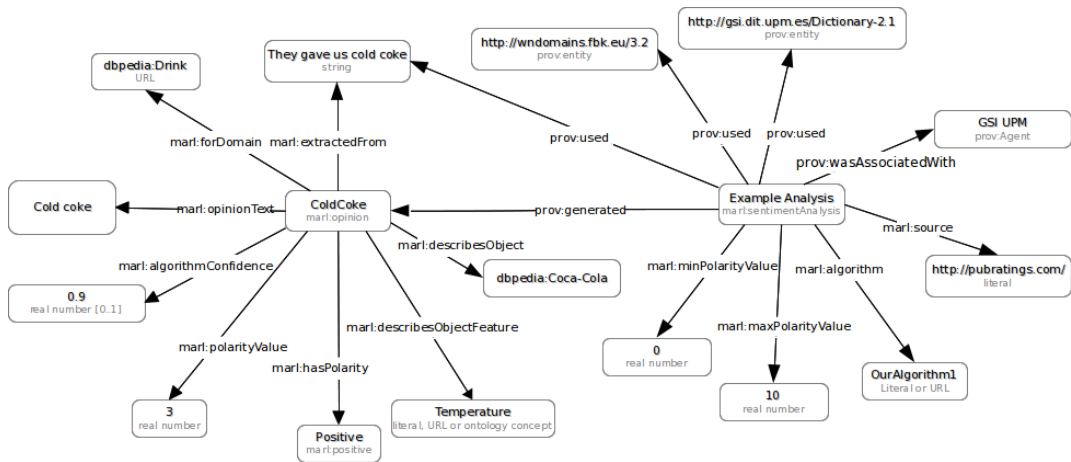


Figure 2.2: Marl, example of use [1]

2.4 Onyx: Describing Emotions on the Web of Data

Onyx [2] is a standardised data schema (also referred as "ontology" or "vocabulary") designed to annotate and describe the emotions expressed by user-generated content on the web or in particular Information Systems. Onyx aims to complement the Marl Ontology by providing a simple means to describe emotion analysis processes and results using semantic technologies.

The goals of the Onyx ontology to achieve as a data schema are: first, enable to publish raw data about emotions in user-generated content. Second, deliver schema that will allow to compare emotions coming from different systems (polarity, topics, features). Third,

interconnect emotions by linking them to contextual information expressed with concepts from other popular ontologies or specialised domain ontologies.

The Onyx class diagram presented in Figure 2.3 shows connections between classes and properties used for describing opinions.

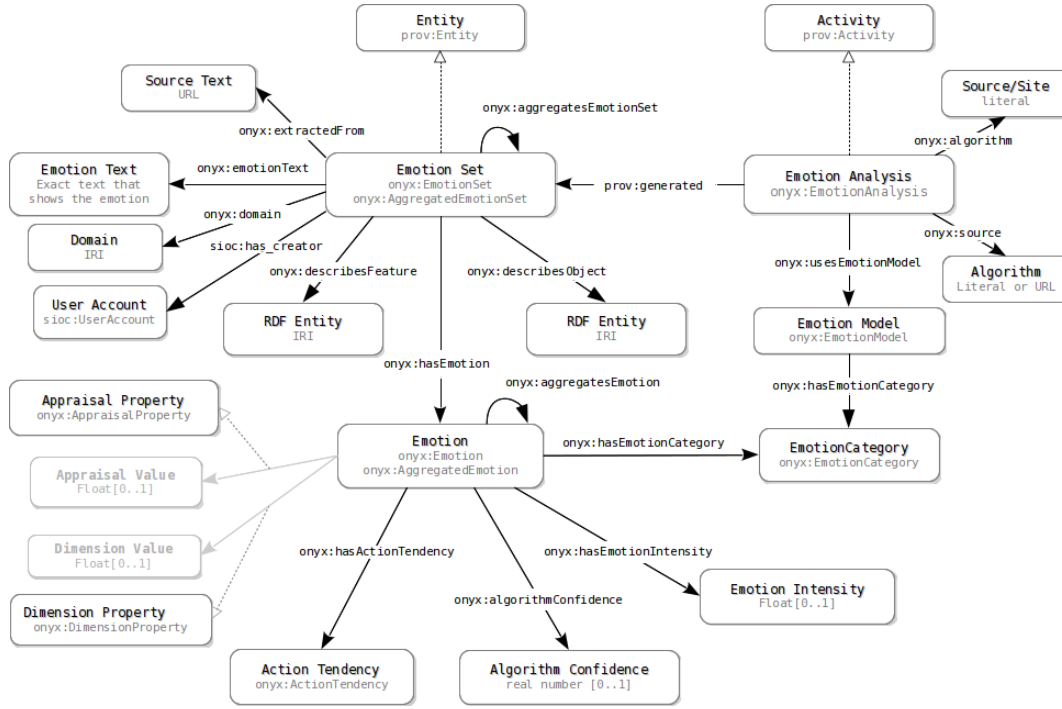


Figure 2.3: Class and Properties Diagram for the Onyx Ontology [2]

A very basic example of use depicted in Figure 2.3 shows a single opinion annotated with Onyx metadata.

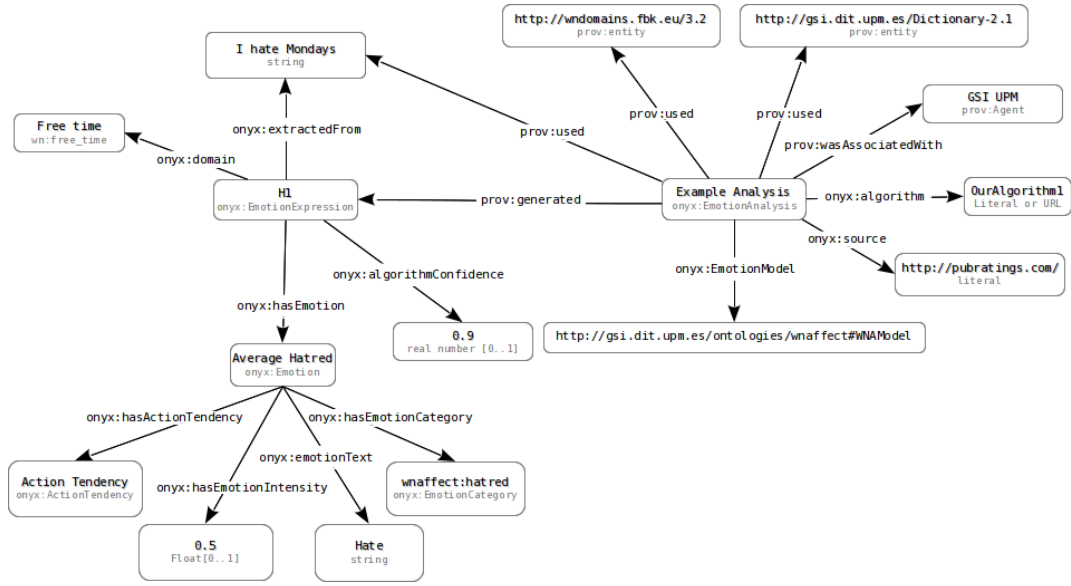


Figure 2.4: Onyx, example of use [2]

2.5 Eurosentiment

Eurosentiment (<http://eurosentiment.eu/section/project/>) is an European project that aims to develop a large shared data pool for language resources meant to be used by sentiment analysis systems, in order to bundle together scattered resources. The project specifies a schema for sentiment analysis and normalize the metrics used for sentiment strength. The sharing of resources is supported by a self-sustainable and profitable framework based on a community governance model, offering contributors the possibility of exploiting commercially the resources they provide. The project is structured around following steps:

- 1.- Definition of a common schema to ensure interoperability.
- 2.- Acquisition and clean up of language resources.
- 3.- Deployment of the resources.
- 4.- Validation through opinion mining demonstrators in the hotel and electronic domains.

The ontologies described in sections 2.3 and 2.4 have been developed under this project.

2.6 NIF 2.0

The NLP Interchange Format (NIF) [13] is an RDF/OWL-based format that aims to achieve interoperability between Natural Language Processing (NLP) tools, language resources and annotations. The development of this technology is motivated by the growing of the available processing tools and services that perform NLP tasks such as language detection, Named Entity Recognition, text classification, relationship extraction, sentiment and emotion analysis and so on.

NIF aims to turn the combination of NLP tools into a simple task by improving the compatibility of the results provided by these tools so they could be integrated in any service. With this, it is easier to build new applications as a result of the mash-up of different processing tools and services and the reuse of these applications.

NIF solves the interoperability problem by structuring itself into three layers:

Structural layer: NIF Core Ontology

The NIF Core Ontology provides classes and properties to describe the relations between substrings, text, documents and their URI schemes as we can see in Figure 2.5.

Conceptual layer

Processing tools and services should use the same vocabularies for the same kind of annotations, so the results are interoperable between services.

Access layer

Processing tools and services should have a standardized way to access them via the definition of parameters related to the processing tasks they perform. Some of them are the original file to be processed as an input, the format of the input file or the output format of the results.

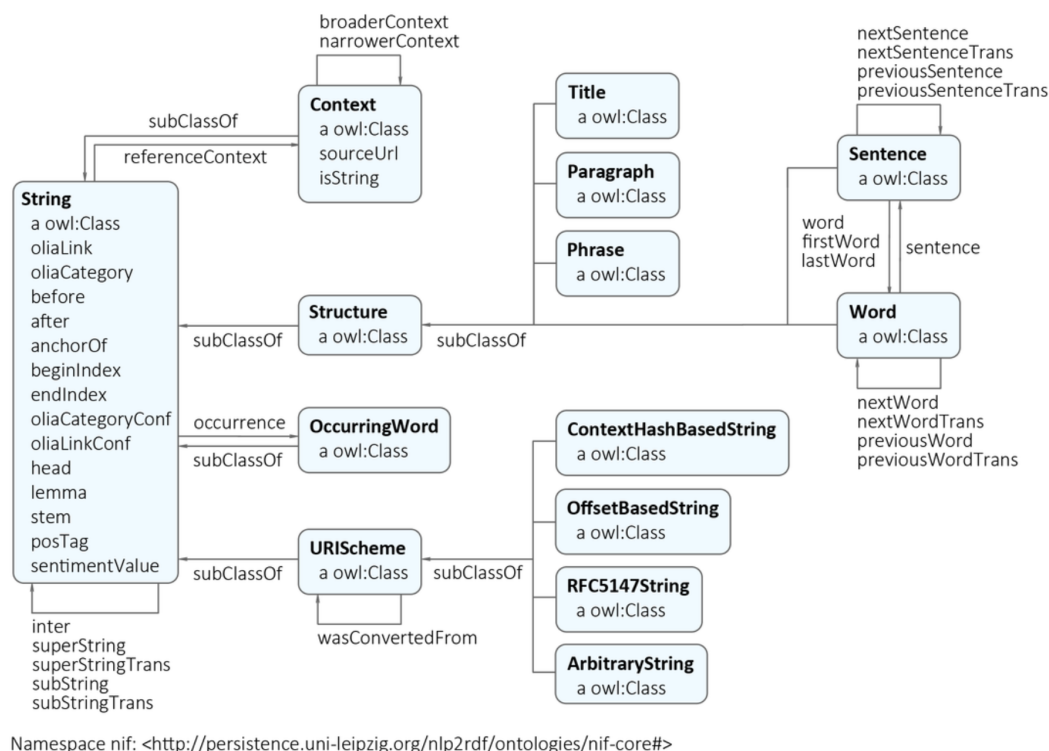


Figure 2.5: NIF Core Ontology [3]

2.7 GATE

GATE [11], general architecture for text engineering, is a platform developed by The University of Sheffield that brings a lot of capabilities to perform processing tasks over text. Although the family of GATE products available to perform this kind of tasks is quite big, we are only going to use and talk about two of them, the ones referenced in sections 2.7.1 and 2.7.2. These capabilities and the software used by them are developed in Java and some of them are available under the GNU Lesser General Public Licence 3.0.

2.7.1 GATE Developer

GATE Developer is a development environment that provides a graphical user interface, resources and capabilities that makes easier the development, configuration and running of text processing applications, as it is shown in Figure 2.6.

To understand the process of creation of an application inside GATE Developer, we should first clarify some important terms related to that matter:

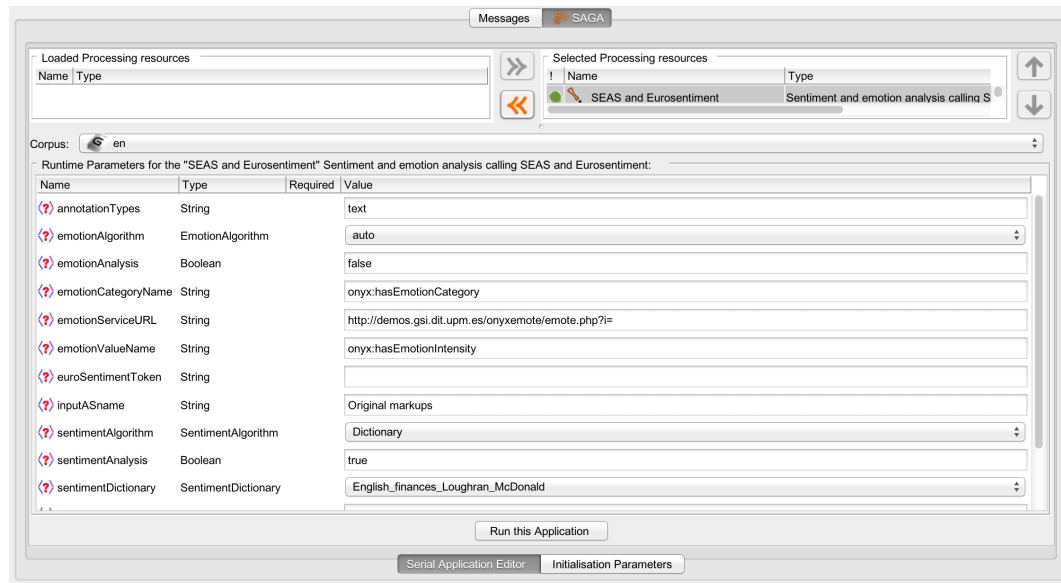


Figure 2.6: Example of configuration of a PR

A corpus is a set of documents which are related to each other. In other words, corpora contain documents about the same topics, for example: finances, electronics, food...

Processing resources (PR) are software components that perform specific processing tasks that manipulate and create annotations on documents.

Plugins are sets of processing resources. Some of them are directly provided by GATE and other ones are developed by third parties.

Pipelines or applications are comprising sequences of processing resources, that can be applied to a document or corpus.

2.7.2 GATE Embedded

GATE Embedded is an object-oriented framework implemented in Java. It is used in all GATE-based systems, and forms the core (non-visual) elements of GATE Developer and it allow us to develop new plugins for this tool. Embedded is split into a rich set of interlinked APIs and based on a standard Java component model. Some of the APIs available in Embedded are summarised in Figure 2.7.

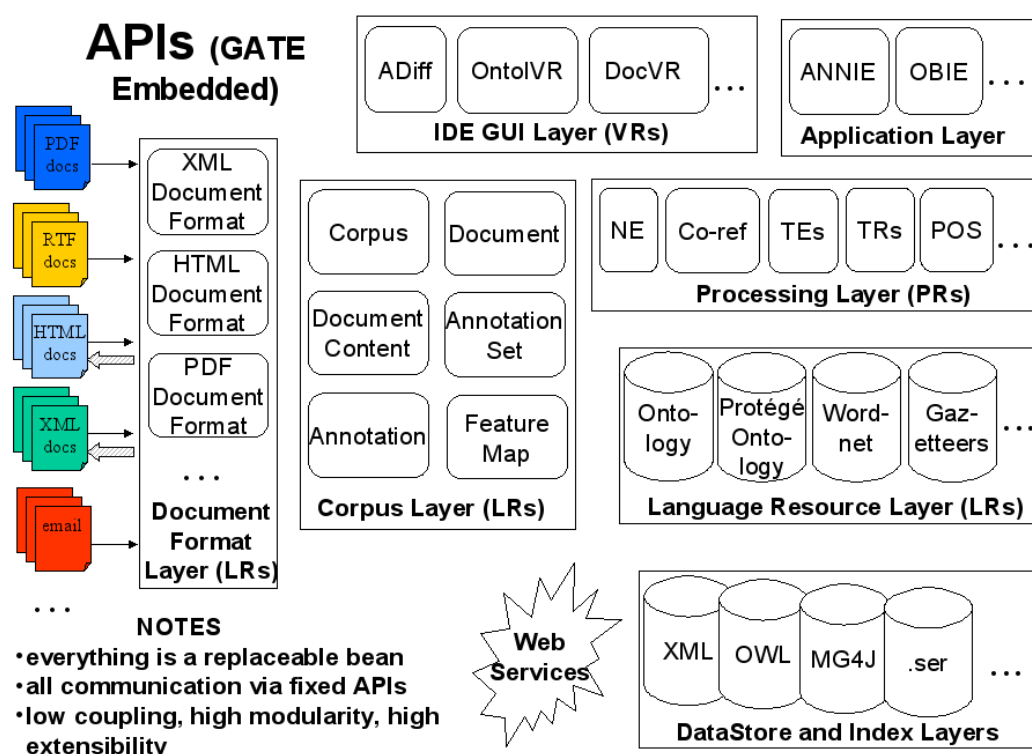


Figure 2.7: GATE Embedded APIs [4]

2.8 Hadoop

Hadoop[14] is a project developed by Apache which aims to provide a reliable platform for scalable distributed computing using simple programming models. It is a framework that allows the processing and storage of large data sets over a cluster of computers, also called nodes. Its main purpose is not to exploit the computing capabilities of the nodes in the cluster, but to exploit the interoperation between them, distribute the storage and processing task and handle failures to provide highly-available services.

These clusters can be configured as stand-alone, also known as single node cluster because there is only one computer in it, usually the ones from beginners user or test developers; or as a cluster with multiple nodes for big applications.

As we can see, Hadoop allows us to perform two different but highly related tasks: to store large data sets in a cluster or to process them. For this, there are two different modules inside Hadoop: HDFS and MapReduce.

2.8.1 HDFS: Hadoop Distributed File System

HDFS is a distributed file system used by Hadoop applications. Its main goal is to provide a highly reliable fault tolerant system that is transparent to the user or application, which perceives it as an only physical disk. This is achieved by Hadoop's architecture which is composed by different process or daemons that are being executed over the system and by the way storage units are configured.

2.8.1.1 HDFS daemons

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, the master, and a number of DataNodes, usually one per node in the cluster.

NameNode is a process executed only over one node of the cluster and its main purpose is to manage the file system metadata and namespace. It executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.

DataNodes manage storage and backup attached to the nodes that they run on. They are responsible for serving read and write requests from the file system's clients and also perform block creation, deletion, and replication upon instruction from the NameNode.

Secondary NameNode performs maintenance tasks that the namenode does not.

2.8.1.2 HDFS blocks

Files are stored in HDFS by dividing them into blocks. A block is the minimum storage unit in HDFS and usually it has a 64MB size. These blocks are managed by the NameNode and stored in the DataNodes. With this, HDFS achieves two things: to reduce the seek time and to make easier the data replication.

Finally, HDFS architecture looks as depicted in Figure 2.8.

2.8.2 Hadoop MapReduce

MapReduce is a paradigm for the development of big data processing services and tools over distributed systems. MapReduce technology is mainly divided into two parts, depending

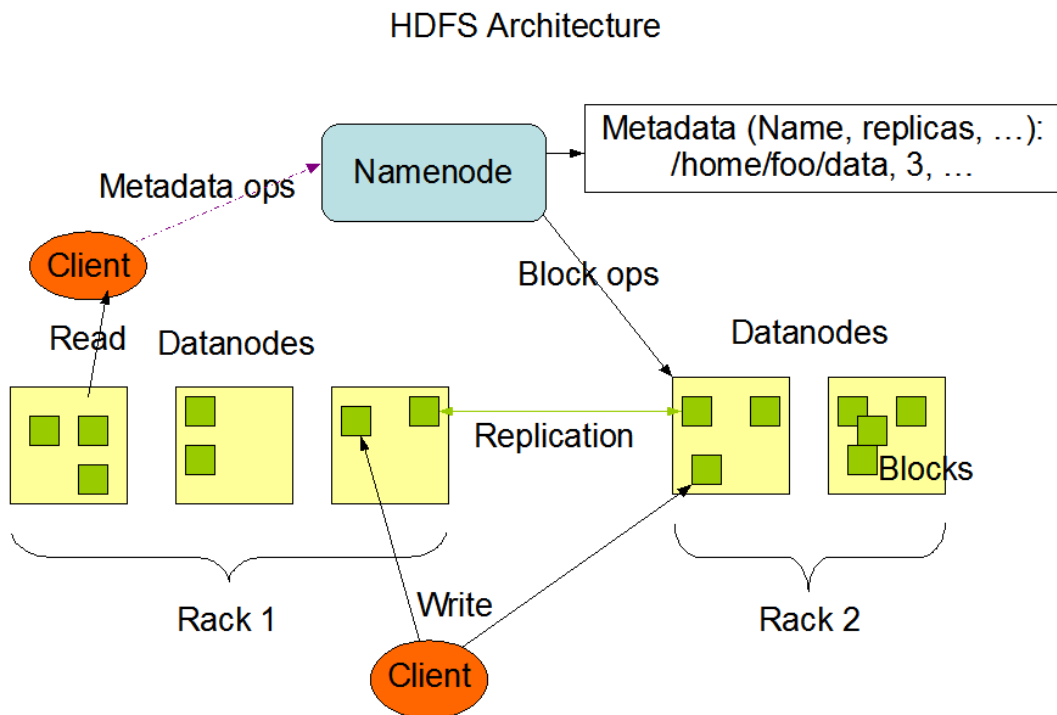


Figure 2.8: HDFS architecture [5]

on the tasks or jobs performed. First, there are map jobs and then, reduce jobs, which are executed in that precise order.

Map jobs take data sets and transforms them into another data sets, in order to obtain key-value tuples. In particular, they are responsible for distributing the workload between nodes to facilitate the processing tasks.

Reduce jobs take the outputs of the map jobs as inputs and combine them with each other to obtain the desired result for the processing task.

As we can see, the benefits of MapReduce are allowing developers to create parallel processing threads thanks to its high scalability, without having to worry about other things such as communication between nodes, monitoring tasks or fault tolerance. These tasks are automated at each node, which must periodically report its status and information regarding the work completed.

In Hadoop, the tasks described before are performed by the following processes:

JobTracker is going to distribute the map and reduce jobs between the different TaskTrackers, trying to bring these jobs the closest as possible to the DataNodes that contains the data to be processed. It also maintains updated information about the

status of the TaskTrackers.

TaskTrackers perform the map or reduce jobs assigned by the JobTracker.

Finally, Hadoop architecture for a MapReduce application looks as shown in Figure 2.9.

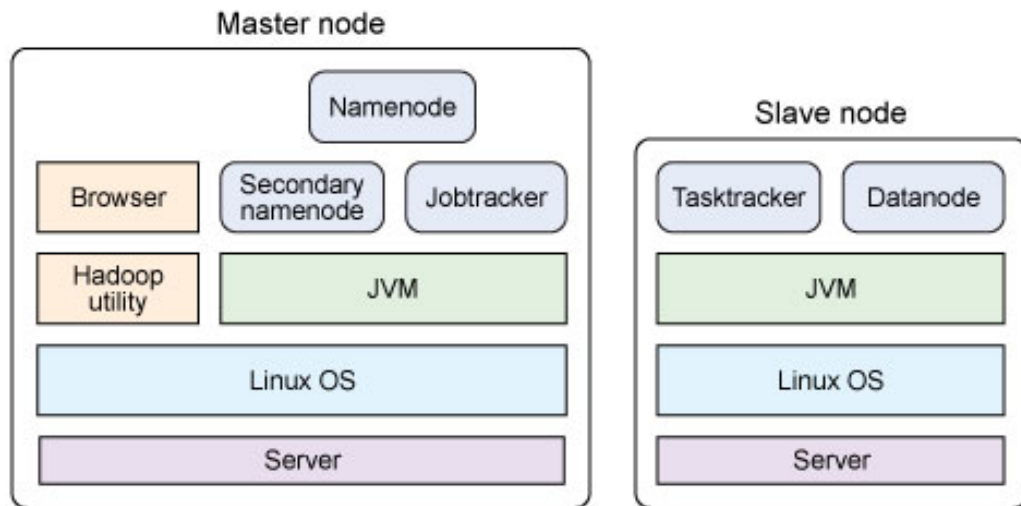


Figure 2.9: Hadoop master/slave architecture [6]

2.9 Flume

Flume [15] is a project (<http://flume.apache.org>) developed by Apache which aims to provide a distributed and reliable service to obtain large amounts of data from the web. Its architecture is based on streaming data flows.

Flume can receive events, which are little pieces of data, from different web sources and stores them in a channel until they are consumed by a sink. When this happens, these data are put in an external repository like HDFS. This is depicted in Figure 2.10.

In this case, we will configure Flume to fetch data from Twitter as its source and store them in HDFS as it is shown in the following example:

Listing 2.1: Example Flume-Twitter configuration

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.type = com.cloudera.flume.source.TwitterSource
```

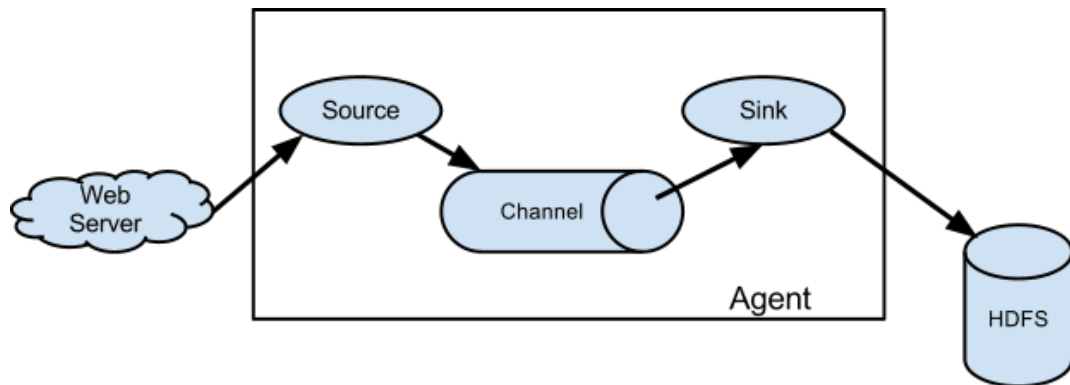


Figure 2.10: Flume architecture [7]

```

TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey = consumerKey
TwitterAgent.sources.Twitter.consumerSecret = consumerSecret
TwitterAgent.sources.Twitter.accessToken = accessToken
TwitterAgent.sources.Twitter.accessTokenSecret = accessTokenSecret

TwitterAgent.sources.Twitter.keywords = #ARGvsGER, ARGvsGER, #ArgentinavsGermany,
    ArgentinavsGermany, #ArgentinavsAlemania, ArgentinavsAlemania
TwitterAgent.sources.Twitter.keywords.created_at = Sun Jul 13

TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:54310/user/david/data/input/
    football/ArgentinaGermany
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100

```

2.10 Pig

Pig [16] is an engine that allows the execution of data processing scripts over Hadoop. These scripts, also called data flows, are written in a data processing oriented language called Pig Latin. Pig aims to use Hadoop capabilities such as HDFS and MapReduce to achieve reliable, fast and distributed parallel data processing.

One of the big advantages of Pig is that its configuration in terms of distributed data pro-

cessing relies in the configuration of Hadoop clusters, so once you have configured Hadoop, you don't have to think about setting up Pig.

Pig Latin focuses in data flow instead of control flow as other traditional programming languages would do. In other words, Pig Latin focuses on loading, processing and storing the data. Even though Pig Latin offers the user a lot of relational operations and functions, users are allowed to develop their own functions, called UDFs.

2.10.1 UDFs

User Defined Functions can be developed in Java or Python. They allow the user to create its own data processing functions and export their results into the data flow as a new parameter of the relations.

2.10.2 Elephant Bird

Elephant Bird (<https://github.com/kevinweil/elephant-bird/>) is Twitter's open source library of LZ4, Thrift, and/or Protocol Buffer-related Hadoop InputFormats, OutputFormats, Writables, Pig LoadFuncs, Hive SerDe, HBase miscellanea, etc. The majority of these are in production at Twitter running over data every day.

In this project, we will be using Pig UDFs to load tweets in JSON format which were obtained using Flume.

2.11 Conclusions

In this chapter we have introduce some of the technologies which are part of this project and conform the base for this master thesis.

As we can see, each one of them covers a step on the traditional data processing flow: capture, storage, search, sharing, analysis and visualization.

Because of the purpose of this project, this chapter has introduced a lot of data processing oriented technologies, that are going to be combined to build the goal of this master thesis: a sentiment and emotion analysis service over a Big Data infrastructure.

Requirement Analysis

This chapter describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language.

3.1 Overview

The result of this chapter will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

3.2 Use cases

These sections identify the use cases of each module. This helps us to obtain a complete specification of the uses of the system, and therefore define the complete list of requisites to match. We will present a list of the actors and a UML diagram representing all the actors participating in the different use cases for each module. This representation allows to specify the actors that interact in the system and the relationships between them.

These use cases will be described the next sections, including each one a table with their complete specification. Using these tables, we will be able to define the requirements to be established.

3.2.1 SEAS

3.2.1.1 Actors dictionary

The list of primary and secondary actors is presented in table 3.1. These actors participate in the different use cases, which are presented later.

Actor identifier	Role	Description
ACT-1	User	End user that uses SEAS to perform sentiment or emotion analysis over a text
ACT-2	Developer	Technical developer that wants to add new NIF services to SEAS
ACT-3	GATE user	GATE user that calls SEAS to perform sentiment or emotion annotations over a corpus of documents
ACT-4	Eurosentiment Marl and Onyx generator	External services that performs conversions from SEAS's generic results into Marl and Onyx ontologies
ACT-5	Onyxemote	External services for emotion analysis expressed in onyx ontologies.

Table 3.1: SEAS - Actors list

3.2.1.2 Use cases

This use case package collects the analysis functionalities of SEAS, as shown in 3.1.

The use cases presented in this section are as shown in the Figure 3.1:

- *sentiment analysis* detailed in sub-section 3.2.1.3.
- *emotion analysis* detailed in sub-section 3.2.1.4.
- *new services* detailed in sub-section 3.2.1.5.
- *test and demo* detailed in sub-section 3.2.1.6.

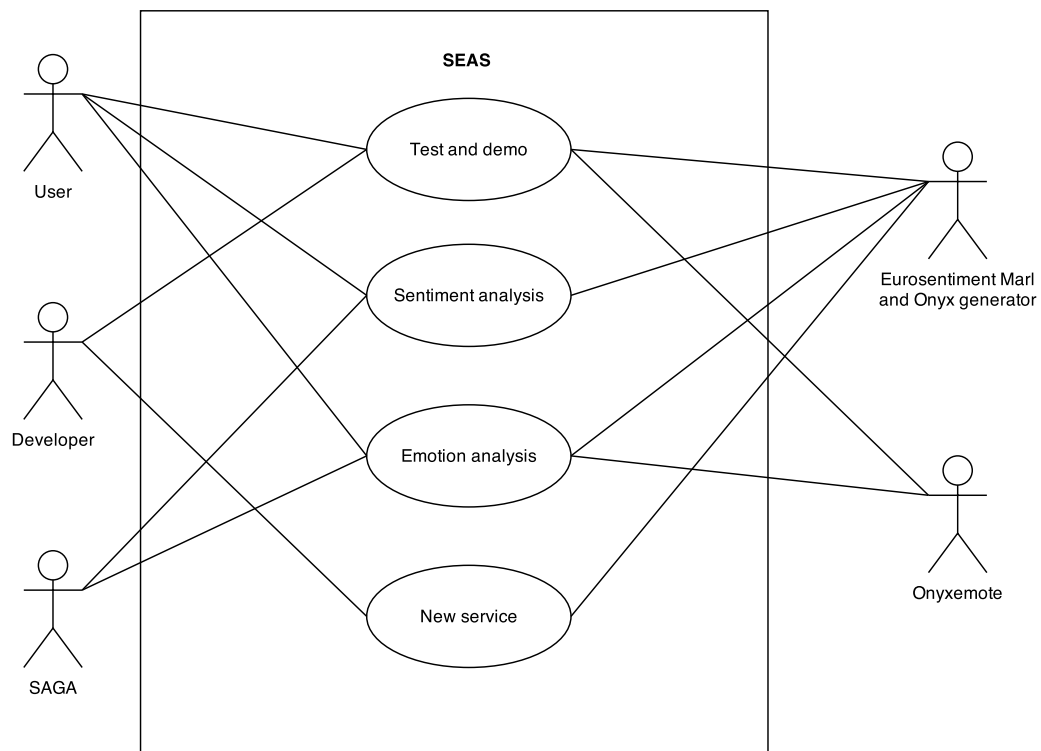


Figure 3.1: SEAS use case

3.2.1.3 Sentiment analysis

Use Case Name	sentiment analysis		
Use Case ID	UC1.1		
Primary Actor	User		
Secondary Actor	Eurosentiment Marl Generator		
Pre-Condition	Tomcat server has been initialized, database access and Eurosentiment Marl Generator are available		
Post-Condition	Tomcat server keeps running and database access is available		
Flow of Events		Actor Input	System Response
	1a	The user sends a POST request to perform the analysis with the wrong NIF parameters.	SEAS sends a HTTP response indicating the wrong NIF parameter that needs to be corrected.
	1b	The user sends a POST request to perform the analysis with correct, but not supported by the service, NIF parameters.	SEAS sends a HTTP response indicating the correct, but not supported, NIF parameter that needs to be a valid one.
	1c	The user sends a POST request to perform the analysis with the correct and supported NIF parameters, including the text to be analyzed	SEAS checks what sentiment analysis algorithm has been chosen and performed the analysis. Then, it sends the results to the external service called Eurosentiment Marl Generator, which puts them into JSON-LD format according to Marl ontology. Then, this JSON is returned to the user.

3.2.1.4 Emotion analysis

Use Case Name	Emotion analysis		
Use Case ID	UC1.2		
Primary Actor	User		
Secondary Actor	EuroSentiment Onyx Generator, Onyxemote		
Pre-Condition	Tomcat server has been initialized, database access, EuroSentiment Onyx Generator and Onyxemote are available		
Post-Condition	Tomcat server keeps running and database access is available		
Flow of Events		Actor Input	System Response
	1a	The user sends a POST request to perform the analysis with the wrong NIF parameters.	SEAS sends a HTTP response indicating the wrong NIF parameter that needs to be corrected.
	1c	The user sends a POST request to perform the analysis with the correct and supported NIF parameters, including the text to be analyzed, indicating the use of Onyxemote service to the analysis.	SEAS sends the text to be analyzed to the external service called Onyxemote, which performs emotion analysis over that text and puts the result into JSON-LD format according to Onyx ontology. Then, this JSON is returned to the user.
	1d	The user sends a POST request to perform the analysis with the correct and supported NIF parameters	SEAS checks the chosen emotion analysis algorithm, performs it and sends the results to the external service call Eurosentiment Onyx Generator.

3.2.1.5 New services

Use Case Name	new services		
Use Case ID	UC1.3		
Primary Actor	Developer		
Secondary Actor	Eurosentiment Marl or Onyx generator		
Pre-Condition	The developer has installed Tomcat into its system and has a suitable JEE platform like Eclipse		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1a	The developer implements a proper NIF wrapper of a current sentiment or emotion analysis service so it can be used inside SEAS	-
	1b	The developer implements a new sentiment or emotion analysis algorithm inside SEAS so it is according to NIF	-

3.2.1.6 Test and demo

Use Case Name	text and demo		
Use Case ID	UC1.4		
Primary Actor	User, Developer		
Secondary Actor	Eurosentient Marl or Onyx generator, Onyxemote		
Pre-Condition	The user/ developer has an internet connection		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The User / Developer test the service by going to http://demos.gsi.dit.upm.es/tomcat/SEAS/Controller	SEAS responds to the petitions in the same web so the user/developer can see the kind of responses the system gives.

3.2.2 SAGA

3.2.2.1 Actors dictionary

The list of primary and secondary actors is presented in table 3.2. These actors participate in the different use cases, which are presented later.

Actor identifier	Role	Description
ACT-1	User	End user that uses SAGA to perform sentiment or emotion annotations over a corpus of documents
ACT-2	Developer	Technical developer that wants to add new Processing Resources to SAGA or to update the existing ones.
ACT-3	SEAS	External service for sentiment and emotion analysis called by SAGA
ACT-4	Eurosentiment	External services for sentiment and emotion analysis called by SAGA

Table 3.2: SAGA - Actors list

3.2.2.2 Use cases

This use case package collects the SAGA use cases, as shown in 3.2

- *sentiment and emotion annotations calling SEAS* detailed in subsection 3.2.2.3
- *opinion annotation* detailed in subsection 3.2.2.4
- *eurosentiment and other endpoints* detailed in subsection 3.2.2.5

- *update SAGA* detailed in subsection 3.2.2.6
- *new processing resource* detailed in subsection 3.2.2.7

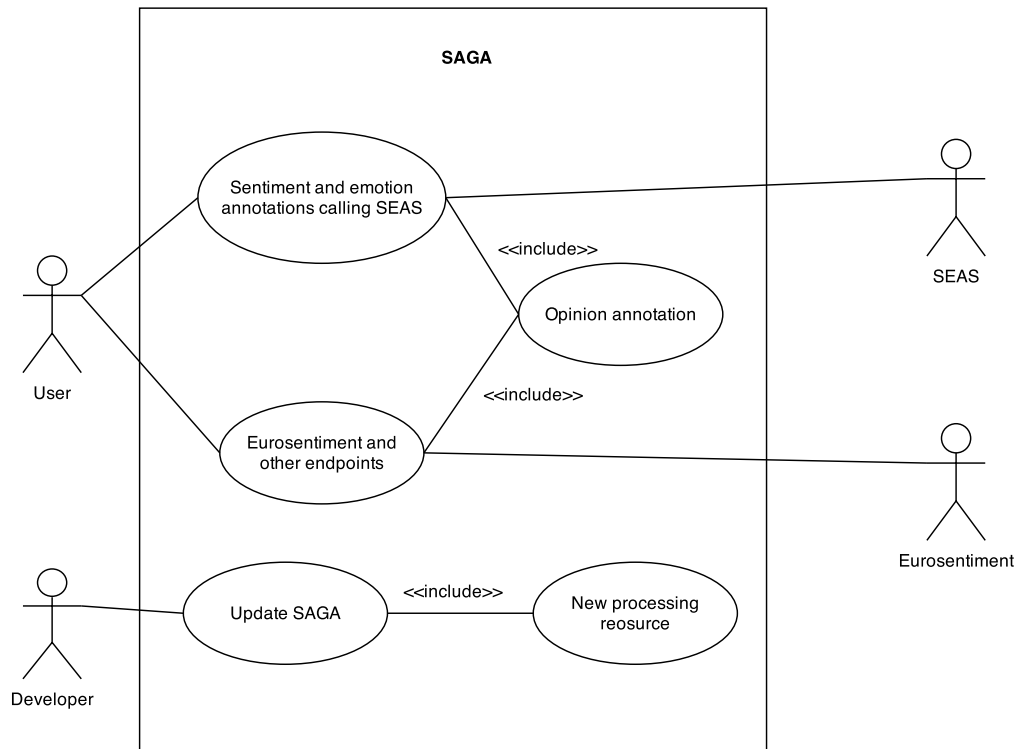


Figure 3.2: SAGA use case

3.2.2.3 Sentiment and emotion annotations calling SEAS

Use Case Name	sentiment and emotion annotations calling SEAS		
Use Case ID	UC2.1		
Primary Actor	User		
Secondary Actor	SEAS		
Pre-Condition	GATE Developer. Availability of SEAS, local or as a web service		
Post-Condition	Annotations are added to the document		
Flow of Events		Actor Input	System Response
	1	The user creates the corpus of documents to be analyzed and populates it	The system creates the corpus
	2	The user adds the analysis PR to the pipeline inside GATE and configures it to call SEAS services as a sentiment or emotion analysis endpoint. Then, the user sets the pipeline to run over the created corpus and runs it.	When executed inside the pipeline, SAGA's PR will call SEAS to perform the analysis and will receive the result in JSON format.
	3a		If the JSON is correct, the PR will use it to generate the corresponding sentiment annotations over the document.
	3b		If the JSON is incorrect, no annotations will be created on the document.

3.2.2.4 Opinion annotations

Use Case Name	opinion annotations		
Use Case ID	UC2.2		
Primary Actor	User		
Pre-Condition	GATE Developer		
Post-Condition	Annotations are created		
Flow of Events		Actor Input	System Response
	1	The user creates a corpus and populates it with documents with the same sentiment polarity or emotion category.	The system creates the corpus.
	2	The user configures the annotation PR to make the same annotation over all the documents in the corpus according to its sentiment polarity or emotion category.	The PR annotates the entire corpus.

3.2.2.5 Sentiment and emotion annotations calling other NIF services

Use Case Name	sentiment and emotion annotations calling other NIF services		
Use Case ID	UC2.3		
Primary Actor	User		
Secondary Actor	Eurosentiment, other NIF services.		
Pre-Condition	Availability of external services. Results according to Marl and Onyx		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The user creates the corpus of documents to be analyzed and populates it	The system creates the corpus
	2	The user adds the analysis PR to the pipeline inside GATE, sets the API Key and configures it to call Eurosentimen services as a sentiment or emotion analysis endpoint. Then, the user sets the pipeline to run over the created corpus and runs it.	When executed inside the pipeline, SAGA's PR will call Eurosentiment services to perform the analysis and will receive the result in JSON format.
	3a		If the JSON is correct, the PR will use it to generate the corresponding sentiment annotations over the document.
	3b		If the JSON is incorrect, no annotations will be created on the document.

3.2.2.6 Update SAGA

Use Case Name	update SAGA		
Use Case ID	UC2.4		
Primary Actor	Developer		
Pre-Condition	GATE Developer and Embedded		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The developer add new sentiment or emotion analysis algorithms to the list of available services to configure in the PR.	The new algorithms are listed in the Runtime Parameters of the PR.

3.2.2.7 New PR

Use Case Name	new PR		
Use Case ID	UC2.5		
Primary Actor	Developer		
Pre-Condition	GATE Developer and Embedded		
Post-Condition	The new PR is registered in GATE		
Flow of Events		Actor Input	System Response
	1	The developer creates a new PR related to sentiment or emotion analysis that returns its results as annotations.	
	2	The developer registers the new PR in the creole.xml file and loads SAGA again.	
	3a		The new PR is added to the available PRs.
	3b		The registration process fails and and the new PR is not added.

3.2.3 SEAS-Hadoop

3.2.3.1 Actors dictionary

The list of primary and secondary actors is presented in table 3.3. These actors participate in the different use cases, which are presented later.

Actor identifier	Role	Description
ACT-1	User	End user that uses SEAS-Hadoop
ACT-2	Developer	Technical developer that wants to add new scripts to SEAS-Hadoop or to update the existing ones
ACT-3	SEAS	External service for sentiment and emotion analysis called by scripts inside SEAS-Hadoop
ACT-3	Web	External service for getting data

Table 3.3: SEAS-Hadoop - Actors list

3.2.3.2 Use cases

This use case package collects the SEAS-Hadoop use cases, as shown in 3.3.

- *get data* detailed in 3.2.3.3
- *sentiment and emotion analysis over data* detailed in 3.2.3.4
- *new script* detailed in 3.2.3.5

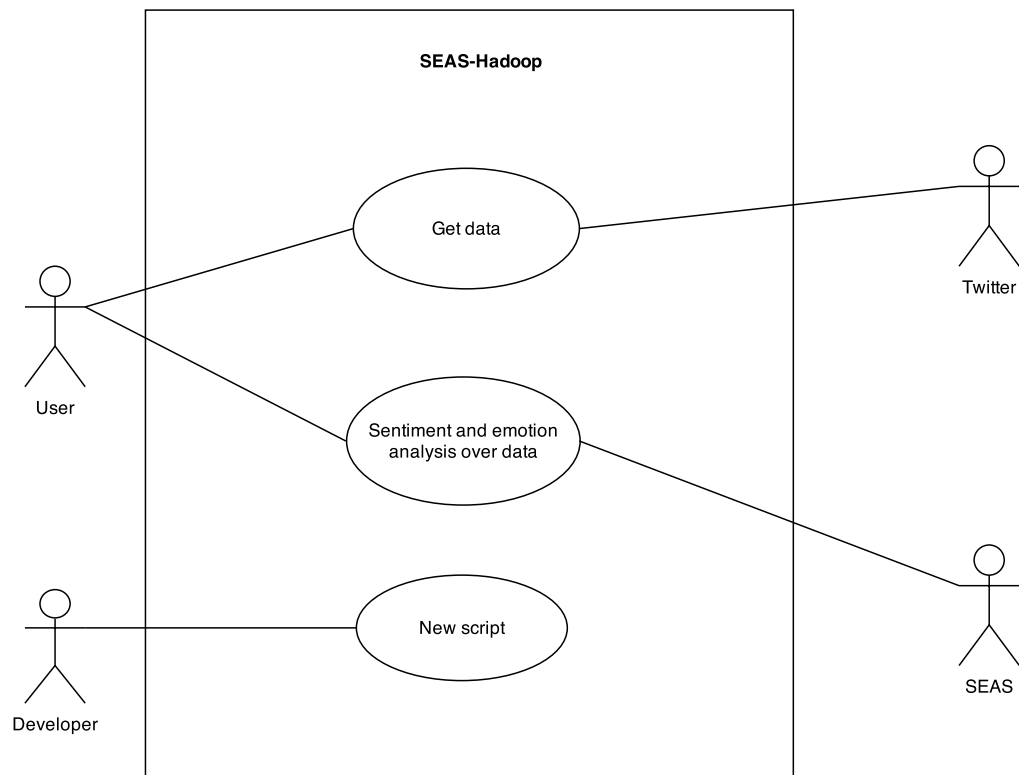


Figure 3.3: SEAS-Hadoop use case

3.2.3.3 Get data

Use Case Name	get data		
Use Case ID	UC3.1		
Primary Actor	User		
Secondary Actor	Web		
Pre-Condition	Hadoop daemons are running.		
Post-Condition	Data are stored in HDFS.		
Flow of Events		Actor Input	System Response
	1	The user configures Flume to fetch data from the Web by setting the corresponding API keys, tokens and keywords to look for.	
	2	The user runs Flume.	Data starts being stored in HDFS with JSON format.
	2	The user stops Flume.	

3.2.3.4 Sentiment and emotion analysis over data

Use Case Name	sentiment and emotion analysis over data		
Use Case ID	UC3.2		
Primary Actor	User		
Secondary Actor	SEAS		
Pre-Condition	Hadoop daemons are running. There are data available in HDFS. SEAS is available.		
Post-Condition	Results are stored in HDFS.		
Flow of Events		Actor Input	System Response
	1	The user opens the Pig Latin script, sets the path to the data stored in HDFS and chooses between sentiment or emotion analysis. Then, run it.	The system calls SEAS to perform the analysis over the content of each data and return its sentiment polarity or emotion category.
	2		Data are classified by their sentiment polarity or emotion category and stored in HDFS.

3.2.3.5 New script

Use Case Name	new script		
Use Case ID	UC3.2		
Primary Actor	Developer		
Pre-Condition	Pig Latin knowledge.		
Flow of Events		Actor Input	System Response
	1	The user can create a lot of new scripts due to the information that each piece of data contains: geolocation, language, hashtags, mentions, number of followers...	

3.2.4 Conclusions

With the use cases described we have introduced the basic functionalities that have been implemented in this project. They help us to understand the different actors that can interact. They can serve as a base for further development and different use cases that can come to mind.

CHAPTER 4

Architecture

This chapter describes in depth how the system is structured in different modules and how the users interact with them and also how the modules interact with other modules by themselves.

4.1 Introduction

The main purpose of this master thesis is to have a **sentiment and emotion analysis service** that is available via a **REST API**, can be used by different kinds of end users and it is standardized by using **NIF** (see section 2.6). First we need to set up a server that allows our service to operate, so it can accept requests from clients to perform the sentiment and emotion analysis algorithms that will be available.

The service will provide a standardized way to request for sentiment and emotion analysis. And it will let the user choose between different sentiment or emotion analysis services in different **languages**, such as English and Spanish. Once the chosen analysis is performed, the result of the analysis will be always return in **JSON-LD** format as a way to provide an **standardized response** for all the sentiment and emotion analysis services that are wrapped inside **SEAS**.

Then, different kinds of users can make use of this service just by using its REST API. For example, the direct use of this service can be performed by using the **demo**¹ that is available online at GSI's web.

Another way to use the service is to use a text processing software such as **GATE** (see section 2.7) so others text processing tools can be used with SEAS. GATE provides different plugins that allows the users to perform different kinds of text analysis such as tokenization, NER, POS tagging, language identification... A plugin called **SAGA** has been developed to perform sentiment and emotion analysis inside GATE by using the capabilities that SEAS offers us.

Finally, we will see that SEAS can be used over a **Big Data** infrastructure as **Hadoop** (See section 2.8). By using a **Pig Latin** (see section 2.10) script we will process data stored in HDFS and using a mapreduce context.

A diagram of the architecture is shown in Figure 4.1. Each module is detailed in the following sections.

¹<http://demos.gsi.dit.upm.es/tomcat/SEAS/Controller>

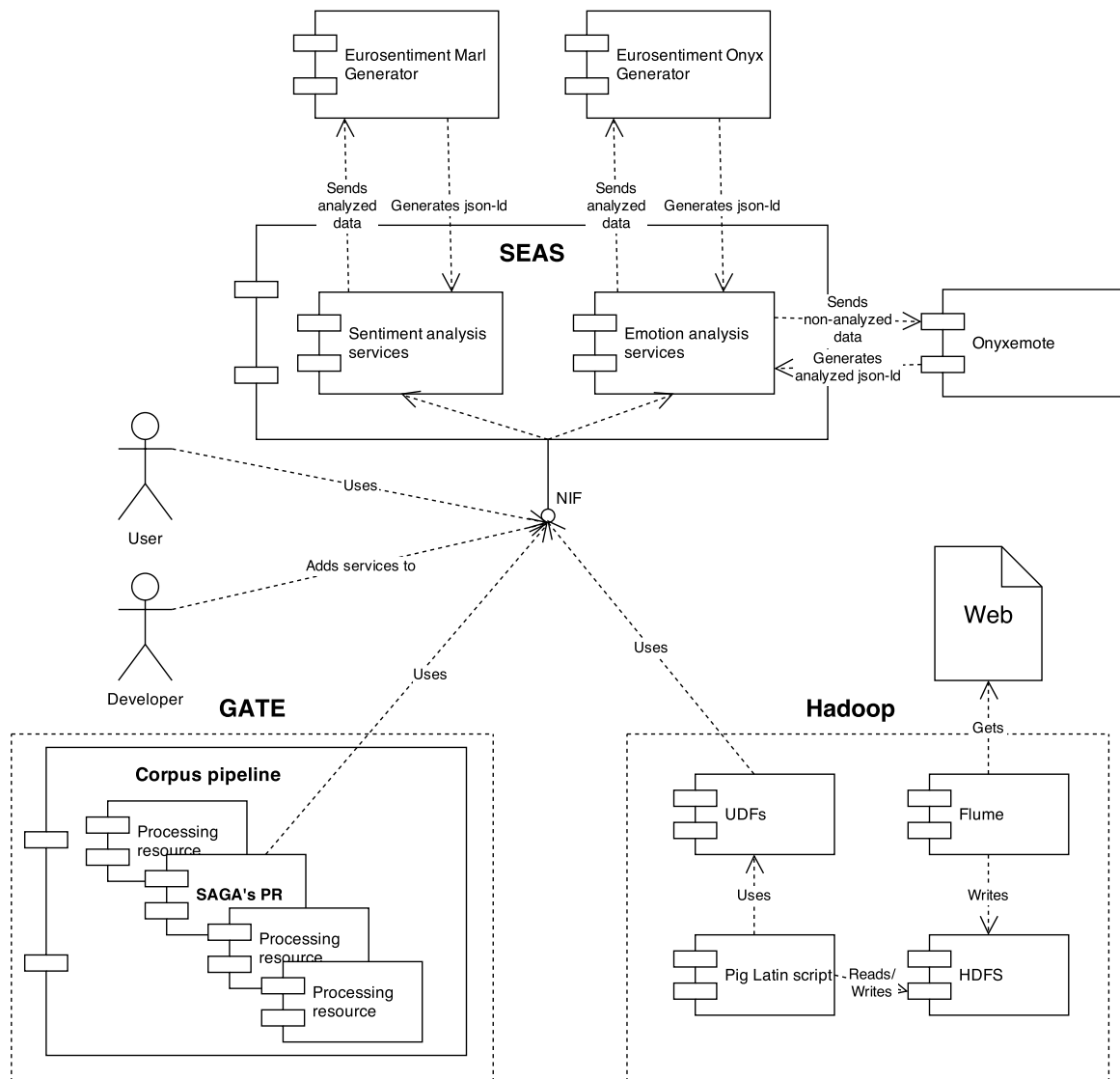


Figure 4.1: General Architecture

4.2 SEAS: Sentiment and emotion analysis services

The purpose of this service is to provide a set of sentiment and emotion analysis services for text processing. These services are offered according to NIF, so their access, input and output formats are standardized so SEAS can be used with others text or data processing tools in an interoperable way.

SEAS has a three tiered client/server architecture:

The client would be the different users of the platform. They could be a user of the demo

available at GSI, a GATE user that has added a SAGA's processing resource to its pipeline or a Hadoop user that is calling SEAS by an UDF. The client will make the HTTP requests to the service.

The server or the application logic will perform the different sentiment and emotion analysis services that are available inside SEAS.

The database will contain dictionaries that are needed for some of the analysis services.

SEAS's application logic is deployed in an Apache Tomcat 7 [17] server, so, as we know, it is a service developed in Java, and SEAS's database is deployed in a MongoDB [?].

SEAS is composed by different sentiment and emotion analysis services, some of them have been developed by me, and others, like Onyxemote, have been added to SEAS as a NIF wrapper.

4.2.1 Input format

As we have said, "these services are offered according to NIF", which means that the way of accessing the resources provided by SEAS are standardized as it is defined by NIF's access layer, as defined in section 2.6. With this standardization we achieve one of the main goals of this master thesis, which is to provide a interoperable service so it can be easily combined with others text processing services and tools.

To access the API you have to send a POST request to `http://localhost:8080/SAGAtoNIF/Service` or to `http://localhost:8080/RestrictedToNIF/RestrictedService` (if SEAS is deployed in a local server) or to `http://demos.gsi.dit.upm.es/tomcat/SAGAtoNIF/Service` or to `http://demos.gsi.dit.upm.es/tomcat/RestrictedToNIF/RestrictedService` (if you want to use our current deployment at GSI) with these parameters:

input is the text that is going to be analyzed and it should be a plain text.

informat is the format of the input, which value should be *text*.

intype value should be *direct*, which means that the text is provided as plain text inside the request.

outformat value indicates the output format, which should be *JSON-LD*.

algo value is used to indicate the sentiment or emotion analysis algorithm to be used. The value can be: *spFinancial*, *emoticon*, *spFinancialEmoticon*, *enFinancial*, *enFinancialEmoticon*, *ANEW2010All*, *ANEW2010Men*, *ANEW2010Women*, *onyx*.

The parameters named *input*, *informat*, *intype* and *outformat* are used to make the HTTP request according to NIF format.

4.2.2 Sentiment analysis

All sentiment analysis services are executed inside SEAS's application logic. The sentiment analysis will be performed in the same way for each sentiment algorithm and the intermediate results will have the same format.

The available sentiment analysis services are:

Spanish finances dictionaries provided by Paradigma.

English finances dictionaries provided by Loughran and McDonald, which are not available for commercial use without authorization.

Emoticon dictionaries, public available.

A combination between the previous dictionaries.

For a given text, any service will return the following intermediate result:

The text that has been analyzed by the sentiment analysis service.

The polarity of the analyzed text, that can be *positive*, *negative* or *neutral*

The value of the polarity, that can go from *-1* (negative) to *1* (positive).

Each relevant word inside the text whit its polarity (*positive* or *negative*) and its value (*-1* or *1*)

These intermediate results are sent to Eurosentiment Marl Generator (<http://demos.gsi.dit.upm.es/eurosentiment/generator/api>), which is another service according to NIF developed by GSI that will receive these data, will process it using a template and will generate the final result or output in JSON format according to Marl ontology (see section 2.3).

All the process is depicted in Figure 4.2.

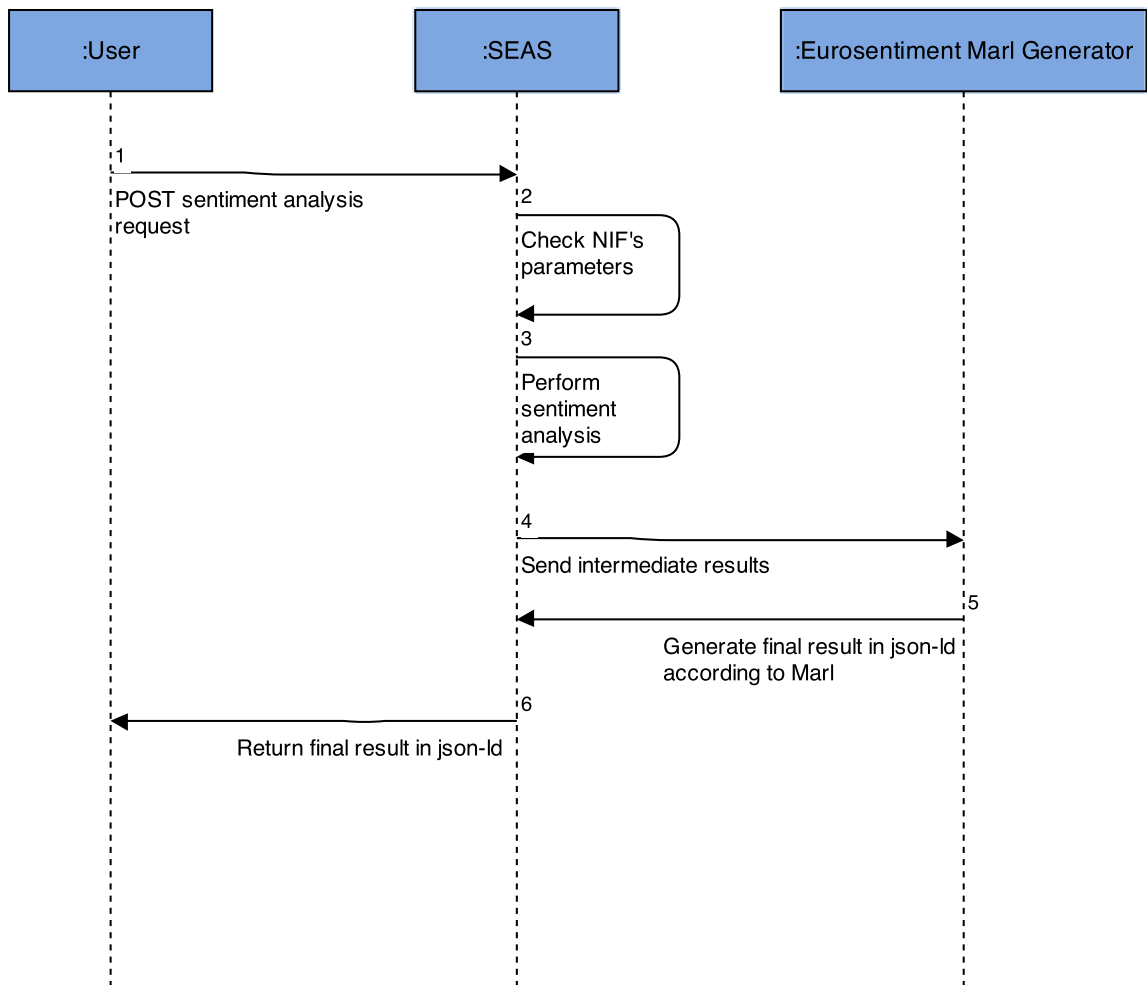


Figure 4.2: Sequence diagram for sentiment analysis

4.2.2.1 Output format

As it happens with the input format, the outputs of the service are also standardized. The final output that comes as a result of a sentiment analysis will be provided in JSON-LD format. This JSON will contain all the information related with the text analyzed, its polarity and its value expressed using Marl Ontology.

For an example input as the following:

Listing 4.1: "Example of sentiment analysis request/input"

```
curl --data "input=I feel good :)&intype=direct&
```

```
informat=text&outformat=json-ld&algo=
enFinancialEmoticon" http://localhost:8080/
RestrictedToNIF/RestrictedService
```

The following output will be returned:

Listing 4.2: "Example of sentiment analysis output"

```
{
"@context": "http://demos.gsi.dit.upm.es/eurosentiment/static
/context.jsonld",
"analysis": [
{
"@id": "http://www.gsi.dit.upm.es/ontologies/analysis#
SAGA",
"@type": [
"marl:SentimentAnalysis"
],
"marl:maxPolarityValue": 1.0,
"marl:minPolarityValue": -1.0
}
],
"entries": [
{
"nif:isString": "I feel good :)",
"opinions": [
{
"@id": "_:Opinion1",
"marl:hasPolarity": "marl:Positive",
"marl:polarityValue": 1.0,
"marl:describesObjectFeature": "Overall"
}],
"strings": [
{
"nif:anchorOf": "good",
"nif:beginIndex": 7,
"nif:endIndex": 10,
```

```
      "opinions": {
        "@id": "_:Opinion",
        "marl:hasPolarity": "marl:Positive",
        "marl:polarityValue": 1.0
      }
    } ,
    {
      "nif:anchorOf": " :) ",
      "nif:beginIndex": 12,
      "nif:endIndex": 13,
      "opinions": {
        "@id": "_:Opinion",
        "marl:hasPolarity": "marl:Positive",
        "marl:polarityValue": 1.0
      }
    }
  ]
}
]
```

4.2.3 Emotion analysis

All emotion analysis services, except Onyxemote, are executed inside SEAS. The emotion analysis will be performed in the same way for each emotion algorithm and the intermediate results will have the same format.

The available emotion analysis services are:

Affective Norms for English Words dictionaries provided by University of Florida², which are not available for commercial use without authorization.

Onyxemote³, a service developed by GSI to perform emotion analysis and express the results in Onyx ontology.

For a given text, any service will return the following intermediate result:

²<http://csea.php.ufl.edu/media/anewmessage.html>

³<http://demos.gsi.dit.upm.es/onyxemote/>

The text that has been analyzed by the emotion analysis service.

The category or categories of the analyzed text, that can be *happiness*, *sadness*, *anger*...

The value of each category, that can go from *-1* to *1*.

Each relevant word inside the text whit its category and its value (*-1* or *1*)

These intermediate results are sent to Eurosentiment Onyx Generator, which is another service according to NIF that will receive these data, will process it using a template and will generate the final result or output in JSON format according to Onyx ontology (see section 2.4).

All the process is depicted in Figure 4.3.

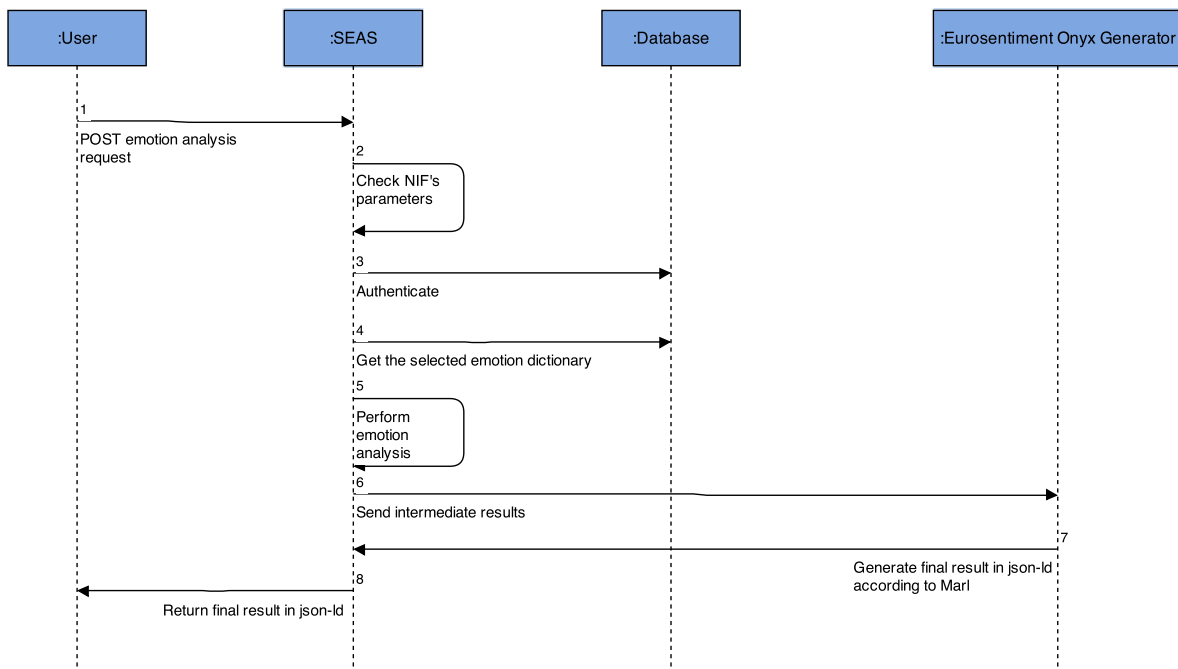


Figure 4.3: Sequence diagram for emotion analysis

4.2.3.1 Output format

As it happens with the input format, the outputs of the service are also standardized. The final output that comes as a result of a emotion analysis will be provided in JSON-LD format. This JSON will contain all the information related with the text analyzed, its categories and its values expressed using Onyx Ontology.

For an example input as the following:

Listing 4.3: "Example of sentiment analysis request/input"

```
curl --data "input=I feel good :)&intype=direct&
informat=text&outformat=json-ld&algo=ANEW2010All"
http://localhost:8080/RestrictedToNIF/
RestrictedService
```

The following output will be returned:

Listing 4.4: "Creating information view"

```
{
"@context":{
  "dc":"http://purl.org/dc/terms/",
  "dc:subject":{
    "@type":"@id"
  },
  "xsd":"http://www.w3.org/2001/XMLSchema#",
  "marl":"http://www.gsi.dit.upm.es/ontologies/marl/ns#",
  "nif":"http://persistence.uni-leipzig.org/nlp2rdf/
    ontologies/nif-core#",
  "onyx":"http://www.gsi.dit.upm.es/ontologies/onyx/ns#",
  "emotions":{
    "@id":"onyx:hasEmotionSet",
    "@type":"onyx:EmotionSet"
  },
  "opinions":{
    "@container":"@list",
    "@id":"marl:hasOpinion",
    "@type":"marl:Opinion"
  },
  "prov":"http://www.w3.org/ns/prov#",
  "rdfs":"http://www.w3.org/2000/01/rdf-schema#",
  "analysis":{
    "@id":"prov:wasInformedBy"
  },
}
```

```
"entries":{
  "@id":"prov:generated"
},
"strings":{
  "@reverse":"nif:hasContext",
  "@type":"nif:String"
},
"date":{
  "@id":"dc:date",
  "@type":"xsd:dateTime"
},
"wnaffect":"http://www.gsi.dit.upm.es/ontologies/wnaffect#"
},
"@id":"http://demos.gsi.dit.upm.es/onyxemote/#Analysis",
"@type":"prov:Activity",
"entries":{
  "@id":"http://demos.gsi.dit.upm.es/onyxemote/emote.php?i=I+
    feel+good+:)&o=jsonld#char=0,14",
  "@type":"nif:Context",
  "nif:isString":"I feel good :)",
  "onyx:hasEmotionSet":{
    "@id":"_:b0",
    "@type":"onyx:EmotionSet",
    "onyx:hasEmotion":[
      {
        "@id":"_:b1",
        "@type":"onyx:Emotion",
        "onyx:hasEmotionCategory":{
          "@id":"http://www.gsi.dit.upm.es/ontologies/
            wnaffect/ns#happiness"
        },
        "onyx:hasEmotionIntensity":{
          "@type":"xsd:decimal",
          "@value":"0.75"
        }
      }
    ]
  },
  {

```

```

        "@id": "_:b2",
        "@type": "onyx:Emotion",
        "onyx:hasEmotionCategory": {
            "@id": "http://www.gsi.dit.upm.es/ontologies/
                wnaffect/ns#anger"
        },
        "onyx:hasEmotionIntensity": {
            "@type": "xsd:decimal",
            "@value": "0.039705882352941"
        }
    }
]
}
},
"analysis": {
    "@id": "http://demos.gsi.dit.upm.es/onyxemote/#activity",
    "@type": "onyx:EmotionAnalysis",
    "onyx:algorithm": "Synesketch",
    "onyx:usesEmotionModel": {
        "@id": "http://www.gsi.dit.upm.es/ontologies/wnaffect/ns#
            OnyxModel"
    },
    "prov:wasAssociatedWith": {
        "@id": "http://www.gsi.dit.upm.es",
        "@type": "prov:Agent"
    }
}
}
}

```

4.3 SAGA: Sentiment and Emotion Analysis integrated in GATE

SAGA is a set of processing and linguistic resources, written in Java, developed to run sentiment and emotion analysis over text using GATE platform. SAGA is distributed as a GATE plugin.

As we have said in section 2.7, GATE is a platform developed by The University of

Sheffield that brings a lot of capabilities to perform processing tasks over text. It offers a lot of different plugins and processing resources, some of them developed by them and others developed by third-parties, to perform text analysis over big sets of documents.

These text analyses can be language identification, POS tagging, sentence splitter, tokenization... SAGA is a plugin that contains a set of processing resources that are developed to perform sentiment and emotion analysis over text or xml documents.

The main purpose of SAGA is to provide a SEAS's client that can be used from GATE, so it can perform sentiment and emotion analysis by calling SEAS or other similar NIF's services such as the available at Eurosentiment Portal⁴. With this, SAGA gives interoperability to GATE pipelines, because SAGA also respects NIF's API and make sentiment and emotion annotations according to Marl and Onyx ontologies.

4.3.1 PR: Processing resources

As a reminder of what it was said in section 2.7, a processing resource is a software component that perform specific processing tasks that manipulate and create annotations on documents. In this case, SAGA's processing resources will create sentiment and/or emotion annotations over the analyzed documents. These annotations will be according Marl and Onyx ontologies so they have semantic value.

4.3.1.1 Predefined Sentiment Annotation PR

This is a processing resource that will create predefined sentiment annotations over a corpus of documents. In other words, this processing resource won't call any analysis services, it will create the sentiment annotations defined by the user in its Runtime Parameters.

Because of its nature, this processing resource will be useful to annotate already classified corpus to GATE format, or what is the same, it will annotate all the documents in a corpus with the same corresponding sentiment polarity. So, if a corpus is composed by positive documents, it will annotate all the documents as positive; and if a corpus is composed by negative documents, it will annotate all the documents as negative.

To do so, the processing resource will have the following runtime parameters:

Annotation Type is the name of the annotation in which the sentiment polarity will be added.

⁴<http://portal.eurosentiment.eu>

Input Annotation Set Name is the name of the annotation set containing the Annotation Type.

Sentiment Polarity Name is the name of the annotation key.

Sentiment Polarity is the value of the annotation value.

For example, to annotate a corpus of negative documents with negative annotations, the parameter will be configured as depicted in Figure 4.4.





Name	Type	Required	Value
 annotationType	String		paragraph
 inputASName	String		Transferred
 sentimentPolarity	String		marl:Negative
 sentimentPolarityName	String		marl:hasPolarity

Figure 4.4: Runtime parameters configuration for negative annotations

As result, all the document inside this negative corpus will get a negative annotation inside *paragraph* with the value *marl:hasPolarity=marl:Negative*

All the process is depicted in Figure 4.5.

4.3.1.2 Sentiment And Emotion Analysis Calling SEAS PR

This is a processing resource that will create sentiment and/or emotion annotations over a corpus of documents by calling sentiment or emotion analysis services such as the ones provided by SEAS.

Even Though this PR was first thought to be used only with SEAS's services, because its use of NIF and its standardization, it is compatible with other NIF's sentiment and emotion analysis services such as the provided in the Eurosentiment Portal. This compatibility is possible because of the highly configurable Runtime Parameters of this processing resource, that allows the user to configure which web services want to use to perform the analysis.

To do so, the processing resource will have the following runtime parameters:

inputASName is the Annotation Set that contains the annotation type to be analyzed.

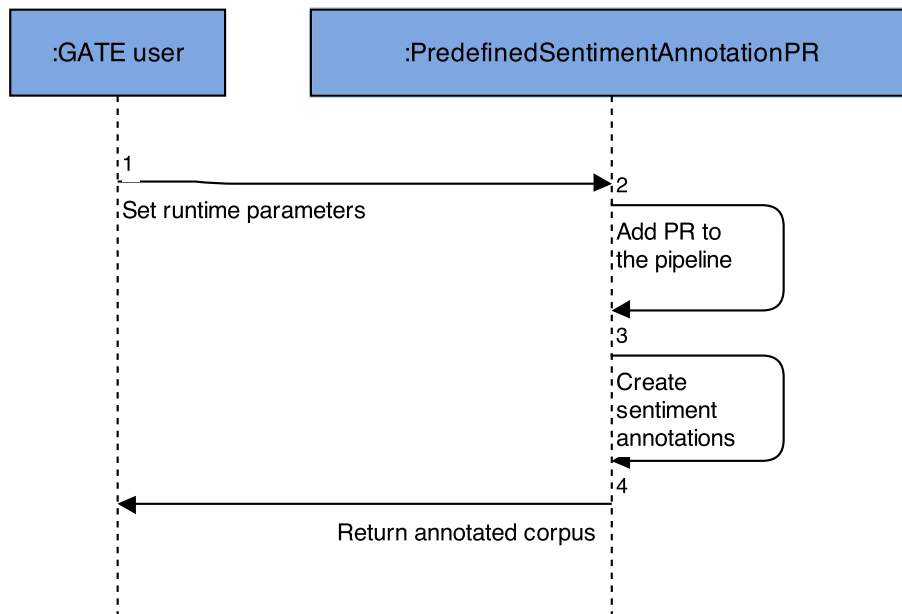


Figure 4.5: Sequence diagram for Predefined Sentiment Annotation PR

annotationType is the annotation type to be analyzed.

sentimentAnalysis is a runtime parameter that sets if the PR is going to perform sentiment analysis with the chosen url or algorithm.

emotionAnalysis is a runtime parameter that sets if the PR is going to perform emotion analysis with the chosen url or algorithm.

SentimentServiceURL is the endpoint of the sentiment analysis service. If you deploy SEAS as a local service in your computer (Recommended): <http://localhost:8080/SAGAtoNIF/Service> and <http://localhost:8080/RestrictedToNIF/RestrictedService>. You can use the demo available at GSI's website: <http://demos.gsi.dit.upm.es/tomcat/SAGAtoNIF/Service> and <http://demos.gsi.dit.upm.es/tomcat/RestrictedToNIF/RestrictedService>. For more endpoints visit the Eurosentiment portal - <https://portal.eurosentiment.eu>

EmotionServiceURL is the endpoint of the emotion analysis service.

APIKey is the Eurosentiment token to use their services or other similar services that require an API KEY.

ApiKeyName is Eurosentiment (or other similar services) token name to use their services.

sentimentAlgorithm is the runtime parameter that sets the sentiment algorithm that the service is going to use. At the moment, you can use dictionary based algorithms.

sentimentDictionary is the runtime parameter that sets the sentiment dictionary that the service is going to use (in case that **sentimentAlgorithm** has been chosen). You can use the values *AUTO (Detects language)*, *Spanish finances Paradigma*, *English finances Loughran McDonald*, *Emoticon*, *Spanish finances and Emoticon*, *English finances and Emoticon*.

emotionAlgorithm is a runtime parameter that sets the emotion algorithm that the service is going to use. You can use *AUTO (Detects language)*, *onyx*, *ANEW2010All*, *ANEW2010Men*, *ANEW2010Women*.

SentimentPolarityName is the name of the sentiment polarity feature.

SentimentValueName is the name of the sentiment value feature.

EmotionCategoryName is the name of the emotion category feature.

EmotionValueName is the name of the emotion value feature.

Once the parameters have been configured, this processing resource will make a POST request for each document to the sentiment or emotion analysis service that has been set. If the set service is SEAS, it will add to the request the value of the selected service.

SEAS will receive the request from this PR and will do what has been described in section 4.2. SEAS will return a JSON with the result of the analyzed document.

Finally, the processing resource will parse this JSON and will extract the relevant information related with the sentiment or emotion analysis:

If it is a sentiment analysis , the PR will create an annotation for the sentiment polarity, which can be *marl:hasPolarity=marl:Negative*, *marl:hasPolarity=marl:Neutral* or *marl:hasPolarity=marl:Positive*; and then it will create other annotation for the polarity value with the format *marl:polarityValue=number*.

If it is an emotion analysis , the PR will create an annotation for the emotion category with the format *onyx:hasCategory=URIemotion* and then it will create other annotation for the polarity value with the format *onyx:categoryValue=number*.

For example, to perform a financial sentiment analysis over an English corpus, the parameters will be configured as depicted in Figure 4.6 and 4.7.

APIKey	String	
annotationTypes	String	paragraph
apiKeyName	String	x-eurosentiment-token
emotionAlgorithm	EmotionAlgorithm	auto
emotionAnalysis	Boolean	false
emotionCategoryName	String	onyx:hasEmotionCategory
emotionServiceURL	String	
emotionValueName	String	onyx:hasEmotionIntensity
inputASName	String	Original markups
sentimentAlgorithm	SentimentAlgorithm	Dictionary

Figure 4.6: Runtime parameters configuration for sentiment annotations

sentimentAnalysis	Boolean	true
sentimentDictionary	SentimentDictionary	English_finances_and_Emoticon
sentimentPolarityName	String	marl:hasPolarity
sentimentServiceURL	String	http://localhost:8080/RestrictedToNIF/RestrictedService
sentimentValueName	String	marl:polarityValue

Figure 4.7: Runtime parameters configuration for sentiment annotations

As result, the document inside the corpus will get sentiment annotation inside *paragraph* with the value *marl:hasPolarity=polarity*

All the process is depicted in Figure 4.8.

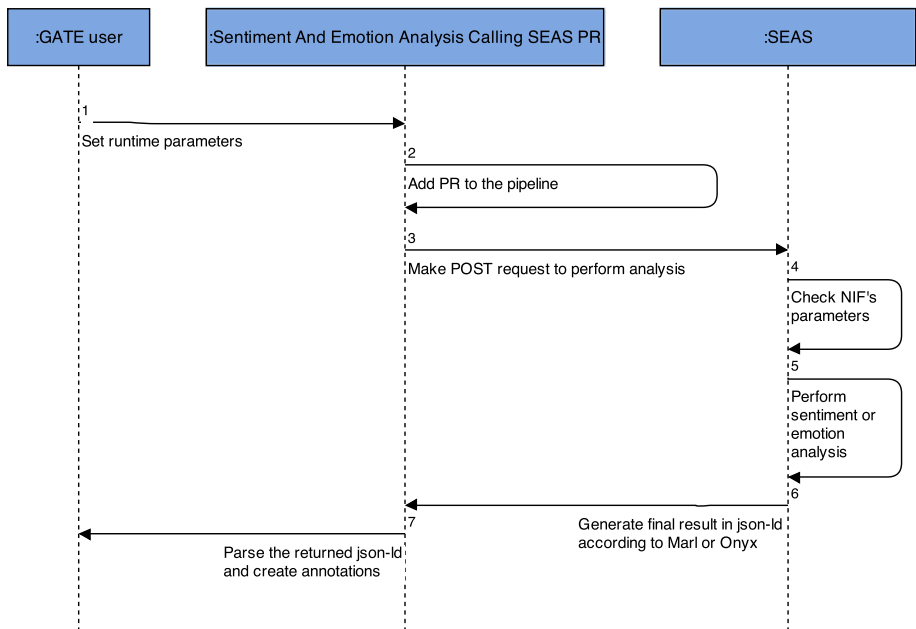


Figure 4.8: Sequence diagram for Sentimen And Emotient Analysis Calling SEAS PR

4.4 SEAS-Hadoop: Sentiment and emotion analysis over a Big Data infrastructure

The main goal of this module is to provide a Big Data infrastructure that achieves two things: the first one, to automatically obtain data from web sources and store them into our system, and the second one, to process these data using, among other tools, the sentiment and emotion analysis services provided by SEAS.

To do so, Hadoop will be used as the Big Data infrastructure that will serve as a platform for our purposes. As it was indicated in section 2.8, Hadoop is a project developed by Apache which aims to provide a reliable platform for scalable distributed computing using simple programming models. With Hadoop we get two important things: a distributed file system to store our data called HDFS (see section 2.8.1) and a distributed processing system called MapReduce (see section 2.8.2).

The distribution of data storage and processing is achieved by Hadoop's architecture itself, which consists in a cluster of computers, so the storage and processing tasks are distributed between the different nodes.

In our case, our architecture for this module consists only in one node or computer, also known as *stand-alone* or *single node configuration*. With this, we get all the Hadoop's capabilities in only one computer, so this node will be at the same time the *NameNode*, *DataNode*, *Secondary NameNode*, *JobTracker* and *TaskTracker*, as it was explained in section 2.8.1.1 and 2.8.2.

With this we get all the necessary to have a reliable Big Data infrastructure oriented to data processing, but how do we achieve our two goals: get data from the web and process it?

4.4.1 Flume

Flume is, as it was explained in 2.9, a project developed by Apache that allows the users to obtain data from different web sources. It works over Hadoop, so the users don't have to care about the distributed storing or processing of data.

First of all, Flume needs to be configured to know from which web service should obtain data and in which way. Flume allows to configure different parameters, such as the endpoint service, keywords to look for in the data or the maximum and minimum size of the obtained files.

When it is executed, the different Flume processes are distributed through the different nodes of the user's Hadoop configuration and start fetching data from the configured web service. Once the data are obtained, Flume stores them in HDFS in a raw format.

This process is depicted in Figure 4.9.

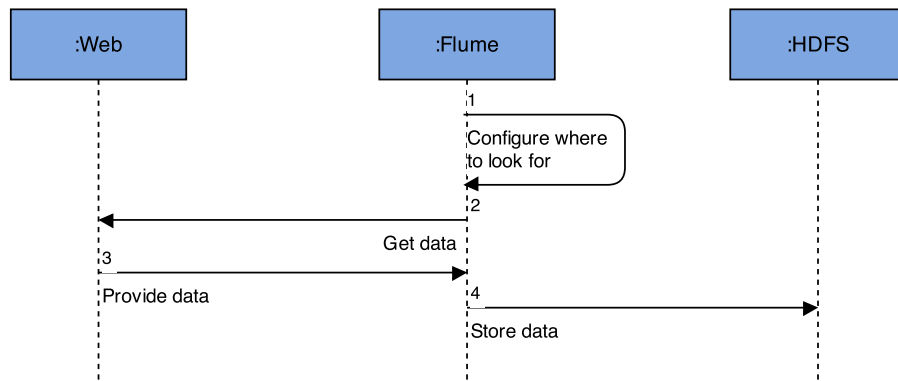


Figure 4.9: Sequence diagram for Flume

When it is used with Twitter as a data source, Flume should be configured with the following parameters:

Consumer key and consumer secret , which are provided by Twitter when a user registers an application on its developers services.

Keywords to look for inside the tweets.

Date , when the tweets were created.

HDFS path to store fetched data.

4.4.2 Pig

As it was explained in section 2.10, Pig is an engine that allows the execution of data processing scripts over Hadoop. These scripts, also called data flows, are written in a data processing oriented language called Pig Latin. As it happens with Flume, Pig users don't have to care about the distribution of storage nor the distributed processing, because these things are handled by Hadoop.

Pig Latin scripts will be used to process the data that were retrieved and stored in HDFS by Flume. Pig Latin offers, as a data processing language, a lot of relational operations and functions to perform fast and reliable data processing.

First, data will be loaded from HDFS into the scripts and without any external or third-party functions, the scripts will filter or group data by different kinds of parameters, such as language, location, date, number of views or responses...

Then, sentiment and/or emotion analysis will be performed over the data that are being processed. This will be achieved thanks to UDFs that have been developed to exploit SEAS capabilities over a Big Data infrastructure. The UDFs that will perform sentiment and/or emotion analysis will be explained in section 4.4.2.1.

Finally, data results will be stored again in HDFS. This process is depicted in Figure 4.10.

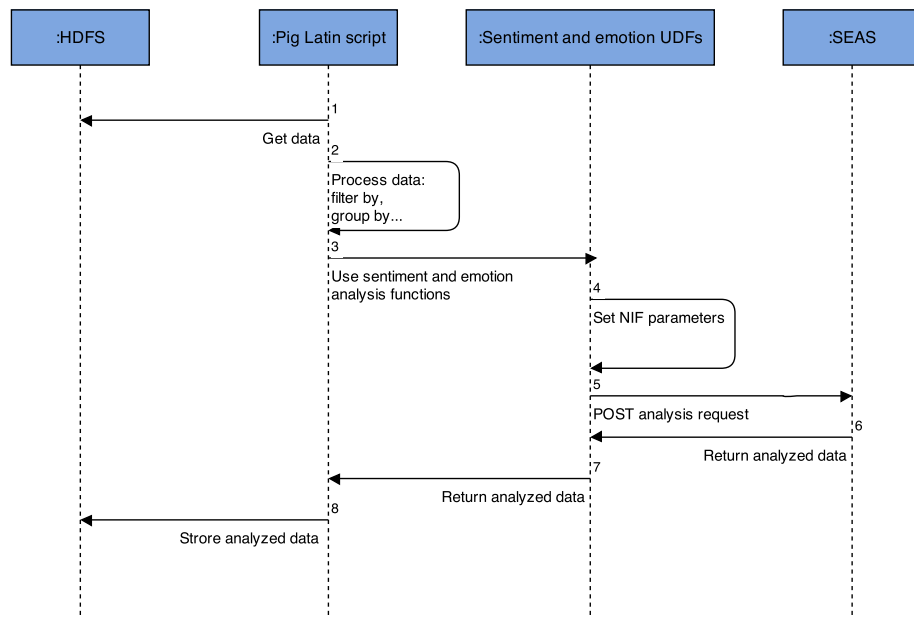


Figure 4.10: Sequence diagram for Pig

4.4.2.1 UDFs

As it was explained in section 2.10.1, UDF stands for User Defined Functions. They can be developed in Java or Python, and they allow the user to create its own data processing functions and export their results into the data flow as a new parameter of the relations.

In this case, two Java UDFs have been developed to perform sentiment and emotion analysis calling SEAS services, so these UDFs are SEAS's clients:

SentimentAnalyzer will be called for each text that is going to be analyzed by the Pig Latin Script. This UDF will set the corresponding NIF parameters, make a POST

request to SEAS sentiment analysis service, receive the response in a JSON and parse it. The analyzed text will be returned with its sentiment polarity, that can be *positive*, *negative* or *neutral*, in a tuple in the format *(text, polarity)*.

EmotionAnalyzer will be called for each text that is going to be analyzed by the Pig Latin script. This UDF will set the corresponding NIF parameters, make a POST request to SEAS emotion analysis service, receive the response in a JSON and parse it. The analyzed text will be returned with its most representative emotion category, that can be *happiness*, *sadness*, *anger*..., in a tuple in the format *(text, category)*.

4.5 Conclusions

We have shown a three tiered client/server architecture, in which there are clearly different modules: SEAS is the server, MongoDB is the database and SAGA (GATE) and SEAS-Hadoop are the clients.

Because of the standardization that NIF offers, each module can be developed and maintained individually, as long as developers respect NIF interfaces to communicate to each other module.

Also, this brings interoperability to each module, so SEAS can be called and used from other clients or SAGA and SEAS-Hadoop can call other sentiment and emotion analysis services that follow NIF API.

All modules are available as open source projects.

- SEAS:
 - <https://github.com/gsi-upm/SEAS>
- Eurosentiment Marl and Onyx generator:
 - <https://github.com/gsi-upm/eurosentiment-generator>
- Onyxemote:
 - <http://demos.gsi.dit.upm.es/onyxemote/>
- SAGA:
 - <https://github.com/gsi-upm/SAGA>
- SEAS-Hadoop:

– <https://github.com/gsi-upm/SEAS/Hadoop>

Case study

In this chapter we are going to describe a selected use case. The running of all the modules involved and its purpose is going to be explained. We are going to cover all the important aspects of the system: how SEAS can be used by different clients, the advantages of using SAGA and the possibilities provided by the use of SEAS in Hadoop.

5.1 Introduction

We are going to see how to use SEAS, SAGA and Hadoop to perform different sentiment and emotion analyses. More details will be given in the following sections.

In this scenario each module will run separately to demonstrate that they all can be standalone applications.

SEAS could be located in any Tomcat server, but for the purpose of this case study is going to be deployed in one of the computers in the laboratory of Grupo de Sistemas Inteligentes. SAGA and GATE will run in my own personal computer. Hadoop and all its associated tools will run in a computer located also in Grupo de Sistemas Inteligentes. This is resumed in table 5.1.

Table 5.1: Execution enviroment

Component	where it runs
SEAS	Laboratory, GSI (demos.gsi.dit.upm.es)
GATE and SAGA	Own personal computer
Hadoop	Laboratory, GSI, (fano.gsi.dit.upm.es)

5.2 SEAS

In this case study, we are going to see how to use SEAS in four different ways: directly making HTTP requests, using Eurosentiment Playground, using the demo available at GSI and performing real-time video analysis. Also, using the last one, we are going to see the performance of all the services available inside SEAS with different texts to be analyzed.

As it was indicated in section 4.2, the purpose of this web service is to provide a set of sentiment and emotion analysis services for text processing. These services are offered with a REST API according to NIF, so their access, input and output formats are standardized so SEAS can be used with others text or data processing tools in an interoperable way.

To use the services, the user has to make a POST request with the parameters described in section 4.2.1 to indicate the text to be analyzed, the input and output format, and the service to be used.

The result of the analysis will be returned in a JSON, which is structured using Marl ontology for sentiment analysis or Onyx ontology for emotion analysis.

5.2.1 Call SEAS using a command line shell

POST requests can be made using *curl* command. For example, if we want to analyze the sentence *A lot of companies have closed in the last year*, we should send a POST request to SEAS indicating that we want to perform sentiment analysis over this text with the English financial analyzer. This is achieved with the following command:

Listing 5.1: "Calling SEAS from command line"

```
curl --data "input=A lot of companies have closed in the last year&intype=direct&
informat=text&outformat=json-ld&algo=enFinancialEmoticon" http://localhost:8080/
RestrictedToNIF/RestrictedService
```

As a response, we will obtain a JSON containing the result of the analysis. This JSON contains valuable information related to the sentiment contained in the analyzed sentence.

The sentence *A lot of companies have closed in the last year* will be classified with a negative polarity and a sentiment value of -1. This information can be extracted from the JSON, in which these polarity and value are expressed using Marl ontology. The polarity is contained in a parameter called *marl:hasPolarity* and the value in another one called *marl:polarityValue*. Both parameters are contained in a *opinion* inside the JSON. See

section 2.3 for more information.

The response described before is the following one:

Listing 5.2: "Sentiment analysis result"

```
{
  "@context": "http://demos.gsi.dit.upm.es/eurosentiment/static/context.jsonld",
  "analysis": [
    {
      "@id": "http://www.gsi.dit.upm.es/ontologies/analysis#SAGA",
      "@type": [
        "marl:SentimentAnalysis"
      ],
      "marl:maxPolarityValue": 1.0,
      "marl:minPolarityValue": -1.0
    }
  ],
  "entries": [
    {
      "nif:isString": "A lot of companies have closed in the last year",
      "opinions": [
        {
          "@id": "_:Opinion1",
          "marl:hasPolarity": "marl:Negative",
          "marl:polarityValue": -1.0,
          "marl:describesObjectFeature": "Overall"
        }
      ],
      "strings": [
        {
          "nif:anchorOf": "closed",
          "nif:beginIndex": 24,
          "nif:endIndex": 29,
          "opinions": {
            "@id": "_:Opinion",
            "marl:hasPolarity": "marl:Negative",
            "marl:polarityValue": -1.0
          }
        }
      ]
    }
  ]
}
```

5.2.2 Call SEAS using Eurosentiment playground

POST requests can be made using *the Eurosentiment playground*¹, <http://demos.gsi.dit.upm.es/eurosentiment-playground/#services>, which is a web service developed by J. Fernando Sánchez-Rada for the Eurosentiment hackathon that took place at ETSIT, UPM. It allows the user to easily create HTTP requests to call NIF based services, so SEAS services can be used using this web service.

The Eurosentiment playground allows the user to perform HTTP requests indicating the following:

HTTP request to be performed. Its value can be *GET*, *POST*, *PUT* and *DELETE*.

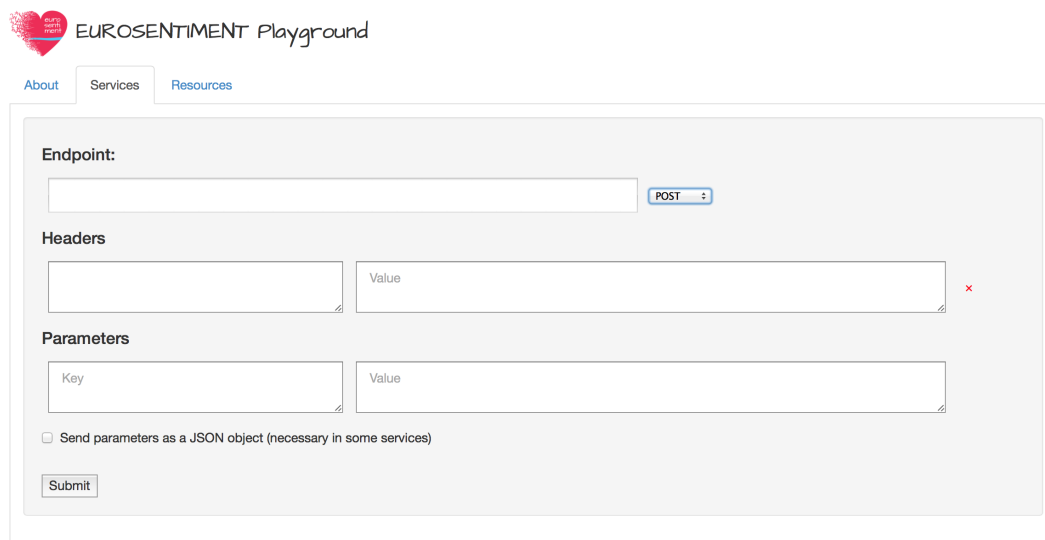
Endpoint of the service that is going to be used.

Headers , like an API key.

Parameters that are going to be added to the HTTP request.

Send parameters as a JSON , needed in some services.

The service looks like depicted in Figure 5.1.



The screenshot shows the 'EUROSENTIMENT Playground' web interface. At the top, there is a navigation bar with 'About', 'Services', and 'Resources' tabs. The main form area is titled 'Endpoint:' and contains a text input field for the endpoint and a dropdown menu currently set to 'POST'. Below this is a 'Headers' section with a table-like structure for adding headers, showing a 'Key' input and a 'Value' input field. Underneath is a 'Parameters' section with a similar 'Key' and 'Value' input structure. At the bottom of the form, there is a checkbox labeled 'Send parameters as a JSON object (necessary in some services)' and a 'Submit' button.

Figure 5.1: The Eurosentiment playground

We can make the exact same request performed in the example of section 5.2.1 using the playground. In this case, the playground would look like depicted in Figure 5.2.

¹This demonstrator has been made possible by the Eurosentiment project

The screenshot shows the Eurosentiment playground interface. At the top, the 'Endpoint' is set to `http://demos.gsi.dit.upm.es/tomcat/RestrictedToNIF/RestrictedService` with a 'POST' method selected. Below this is a 'Headers' section with a table for key-value pairs. The 'Parameters' section contains a table with five rows of key-value pairs. A tooltip points to the 'input' parameter, stating: 'Extra headers for the request, mostly for authentication (x-eurosentiment-token)'. At the bottom, there is a checkbox for 'Send parameters as a JSON object (necessary in some services)' and a 'Submit' button.

Endpoint:	
<code>http://demos.gsi.dit.upm.es/tomcat/RestrictedToNIF/RestrictedService</code> POST	
Headers	
	Value
Parameters	
input	A lot of companies have closed in the last year
informat	text
intype	direct
outformat	json-ld
algo	enFinancialEmoticon

☐ Send parameters as a JSON object (necessary in some services)

Submit

Figure 5.2: The Eurosentiment playground with a POST request

Then, the request can be submitted and a response like the one shown before will be returned inside the playground.

5.2.3 Call SEAS using the demo available at GSI

SEAS services can be used directly from a demo that has been developed by me and it is located at `http://demos.gsi.dit.upm.es/SEAS/Controller`. Its user interface looks like depicted in Figure 5.3.

This web interface has the following elements:

A list of available services in which you can choose the sentiment or emotion analysis service that is going to be used.

A text box to introduce the text that is going to be analyzed.

An analyze button to send the POST request to SEAS and perform the analysis.

A result box in which the analysis response is going to be shown. The color of the box depends on the polarity of the analyzed text: red for negative, green for positive, blue for neutral and yellow for emotion results.

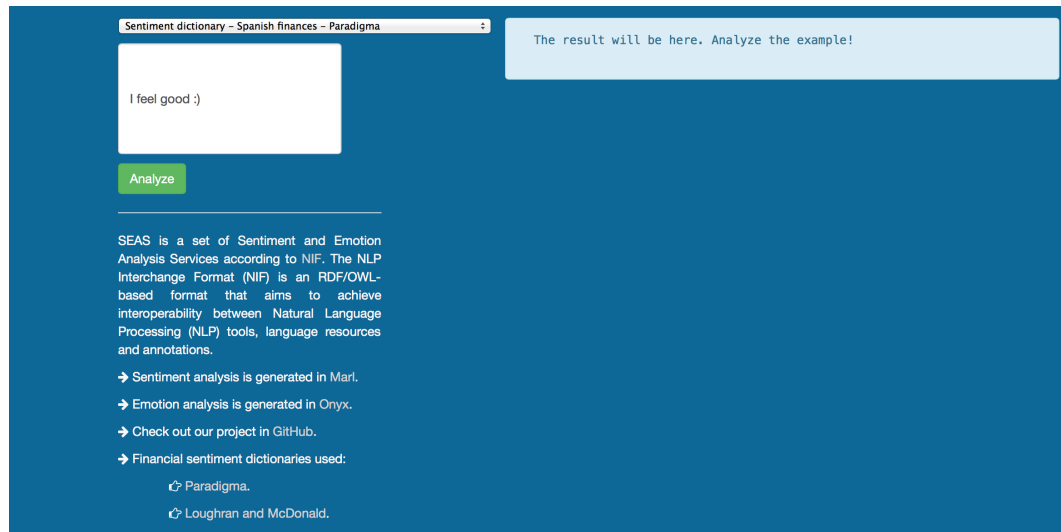


Figure 5.3: SEAS demo

Now that we know how this demo works, we are going to analyze different text with the different services provided by SEAS that have been explained in sections 4.2.2 and 4.2.3. To do so, we are going to select each service, introduce an appropriate input and observe the results:

- Sentiment analysis:
 - English finances dictionaries provided by Loughran and McDonald:
 - * Input: A lot of companies have closed in the last year.
 - * Result: negative polarity with a value of -1.
 - * `"marl:hasPolarity": "marl:Negative",`
`"marl:polarityValue": -1.0,`
 - Spanish finances dictionaries provided by Paradigma:
 - * Input: El valor de muchas empresas sube en bolsa.
 - * Result: positive polarity with a value of 1.
 - * `"marl:hasPolarity": "marl:Positive",`
`"marl:polarityValue": 1.0,`
 - Emoticon dictionaries:
 - * Input: I do not feel anything :—
 - * Result: neutral polarity with a value of 0.
 - * `"marl:hasPolarity": "marl:Neutral",`
`"marl:polarityValue": 0.0,`

- English finances and emoticon dictionaries:
 - * Input: My company is in recession, but I stay positive. :)
 - * Result: positive polarity with a value of 0.33.
 - *

```
"marl:hasPolarity": "marl:Positive",  
"marl:polarityValue": 0.3333333333333333,
```
- Spanish finances and emoticon dictionaries:
 - * Input: El valor de mi empresa unas veces sube y otras cae. :(
 - * Result: negative polarity with a value of -0.33.
 - *

```
"marl:hasPolarity": "marl:Negative",  
"marl:polarityValue": -0.3333333333333333,
```
- Emotion analysis:
 - Onyxemote service:
 - * Input: I feel afraid of what is going to happen to us.
 - * Result:
 1. Fear:
 - ```
"@value": "0.6"
```
      2. Anger:
        - ```
"@value": "0.039705882352941"
```
 3. Happiness:
 - ```
"@value": "0.039705882352941"
```

#### 5.2.4 Call SEAS to analyze videos in real-time

This is a service developed by Ronald Gil, who is a student in Massachusetts Institute of Technology at this moment, during his stay at Group on Intelligent Systems (GSI).

This project, <https://github.com/gsi-upm/video-sentiment-analysis>, consist on a real-time video analyzer that uses SEAS API to perform sentiment analysis over a video in real-time.

First, the service allow us to do the following things, as we can see in Figure 5.4:

1. Select the video that we want to analyze from YouTube.
2. Select the sentiment analysis service that we want to use.

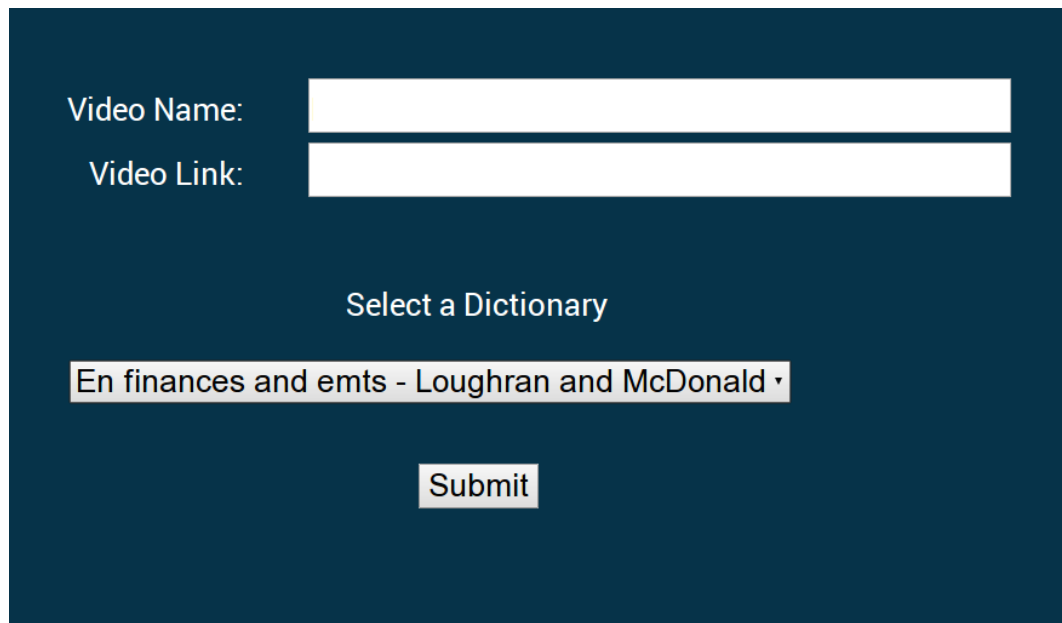
A dark blue rectangular form with white text and input fields. At the top left, there are two labels: 'Video Name:' and 'Video Link:', each followed by a white rectangular input field. Below these, centered, is the text 'Select a Dictionary'. Underneath this text is a white dropdown menu showing the selected option 'En finances and emts - Loughran and McDonald' with a small downward arrow on the right. At the bottom center of the form is a white rectangular button with the text 'Submit' in dark blue.

Figure 5.4: Real-time video sentiment analyzer options

3. Submit our request.

Then, the video will be analyzed in real time as depicted in Figure 5.5.

This is possible because the service parses the audio of the video, sends each sentence to SEAS in order to analyze it and shows the evolution of the sentiment polarity in a graphic below the video.



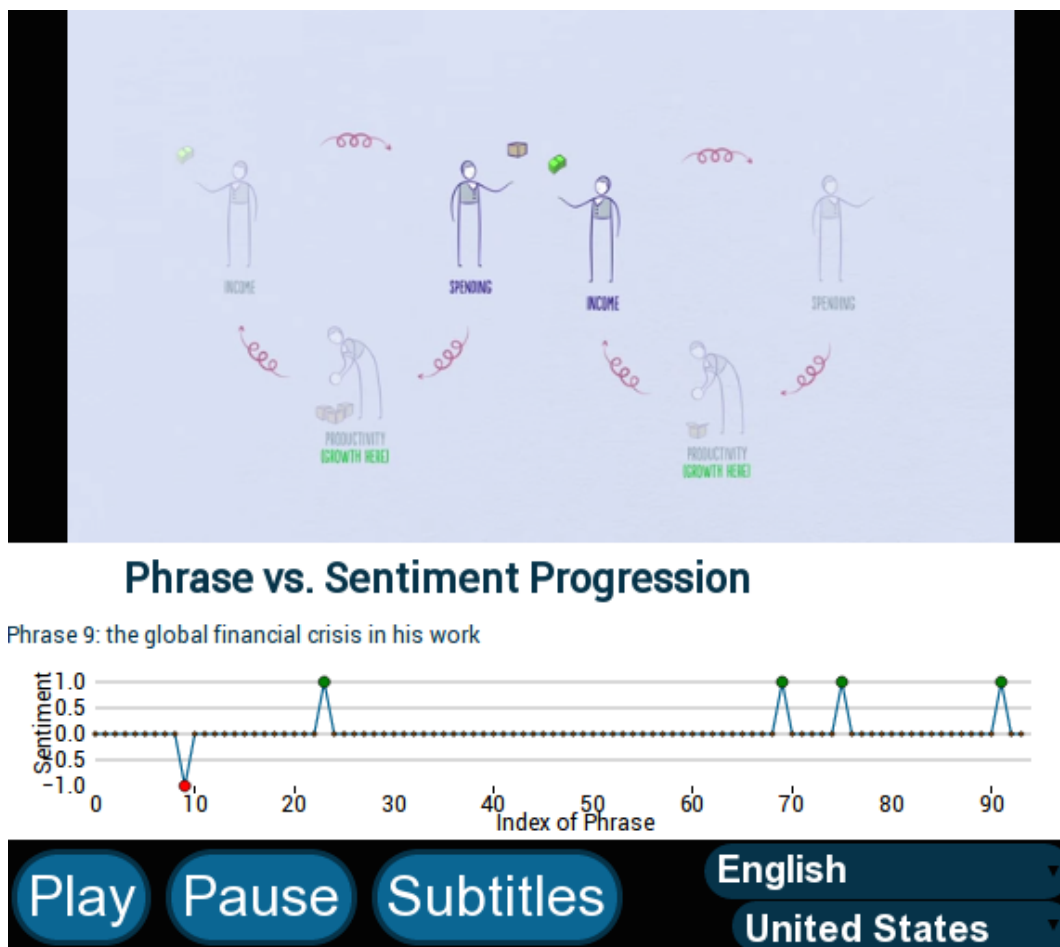


Figure 5.5: Real-time video sentiment analyzer performance

### 5.3 SAGA

In this case study, we are going to present a corpus, we are going to show how to perform sentiment and emotion analysis over it using SAGA's processing resources, then we are going to validate this analysis and finally we are going to repeat the process with each service available at SEAS.

As it was indicated in section 4.3, SAGA is a set of processing and linguistic resources, written in Java, developed to run sentiment and emotion analysis over text using GATE platform and it is distributed as a plugin. The main purpose of SAGA is to provide a SEAS's client that can be used from GATE, so it can perform sentiment and emotion analysis by calling SEAS or other similar NIF's services such as the available at Eurosentiment Portal, <http://portal.eurosentiment.eu>

As we have said in section 2.7, GATE is a platform developed by The University of

Sheffield that brings a lot of capabilities to perform processing tasks over text. It offers a lot of different plugins and processing resources, some of them developed by them and others developed by third-parties, to perform text analysis over big sets of documents.

### 5.3.1 Corpus

The selected corpus is composed by a total of 165 financial texts that have been studied in an article titled "Extracting Investor Sentiment from Weblog Texts: A Knowledge-based Approach" [18]. The corpus is divided in two folders:

**pos** contains 63 positive text documents.

**neg** contains 102 negative text documents.

We load each folder inside GATE, creating one corpus for the positive documents and other corpus for the negative ones. To do so, take a look at Appendix B. Once the corpuses have been loaded, they look like depicted in Figure 5.6.

If we look at each document that has been loaded inside each corpus, we can see that each one of them has an *annotation type* named *paragraph* (Figure 5.7). Also, we can see that there is only one *paragraph* annotation and it has empty *features*. There is where the sentiment or emotion annotation is going to be.

### 5.3.2 Finance sentiment analysis calling SEAS

To perform sentiment analysis over the loaded corpuses, we are going to create a new GATE application or pipeline with the following processing resource:

**Sentiment and emotion analysis calling SEAS** is a processing resources provided by SAGA. It allows us to perform the sentiment and emotion analyses provided by SEAS. It is the processing resource that we are going to use to analyze each document inside the corpus. In this case, we are going to perform an English finance analysis. The result analysis annotation will be created inside the original *paragraph* annotation. In order to do that, this processing resource is configured with the runtime parameters depicted in Figure 5.8 and 5.9.

As we can see, emotion analysis is set as false and sentiment analysis is set as true. Now, the user runs the application over the corpus.













| Index | Document name                                                                                                                                             |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0     |  0018_Per1_2008-08-23_japanese-yen-hedging-against-falling.txt_000B9     |
| 1     |  0034_Per1_2008-09-15_wheres-s-500-support-after-todays-big.txt_000BA    |
| 2     |  0049_Per1_2008-10-06_week-ahead-trepidation.txt_000BB                   |
| 3     |  0051_Per1_2008-10-07_real-price-of-gold-soars.txt_000BC                 |
| 4     |  0052_Per1_2008-10-07_stocks-in-worst-yearly-slump-since.txt_000BD       |
| 5     |  0056_Per1_2008-10-08_stock-market-is-about-where-it-should.txt_000BE    |
| 6     |  0061_Per1_2008-10-10_boy-who-cried-wolf.txt_000BF                       |
| 7     |  0062_Per1_2008-10-11_some-historical-bear-market-perspective.txt_000C0  |
| 8     |  0063_Per1_2008-10-12_what-is-graham-pe-telling-us.txt_000C1             |
| 9     |  0070_Per1_2008-10-14_crash-warning-part-ii.txt_000C2                    |
| 10    |  0077_Per1_2008-10-22_s-recession-forecast.txt_000C3                     |
| 11    |  0079_Per1_2008-10-22_market-ticker-have-we-reached-bottom.txt_000C4     |
| 12    |  0089_Per1_2008-11-04_csr-and-financial-crisis-taking-stock.txt_000C5    |
| 13    |  0090_Per1_2008-11-04_equity-mutual-fund-flows-and-uncanny.txt_000C6   |
| 14    |  0096_Per1_2008-11-12_dow-jones-close-111208-stock-market.txt_000C7    |
| 15    |  0100_Per1_2008-11-16_historical-annual-returns-for-s-500_14.txt_000C8 |
| 16    |  0107_Per1_2008-11-25_state-of-new-jersey-is-insolvent.txt_000C9       |

Figure 5.6: Sample of documents inside a GATE corpus

When the app is run, this processing resource will make a POST request to the selected SEAS service for each document. The service will take each POST request, will perform the analysis, will send the intermediate results to Eurosentiment Marl Generator and it will return a JSON in Marl format with the analyzed document. For the negative document used as a sample in this case study, the generated JSON will look like this:

Listing 5.3: "Example of negative document in json and Marl"

```
{
 "@context": "http://demos.gsi.dit.upm.es/eurosentiment/static/context.jsonld",
 "analysis": [
 {
 "@id": "http://www.gsi.dit.upm.es/ontologies/analysis#SAGA",
 "@type": [
 "marl:SentimentAnalysis"
],

```

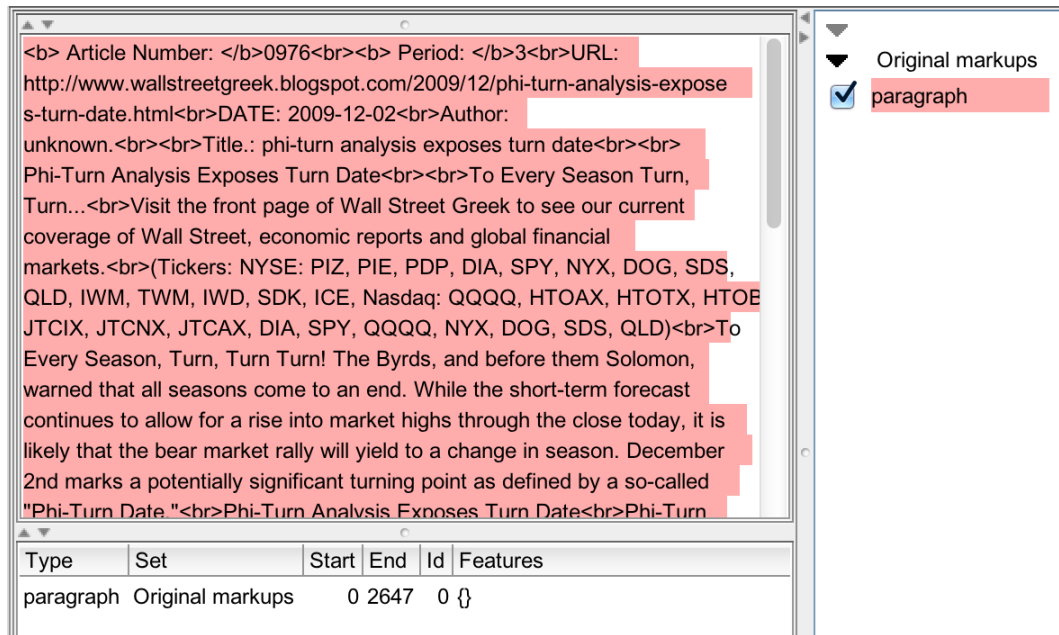


Figure 5.7: Sample of a document loaded in GATE

| Runtime Parameters for the "Sentiment and emotion analysis calling SEAS and Eurosentiment" |                  |          |                          |  |
|--------------------------------------------------------------------------------------------|------------------|----------|--------------------------|--|
| Name                                                                                       | Type             | Required | Value                    |  |
| APIKey                                                                                     | String           |          |                          |  |
| annotationTypes                                                                            | String           |          | paragraph                |  |
| apiKeyName                                                                                 | String           |          | x-eurosentiment-token    |  |
| emotionAlgorithm                                                                           | EmotionAlgorithm |          | auto                     |  |
| emotionAnalysis                                                                            | Boolean          |          | false                    |  |
| emotionCategoryName                                                                        | String           |          | onyx:hasEmotionCategory  |  |
| emotionServiceURL                                                                          | String           |          |                          |  |
| emotionValueName                                                                           | String           |          | onyx:hasEmotionIntensity |  |

Figure 5.8: Sample of Sentiment and emotion analysis calling SEAS PR configuration

```
"marl:maxPolarityValue": 1.0,
 "marl:minPolarityValue": -1.0
},
 "entries": [
 {
 "nif:isString": " Article Number: 0976
Period: 3
```








|                                                                                   |                       |                     |                                                         |
|-----------------------------------------------------------------------------------|-----------------------|---------------------|---------------------------------------------------------|
|  | inputASName           | String              | Original markups                                        |
|  | sentimentAlgorithm    | SentimentAlgorithm  | Dictionary                                              |
|  | sentimentAnalysis     | Boolean             | true                                                    |
|  | sentimentDictionary   | SentimentDictionary | English_finances_and_Emoticon                           |
|  | sentimentPolarityName | String              | marl:hasPolarity                                        |
|  | sentimentServiceURL   | String              | http://localhost:8080/RestrictedToNIF/RestrictedService |
|  | sentimentValueName    | String              | marl:polarityValue                                      |

Figure 5.9: Sample of Sentiment and emotion analysis calling SEAS PR configuration

```

URL: http://www.wallstreetgreek.blogspot.com/2009/12/phi-turn-analysis-exposes-turn-date
.html
DATE: 2009-12-02
Author: unknown.

Title.: phi-turn analysis exposes turn date

Phi-Turn Analysis Exposes Turn Date

To Every Season Turn, Turn...
Visit the front page of Wall Street Greek to see our current coverage of Wall Street,
economic reports and global financial markets.
(Tickers: NYSE: PIZ, PIE, PDP, DIA, SPY, NYX, DOG, SDS, QLD, IWM, TWM, IWD, SDK, ICE,
Nasdaq: QQQQ, HTOAX, HTOTX, HTOBX, JTCIX, JTCNX, JTCAX, DIA, SPY, QQQQ, NYX, DOG,
SDS, QLD)
To Every Season, Turn, Turn Turn! The Byrds, and before them Solomon, warned that all
seasons come to an end. While the short-term forecast continues to allow for a rise
into market highs through the close today, it is likely that the bear market rally
will yield to a change in season. December 2nd marks a potentially significant
turning point as defined by a so-called \"Phi-Turn Date.\"
Phi-Turn Analysis Exposes Turn Date
Phi-Turn Analysis is just one method of identifying market inflection points. These
methods range from examination of lunar cycles, to other astrological events and
onto more believable analysis of cyclic content in the market indices. Phi Turn
Analysis was conceived by Dr. Robert McHugh, and relies on Fibonacci ratios to
establish market turning points. The basis for the calculation begins with the
significant top established in 2000.
Throughout 2008, the calculated Phi Turn dates have fallen quite remarkably on
significant tops and bottoms in the market. While not every date marks the onset of
a multi-month reversal, almost no reversal has happened on a day that has not
matched the Phi Turn calculation result.
NOTE: This article is an amendment to the most recent \" S&P 500 Index Winter Forecast
.\" Through the description of Phi Turn Analysis, this completes the series on the
search for bear market rally top. We realized Turn Analysis had not been addressed
in the article series.

Article may interest investors in NYSE: PIZ, NYSE: PIE, NYSE: PDP, NYSE: DIA, NYSE: SPY,
NYSE: NYX, NYSE: DOG, NYSE: SDS, NYSE: QLD, NYSE: IWM, NYSE: TWM, NYSE: IWD, NYSE:

```

SDK, NYSE: ICE, Nasdaq: QQQQ, Nasdaq: HTOAX, Nasdaq: HTOTX, Nasdaq: HTOBX, Nasdaq: JTCIX, Nasdaq: JTCNX, Nasdaq: JTCAX. Please see our disclosures at the Wall Street Greek website and author bio pages found there. This article and website in no way offers or represents financial or investment advice. Information is provided for entertainment purposes only.

posted by Greek | 10:42 AM

```
",
 "opinions": [
 {
 "@id": "_:Opinion1",
 "marl:hasPolarity": "marl:Negative",
 "marl:polarityValue": -1.0,
 "marl:describesObjectFeature": "Overall"
 }
],
 "strings": [
 {
 "nif:anchorOf": "exposes",
 "nif:beginIndex": 220,
 "nif:endIndex": 226,
 "opinions": {
 "@id": "_:Opinion",
 "marl:hasPolarity": "marl:Negative",
 "marl:polarityValue": -1.0
 }
 },
 {
 "nif:anchorOf": "Exposes",
 "nif:beginIndex": 264,
 "nif:endIndex": 270,
 "opinions": {
 "@id": "_:Opinion",
 "marl:hasPolarity": "marl:Negative",
 "marl:polarityValue": -1.0
 }
 },
 {
 "nif:anchorOf": "warned",
 "nif:beginIndex": 709,
 "nif:endIndex": 714,
 "opinions": {
 "@id": "_:Opinion",
 "marl:hasPolarity": "marl:Negative",
 "marl:polarityValue": -1.0
 }
 },
 {
 "nif:anchorOf": "Exposes",
 "nif:beginIndex": 1049,
 "nif:endIndex": 1055,
 "opinions": {
 "@id": "_:Opinion",
```

```

 "marl:hasPolarity": "marl:Negative",
 "marl:polarityValue": -1.0
 }
}
]
}
]
}
}

```

With this result, the processing resource parses the JSON in order to extract the relevant sentiment information and adds the sentiment annotation to the document. As depicted in Figure 5.10, it takes the polarity of the document that is inside the JSON attribute called *marl:hasPolarity* and the value of this polarity called *marl:polarityValue*.

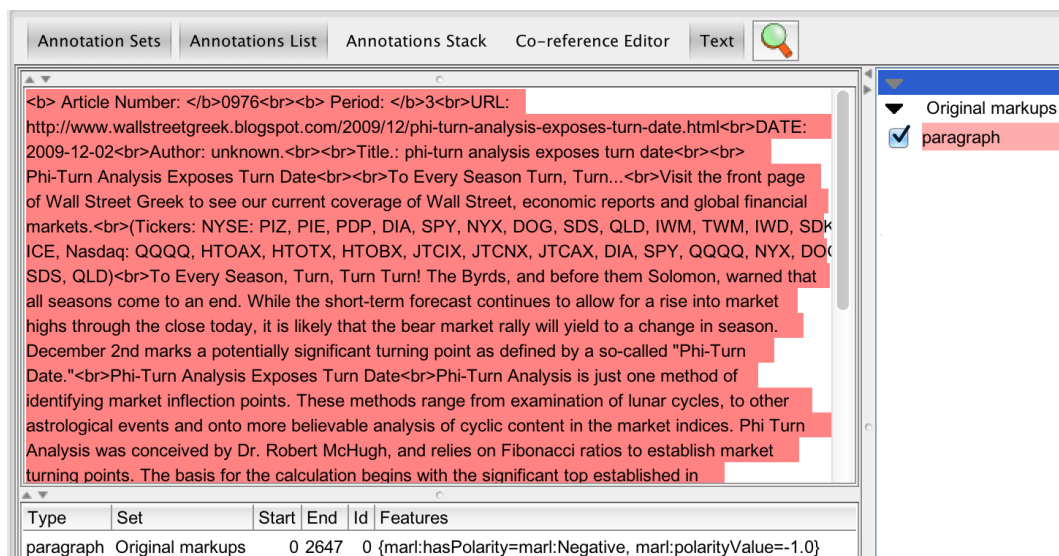


Figure 5.10: Example of an analyzed negative document

### 5.3.2.1 Validation of the sentiment analysis

Now that we have analyzed the corpus, we can measure how good the English financial sentiment analysis provided by SEAS is. To do so, we are going to use the classified corpus to compare the correct classification with the one that SEAS has performed.

But having the documents classified by its actual polarity in different corpus is not enough. We will classify the documents using the same type of annotations that SAGA processing resources have created.

We are going to copy the annotations to have two of them: one *paragraph* annotation is going to be the actual sentiment polarity of the text and the other *paragraph* annotation

is going to be the sentiment polarity returned by the analysis service.

To do so, we create a new GATE application or pipeline with two processing resources that are going to be executed over the corpus in the following order:

**Annotation set transfer** is a processing resources provided by Tools plugin. It allows us to make copies of Annotation Types. It is the processing resource that we are going to use to copy the *paragraph* annotation to have two of them. In order to do that, this processing resource is configured with the runtime parameters depicted in Figure 5.11.








| Runtime Parameters for the "Annotation Set Transfer_0000A" Annotation Set Transfer:                        |           |          |                  |
|------------------------------------------------------------------------------------------------------------|-----------|----------|------------------|
| Name                                                                                                       | Type      | Required | Value            |
|  annotationTypes          | ArrayList |          | [paragraph ]     |
|  copyAnnotations          | Boolean   | ✓        | true             |
|  inputASName              | String    |          | Original markups |
|  outputASName           | String    |          | Transferred      |
|  tagASName              | String    |          | Original markups |
|  textTagName            | String    |          |                  |
|  transferAllUnlessFound | Boolean   | ✓        | true             |

Figure 5.11: Sample of Annotation Set Transfer PR configuration

**Predefined Sentiment Annotation** is a processing resources provided by SAGA. It allows us to create the same predefined annotation over an entire corpus. It is the processing resource that we are going to use to annotate every document inside the negative corpus as negative, and every document inside the positive corpus as positive. The annotation will be created inside the copied *paragraph* annotation. In order to do that, this processing resource is configured with the runtime parameters depicted in Figure 5.12.

When both processing resources are configured, the application is executed over each corpus (positive and negative) and each document is annotated as be can see in Figure ???. As a result, each document will have two *paragraph* annotations, one with its actual sentiment polarity in Marl format, an the other one with the one provided by SEAS.







| Runtime Parameters for the "Predefined Sentiment Annotation PR_                                         |        |          |                  |
|---------------------------------------------------------------------------------------------------------|--------|----------|------------------|
| Name                                                                                                    | Type   | Required | Value            |
|  annotationType        | String |          | paragraph        |
|  inputASname           | String |          | Transferred      |
|  sentimentPolarity     | String |          | marl:Negative    |
|  sentimentPolarityName | String |          | marl:hasPolarity |

Figure 5.12: Sample of Predefined Sentiment Annotation PR configuration



The screenshot displays the SAGA interface with the 'Annotations List' tab selected. The main text area shows a document snippet with various HTML tags and a red background. The right sidebar shows a list of annotations with checkboxes for 'Original markups', 'Transferred', and 'paragraph'. The bottom table shows the results of the analysis.

| Type      | Set              | Start | End  | Id | Features                                                  |
|-----------|------------------|-------|------|----|-----------------------------------------------------------|
| paragraph | Original markups | 0     | 2647 | 0  | {marl:hasPolarity=marl:Negative, marl:polarityValue=-1.0} |
| paragraph | Transferred      | 0     | 2647 | 0  | {marl:hasPolarity=marl:Negative}                          |

Figure 5.13: Example of an analyzed negative document and its real polarity

Now that all the documents have been analyzed, we can make use of a GATE tool called Corpus QA, which allows us to see how accurate the service was according to the real polarity of the documents. In the case of the negative ones, as we can see in Figure 5.14, 78 out of 102 documents are classified as negative, 2 as neutral and 8 as positive. That validates the service with an accuracy of 76.47% and a F-score of 0.901, which is pretty good for a dictionary based sentiment analyzer.

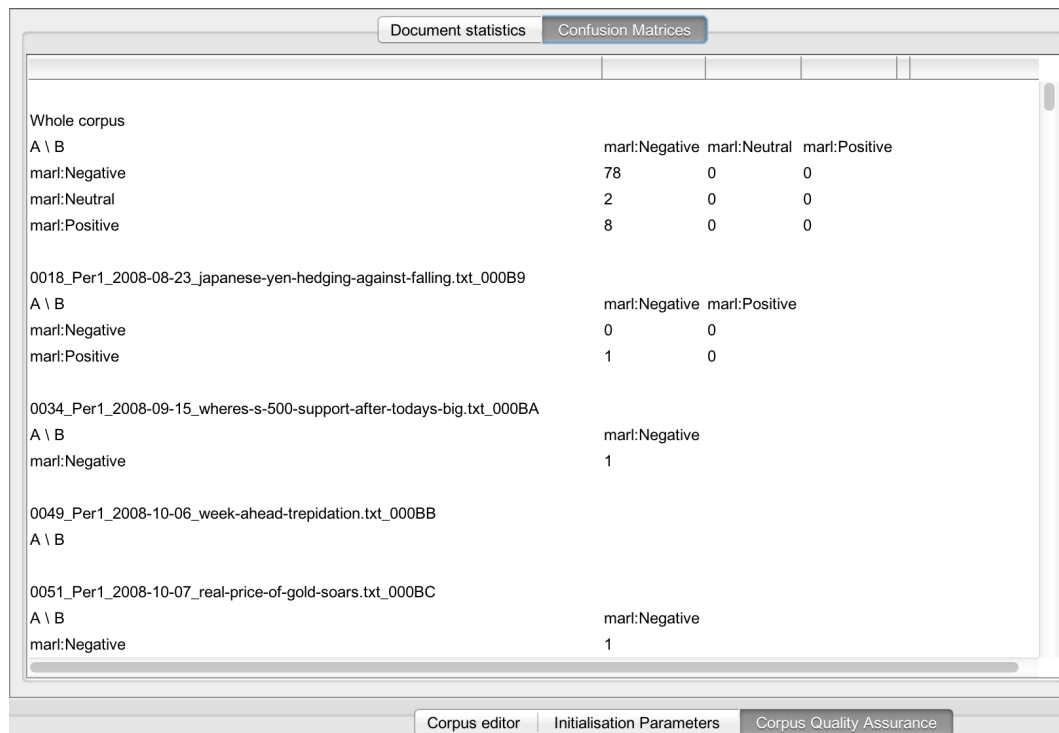


Figure 5.14: Negative corpus Quality Assurance

### 5.3.3 Other sentiment analysis services calling SEAS

As it was explained in section 4.2.2, there are different sentiment analysis services that can be called using SEAS. All of them can be called using the SAGA's processing resource used in section 5.3.2.

To do so, only a few runtime parameters should be configured:

**sentimentAlgorithm** set to *Dictionary*.

**sentimentAnalysis** set to *true*.

**sentimentDictionary** set to one of the services listed in Figure 5.15.

**sentimentServiceURL** set to one of the SEAS's endpoints described in section 4.2.1.

The results provided by the use of these different algorithms will be analogue to the one achieved in section 5.3.2.


|                                                                                                       |                     |                                    |
|-------------------------------------------------------------------------------------------------------|---------------------|------------------------------------|
|  sentimentAlgorithm  | SentimentAlgorithm  | Spanish_finances_Paradigma         |
|  sentimentAnalysis   | Boolean             | English_finances_Loughran_McDonald |
|  sentimentDictionary | SentimentDictionary | Emoticon                           |
|                                                                                                       |                     | Spanish_finances_and_Emoticon      |
|                                                                                                       |                     | English_finances_and_Emoticon      |

Figure 5.15: List of sentiment analysis services

### 5.3.4 Emotion analysis calling Onyxemote

Now, we are going to perform emotion analysis over an example document that contains the following three sentences:

1. I am so happy for you!
2. My best friend is scared of spiders.
3. Thanks for this enormous surprise party!

To do so, we are going to reconfigure the following processing resource:

**Sentiment and emotion analysis calling SEAS** is a processing resources provided by SAGA. It allows us to perform the sentiment and emotion analyses provided by SEAS. It is the processing resource that we are going to use to analyze each document inside the corpus. In this case, we are going to perform an emotion analysis. The result analysis annotation will be created inside the original *paragraph* annotation. In order to do that, this processing resource is configured with the runtime parameters depicted in Figure 5.16 and 5.17.

As we can see, emotion analysis is set as true and sentiment analysis is set as false.

When the app is run, this processing resource will make a GET request to Onyxemote service for each sentence. The service will take each GET request, will perform the analysis and it will return a JSON in Onyx format with the analyzed document.

The results returned from Onyxemote will be the following:

1. I am so happy for you!
  - Emotion category: happiness.
  - Value: 0.75.










| Name                                                                                                  | Type             | Required | Value                        |
|-------------------------------------------------------------------------------------------------------|------------------|----------|------------------------------|
|  APIKey              | String           |          | XXXXXXXXXXXXXXXXXXXXXXXXXXXX |
|  annotationTypes     | String           |          | text                         |
|  apiKeyName          | String           |          | x-eurosentiment-token        |
|  emotionAlgorithm    | EmotionAlgorithm |          | onyx                         |
|  emotionAnalysis     | Boolean          |          | true                         |
|  emotionCategoryName | String           |          | onyx:hasEmotionCategory      |
|  emotionServiceURL   | String           |          |                              |
|  emotionValueName    | String           |          | onyx:hasEmotionIntensity     |
|  inputASname         | String           |          | Original markups             |

Figure 5.16: Sample of Sentiment and emotion analysis calling SEAS PR configuration



|                                                                                                           |                     |                                                         |
|-----------------------------------------------------------------------------------------------------------|---------------------|---------------------------------------------------------|
|  sentimentAlgorithm      | SentimentAlgorithm  | Dictionary                                              |
|  sentimentAnalysis      | Boolean             | false                                                   |
|  sentimentDictionary   | SentimentDictionary | English_finances_and_Emoticon                           |
|  sentimentPolarityName | String              | marl:hasPolarity                                        |
|  sentimentServiceURL   | String              | http://localhost:8080/RestrictedToNIF/RestrictedService |
|  sentimentValueName    | String              | marl:polarityValue                                      |

Figure 5.17: Sample of Sentiment and emotion analysis calling SEAS PR configuration

2. My best friend is scared of spiders.

- Emotion category: fear.
- Value: 0.5.

3. Thanks for this enormous surprise party!

- Emotion category: surprise.
- Value: 0.75.

These results can be seen as depicted in Figure 5.18. To produce them, the processing resource takes the emotion category of the document that is inside the JSON attribute called *onyx:hasEmotionCategory* and the value of this category called *onyx:hasEmotionIntensity*.

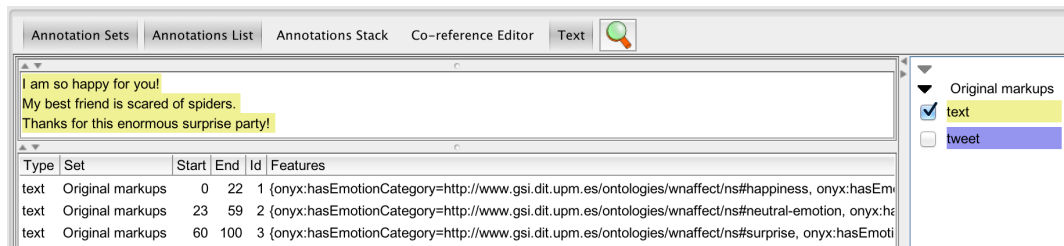


Figure 5.18: Sample of an analyzed emotion document

### 5.3.5 Sentiment analysis calling Eurosentiment services

As we have explained in section 4.3.1.2, the processing resource called Sentiment and Emotion analysis calling SEAS can be used to call other NIF based services that perform sentiment and emotion analysis and return their results using Marl and Onyx ontologies.

The Eurosentiment portal, <https://portal.eurosentiment.eu/service/list#>, offers different services to perform sentiment and emotion analysis as we can see in Figure 5.19.

[Home](#) » [Active services](#)

These services are available to the users of the ESRP. Please refer to the detail in each individual services for specific conditions. Note that the ESRP works an aggregation platform and cannot be responsible for the SLA of the individual services. For information on how to publish a new service [here](#)

| Name                              | Description                                                     | Language                                                         | Domain               | Free | Provider              |
|-----------------------------------|-----------------------------------------------------------------|------------------------------------------------------------------|----------------------|------|-----------------------|
| PT - Domain detection             | Domain detector for Hotels and Electronics domains              | English<br>French<br>Spanish<br>Italian<br>Portuguese<br>Catalan | Hotel<br>Electronics | Yes  | Paradigma Tecnológico |
| ES - Domain detection             | Domain detection service developed by Expert System             | English<br>German<br>Italian                                     | Cars<br>Food         | Yes  | Expert System         |
| ES - Language detection           | Language detection service developed by Expert System           | English<br>German<br>Italian                                     | Cars<br>Food         | Yes  | Expert System         |
| ES - Sentiment & Emotion analysis | Sentiment & Emotion analysis service developed by Expert System | English<br>German<br>Italian                                     | Cars<br>Food         | Yes  | Expert System         |

Figure 5.19: Eurosentiment services

Right now, the available services to perform this kind of analysis are:

- Sentiment and Emotion analysis service developed by Expert System.
- Wordnet Affect based emotion analysis developed by Paradigma Tecnológico.
- Sentiment and emotion (Wordnet Affect based) analysis developed by Paradigma Tecnológico.

So, if we want to perform sentiment analysis over an example document that contains the following sentence using the service provided by Expert System:

1. The iPad is a fantastic device.

We are going to configure the processing resource called *Sentiment and emotion analysis calling SEAS* as depicted in Figures 5.20 and 5.21.










| Runtime Parameters for the "Sentiment and emotion analysis calling SEAS and Eurosentiment"              |                  |          |                              |
|---------------------------------------------------------------------------------------------------------|------------------|----------|------------------------------|
| Name                                                                                                    | Type             | Required | Value                        |
|  APIKey                | String           |          | XXXXXXXXXXXXXXXXXXXXXXXXXXXX |
|  annotationTypes       | String           |          | text                         |
|  apiKeyName            | String           |          | x-eurosentiment-token        |
|  emotionAlgorithm      | EmotionAlgorithm |          | auto                         |
|  emotionAnalysis       | Boolean          |          | false                        |
|  emotionCategoryName | String           |          | onyx:hasEmotionCategory      |
|  emotionServiceURL   | String           |          |                              |
|  emotionValueName    | String           |          | onyx:hasEmotionIntensity     |
|  inputASname         | String           |          | Original markups             |

Figure 5.20: Runtime Eurosentiment parameters 1







|                                                                                                           |                     |                                                                                  |
|-----------------------------------------------------------------------------------------------------------|---------------------|----------------------------------------------------------------------------------|
|  sentimentAlgorithm    | SentimentAlgorithm  | Dictionary                                                                       |
|  sentimentAnalysis     | Boolean             | true                                                                             |
|  sentimentDictionary   | SentimentDictionary | English_finances_and_Emoticon                                                    |
|  sentimentPolarityName | String              | marl:hasPolarity                                                                 |
|  sentimentServiceURL   | String              | http://217.26.90.243:8080/EuroSentimentServices/services/server/access/sesse0328 |
|  sentimentValueName    | String              | marl:polarityValue                                                               |

Figure 5.21: Runtime Eurosentiment parameters 2

As we can see, emotion analysis is set as false and sentiment analysis is set as true.

When the app is run, this processing resource will make a POST request to *Sentiment and Emotion analysis service developed by Expert System*, it will perform the analysis and it will return a JSON in Marl format with the analyzed document.

The result returned from this service will be the following:

1. The iPad is a fantastic device.
  - Sentiment polarity: neutral.
  - Value: 50.

## 5.4 Hadoop for financial analysis

Now we are going to see how Hadoop can be used to perform sentiment and emotion analysis over tweets extracted from the social network called Twitter. To do so: we are going to show how to use Flume to obtain these tweets and then we are going to use Pig Latin and UDFs to process them.

### 5.4.1 Using Flume to obtain data from Twitter

We are going to configure Flume to retrieve financial data using Twitter as the source of these data.

First, the user needs a developer Twitter account and to register a new application in Twitter's website. When the app registration is done, Twitter will provide the user with four tokens that should be put inside Flume configuration.

Then, the user should set Twitter as a data source, the date when the data that wants to retrieve were written and the keywords that the data should contain. In this case, we want to obtain financial information, so these keywords would be finance related, like for example, names of banks.

Finally, the user should set the path to the HDFS location where these retrieved data are going to be stored.

This process described before, it is done by the configuration of *flume.conf* file inside the Flume installation:

Listing 5.4: "Example of Flume configuration"

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS
```

```
TwitterAgent.sources.Twitter.type = com.cloudera.flume.source.
 TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey = consumerKey
TwitterAgent.sources.Twitter.consumerSecret = consumerSecret
TwitterAgent.sources.Twitter.accessToken = accessToken
TwitterAgent.sources.Twitter.accessTokenSecret =
 accessTokenSecret

TwitterAgent.sources.Twitter.keywords = Finance related
 keywords go here
TwitterAgent.sources.Twitter.keywords.created_at = Creation
 date goes here

TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:54310/path
 /to/your/hdfs/folder
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100
```

Once Flume has been configured, we run it with the following command using the bash console:

#### Listing 5.5: "Running Flume"

```
bin/flume-ng agent --conf ./conf/ -f conf/flume.conf -Dflume.root.logger=DEBUG,console -
n TwitterAgent
```

With this, Flume starts retrieving data from Twitter using the configuration set by the



user. Flume stores these data in HDFS, while it keeps retrieving data at the same time. These data are stored in a raw format, because Flume can retrieve data from many sources with different data formats.

The user can execute Hadoop commands to see that data are correctly stored in the configured HDFS path:

**Listing 5.6: "HDFS storing"**

```
hadoop fs -ls /user/david/data/input/finances/banks

Found 13 items
-rw-r--r-- 1 david supergroup 40180 2014-07-31 15:56 /user/david/data/input/
finances/banks/FlumeData.1406814972678
-rw-r--r-- 1 david supergroup 18331 2014-07-31 15:56 /user/david/data/input/
finances/banks/FlumeData.1406814972679
-rw-r--r-- 1 david supergroup 26213 2014-07-31 15:57 /user/david/data/input/
finances/banks/FlumeData.1406814972680
-rw-r--r-- 1 david supergroup 35862 2014-07-31 15:57 /user/david/data/input/
finances/banks/FlumeData.1406814972681
-rw-r--r-- 1 david supergroup 41479 2014-07-31 15:58 /user/david/data/input/
finances/banks/FlumeData.1406814972682
-rw-r--r-- 1 david supergroup 40859 2014-07-31 15:59 /user/david/data/input/
finances/banks/FlumeData.1406814972683
-rw-r--r-- 1 david supergroup 41481 2014-07-31 15:59 /user/david/data/input/
finances/banks/FlumeData.1406814972684
-rw-r--r-- 1 david supergroup 28105 2014-07-31 16:00 /user/david/data/input/
finances/banks/FlumeData.1406814972685
-rw-r--r-- 1 david supergroup 37879 2014-07-31 16:00 /user/david/data/input/
finances/banks/FlumeData.1406814972686
-rw-r--r-- 1 david supergroup 50743 2014-07-31 16:01 /user/david/data/input/
finances/banks/FlumeData.1406814972687
-rw-r--r-- 1 david supergroup 15831 2014-07-31 16:01 /user/david/data/input/
finances/banks/FlumeData.1406814972688
-rw-r--r-- 1 david supergroup 13233 2014-07-31 16:02 /user/david/data/input/
finances/banks/FlumeData.1406814972689
```

When the user thinks that there are enough data to be analyzed, Flume is stopped.

#### 5.4.2 Data processing and sentiment analysis using Pig

Now that we have enough data to analyze, we are going to see how to analyze them using a Pig script.

First, data should be loaded. Twitter data have JSON format, so a special data Loader is needed to do this task. As it was explained in section 2.10.2, we will use the JSON loader

provided by Elephant Bird project. To do so, we will register Elephant Bird's UDFs to be used inside the script and we will use them to load the tweets from HDFS.

Then, for each tweet, we get the content of the tweet (the text), and the language in which is written. With this information, we filter the tweets by their language, choosing English.

Finally, the user analyzes each tweet using an UDF that calls SEAS and perform sentiment analysis over a finance domain.

All this can be done automatically by using the following script:

Listing 5.7: "Example of Pig script for sentiment analysis"

```
-- Register the needed jars
REGISTER UDFs/seasudfs.jar;
REGISTER lib/json-simple-1.1.1.jar;
REGISTER lib/elephant-bird/pig/target/elephant-bird-pig-4.6-SNAPSHOT.jar;
REGISTER lib/elephant-bird/core/target/elephant-bird-core-4.6-SNAPSHOT.jar;
REGISTER lib/elephant-bird/hadoop-compat/target/elephant-bird-hadoop-compat-4.6-SNAPSHOT
.jar;

-- Load finance tweets from HDFS in json format
A = LOAD '/user/david/data/input/finances/banks/FlumeData.1406814972681' USING com.
 twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as (json:map[]);

-- For each tweet we use the text and the language
B = FOREACH A GENERATE json#'text' AS text, json#'lang' AS lang;

-- We keep those which are in english and limit to 10 tweets for testing.
C = FILTER B BY lang == 'en';
C = LIMIT C 10;

D = FOREACH C GENERATE text, es.upm.dit.gsi.udfs.EnglishSentimentAnalyzer(text) as
 polarity;
DUMP D;
```

When this script is executed in MapReduce mode, the following output is obtained, in which we can see that it only uses one minute to process all tweets and this is possible only using one node:

Listing 5.8: "Sentiment analysis output"

```
pig script.pig
.
.
.
```

```

2014-07-31 16:10:52,755 [main] INFO org.apache.pig.backend.hadoop.executionengine.
 mapReduceLayer.MapReduceLauncher - 0% complete
2014-07-31 16:10:56,265 [main] INFO org.apache.pig.backend.hadoop.executionengine.
 mapReduceLayer.MapReduceLauncher - 25% complete
2014-07-31 16:11:03,785 [main] INFO org.apache.pig.backend.hadoop.executionengine.
 mapReduceLayer.MapReduceLauncher - 33% complete
2014-07-31 16:11:04,788 [main] INFO org.apache.pig.backend.hadoop.executionengine.
 mapReduceLayer.MapReduceLauncher - 50% complete
.
.
2014-07-31 16:11:15,695 [main] INFO org.apache.pig.backend.hadoop.executionengine.
 mapReduceLayer.MapReduceLauncher - 75% complete
2014-07-31 16:11:23,213 [main] INFO org.apache.pig.backend.hadoop.executionengine.
 mapReduceLayer.MapReduceLauncher - 83% complete
2014-07-31 16:11:32,744 [main] INFO org.apache.pig.backend.hadoop.executionengine.
 mapReduceLayer.MapReduceLauncher - 100% complete
2014-07-31 16:11:32,791 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats -
 Script Statistics:

HadoopVersion PigVersion UserId StartedAt FinishedAt Features
1.2.1 0.12.0 david 2014-07-31 16:10:47 2014-07-31 16:11:32 FILTER,LIMIT

Success!

Job Stats (time in seconds):
JobId Maps Reduces MaxMapTime MinMapTime AvgMapTime MedianMapTime MaxReduceTime
 MinReduceTime AvgReduceTime MedianReducetime Alias Feature Outputs
job_201407301343_0007 1 1 1 1 1 1 8 8 8 8 A,B,C
job_201407301343_0008 1 1 1 1 1 1 12 12 12 12 D hdfs://localhost:54310/tmp/temp
 -1595047072/tmp314896637,

Input(s):
Successfully read 9 records (40582 bytes) from: "/user/david/data/input/finances/banks/
 FlumeData.1406814972678"

Output(s):
Successfully stored 5 records (625 bytes) in: "hdfs://localhost:54310/tmp/temp
 -1595047072/tmp314896637"

Counters:
Total records written : 5
Total bytes written : 625
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_201407301343_0007 -> job_201407301343_0008,
job_201407301343_0008

2014-07-31 16:11:32,803 [main] INFO org.apache.pig.backend.hadoop.executionengine.

```

```
mapReduceLayer.MapReduceLauncher - Success!
2014-07-31 16:11:32,806 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.
 schematuple] was not set... will not generate code.
2014-07-31 16:11:32,808 [main] INFO org.apache.hadoop.mapreduce.lib.input.
 FileInputFormat - Total input paths to process : 1
2014-07-31 16:11:32,808 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.
 MapRedUtil - Total input paths to process : 1

(tweet1,marl:Neutral)
(tweet2,marl:Negative)
(tweet3,marl:Positive)
(tweet4,marl:Negative)
(tweet5,marl:Negative)
```

With this, new tuples are generated containing the tweets and its sentiment polarity. In this case, all the tweets have been correctly classified with their corresponding sentiment polarity. This proves how useful is the validation process provided by the GATE user, which brings an accurate sentiment service to the Hadoop user.

These tuples can be stored in HDFS as well.

#### 5.4.2.1 Hadoop vs GATE

Let's make a comparison between the time that takes to process 100 tweets using SAGA and SEAS-Hadoop.

To do so, we are going to use the same corpus. In this case, the corpus will be composed by the same tweet which is repeated 100 times. We use the same tweet because we want to see the time that takes to analyze 100 tweets, not the quality of these analysis. The chosen tweet will be the following:

1. I missed the World Cup.

With this, we obtain the following execution times:

As we can see, Hadoop brings us a faster platform to perform text analysis because of its distributed processing nature.

Table 5.2: Execution times in GATE and Hadoop

| Context                | Time        |
|------------------------|-------------|
| GATE and SAGA          | 165 seconds |
| Hadoop and SEAS-Hadoop | 62 seconds  |

## 5.5 Conclusions

As we can see, SEAS brings us a reliable and accurate sentiment and emotion analysis service that can be used in multiple tools and platforms, because it is standardized and interoperable. Users can chose wherever they want to use SEAS with their text processing tools.

SAGA is a plugin for GATE full of possibilities. It can be used normally to perform sentiment or emotion analysis over corpus of documents using SEAS or other sentiment and emotion analysis services which are based in NIF. Also, and because of all the capabilities that GATE and its plugins offer us, it can be used to validate analysis services if the user has a correctly classified corpus.

Hadoop is a fast, reliable and highly configurable Big Data platform and there are a lot of projects that work over Hadoop. Flume can be used to obtain data from different web sources and it is so easy to set it to use Twitter as a source. Pig Latin is an easy and fast data processing language that allows user to perform any kind of data processing and to use their own functions by using UDFs.

In data processing, time is a very valuable resource, so it is important to use good analysis services. As we have seen in this case study, SEAS-Hadoop is faster than SAGA in the same tasks.



## Conclusions and future lines

---

*In this chapter we will describe the conclusions extracted from this master thesis, the achievements and future work.*





## 6.1 Conclusions

We have achieved all the goals proposed in this master thesis, as we have listed in section 6.2.

By using NIF, we have standardized sentiment and emotion analysis services, so they can be used in different analysis tools knowing how their input and output formats are going to be.

Also, the use of semantic ontologies such as Marl and Onyx have helped us to standardize sentiment and emotion resources, like the results provided by SEAS.

As a consequence, any SEAS client can be used with any other sentiment and emotion analysis service that uses NIF, Marl and Onyx, such as the ones provided in the Eurosentiment Portal.

With the use of GATE, we have extended SEAS capabilities to a text analysis software that has a lot of tools and plugins that complement the analysis provided by SEAS.

We have integrated SEAS with a Big Data platform as Hadoop. In this context, we have created an environment in which we can directly obtain data from web sources and analyze them.

Also, we have understood what distributed storing and processing in Big Data platforms is about.

## 6.2 Achieved goals

**Sentiment and emotion analysis services** We have been able to offer a service, SEAS, that provides several sentiment and emotion analysis services that the user will need. It consist in a REST API that allows users to perform these analyses using HTTP request. This is deeply described in section 4.2.

**Standardization and interoperability** provided by NIF. The way of accessing the resources provided by SEAS are standardized as it is defined by NIF's access layer, as defined in section 2.6. With this standardization we achieve one of the main goals of this master thesis, which is to provide a interoperable service so it can be easily combined with others text processing services and tools. This is detailed in chapter 4.2.1

**Standardized analysis results** by using semantic technologies. Sentiment results will contain all the information related with the text analyzed, its polarity and its value expressed using Marl Ontology. More information can be consulted in section 4.2.2. Emotion results will contain all the information related with the text analyzed, its categories and its values expressed using Onyx Ontology. More information can be consulted in section 4.2.3.

**Modular structures** achieved by the standards used. Each module described in this master thesis can be used in other systems that use NIF as a standard and Marl and Onyx ontologies to express sentiment and emotion resources.

**GATE integration** With SAGA we have achieved to integrate SEAS with GATE. SAGA provides a SEAS's client that can be used from GATE, so it can perform sentiment and emotion analysis by calling SEAS or other similar NIF's services. With this, SEAS services can be combined with other analysis tools. This is described in section 4.3.

**Service validation** Thanks to the capabilities provided by GATE, SAGA can be also used to validate how good an analysis service is. This is achieved with the GATE built-in function called Corpus Quality Assurance. To see detailed information see section 5.3.2.1.

**Big Data** We have achieved to create a Big Data infrastructure in which we cover all the steps in data processing: obtain, store, load, process and visualize data. To see detailed information see section 4.4

### 6.3 Future work

There are several lines than can be followed to continue and extend features of this work.

In the following points some fields of study or improvement are presented to continue the development.

- Dictionary based algorithms have an accuracy over 75%, but more complex algorithms could be used in order to improve accuracy.
- Machine learning algorithms could be added in order to have a different approach to sentiment and emotion analysis.

- More third-party services could be added by making NIF wrappers, so their inputs and outputs are adapted to NIF, Marl and Onyx.
- When making requests to SEAS, let the user choose other NIF parameters values that the ones that are currently allowed.
- Improve the processing resources inside SAGA. Offer a list of available NIF sentiment and emotion analysis services that are compatible with this processing resource.
- Allow the GATE user to make a bigger configuration over the HTTP requests that the processing resource does.
- Adding more nodes to Hadoop instead of having a single node configuration.
- Use Flume to obtain data for other sources and integrate all the data obtained in the same format.
- Developing more Pig scripts depending on the nature of the data and their use.
- Improving the UDFs to make them configurable during the execution.



## Installing and configuring SEAS

---

This tutorial goes through the process of installing and configuring SEAS to deploy it as a web service. SEAS's code is available at <https://github.com/gsi-upm/SEAS>

### A.1 Installation

#### A.1.1 Requirements

- Java 7
  - Download - <https://www.java.com/en/download/>
- Eclipse JEE
  - Download<sup>1</sup>
- Apache Tomcat 7
  - Download - <http://tomcat.apache.org/download-70.cgi>

---

<sup>1</sup><https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplersr2>

- Setup - <http://tomcat.apache.org/tomcat-7.0-doc/setup.html>
- MongoDB
  - Download - <http://www.mongodb.org/downloads>
  - Setup - <http://docs.mongodb.org/manual/>
  - JAR<sup>2</sup>

### A.1.2 Installation steps

- Setup Tomcat in Eclipse:
  - In Eclipse go to Preferences → Server → RunTime Enviroments → Add → Apache → Apache Tomcat 7.0 → Select path to Tomcat installation → Finish.
- Set up CretateMongoBD project
  - Import the project into Eclipse.
  - Add the downloaded mongo.jar to the project's library.
  - Go to <http://csea.php.ufl.edu/media/anewmessage.html> and make a request for Affective Norms for English Words so they provide you with the dictionaries, which are only abaliable for accademic purposes.
  - Configure the project with your MongoDB and with the path to the dictionaries.
  - Execute the project As a Java Application to parse ANEW into your MongoDB.
- Set up RestrictedToNIF project
  - Import the project into Eclipse.
  - Add Tomcat Libraries into the project.
  - Configure your MongoDB installation in the code.
  - Go to [http://www3.nd.edu/%7Emcdonald/Word\\_Lists.html](http://www3.nd.edu/%7Emcdonald/Word_Lists.html) to get these financial dictionaries, which are not abaliable for commercial use without authorization.
  - Put these dictionaries in GATE format. (See the dictionary format (<https://github.com/gsi-upm/SEAS/tree/master/SAGAtaNIF/src/resources/gazetteer/finances/spanish/paradigma>) used in SAGAtaNIF as an example of how to do it)

---

<sup>2</sup><http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-java-driver/>

- Run As Server.
- Set up SAGAToNIF project
  - Import the project into Eclipse.
  - Add Tomcat Libraries into the project.
  - Run As Server.
- Set up SEAS project
  - Import the project into Eclipse.
  - Add Tomcat Libraries into the project.
  - Run As Server.

This will deploy SEAS project in your local Tomcat Server, so you can use it as a web service.

## A.2 User manual

SEAS is a set of Sentiment and Emotion Analysis Services based on a REST API. To use it, the user has to make POST requests to the service. These requests should contain the following parameters:

**input** is the text that is going to be analyzed and it should be a plain text.

**informat** is the format of the input, which value should be *text*.

**intype** value should be *direct*, which means that the text is provided as plain text inside the request.

**outformat** value indicates the output format, which should be *JSON-LD*.

**algo** value is used to indicate the sentiment or emotion analysis algorithm to be used. The value can be: *spFinancial*, *emoticon*, *spFinancialEmoticon*, *enFinancial*, *enFinancialEmoticon*, *ANEW2010All*, *ANEW2010Men*, *ANEW2010Women*, *onyx*.

The requests have to be made to the following URLs:

- `http://localhost:8080/SAGAToNIF/Service`

- The user should call this URL to use the following algorithms: *spFinancial*, *emoticon*, *spFinancialEmoticon*
- `http://localhost:8080/RestrictedToNIF/RestrictedService`
- The user should call this URL to use the following algorithms: *enFinancial*, *enFinancialEmoticon*, *ANEW2010All*, *ANEW2010Men*, *ANEW2010Women*, *onyx*

### A.2.1 Command line interface

The command line interface can be used in a command shell window. On GNU/Linux, you can test the API using curl. A request would look like this:

```
curl --data "input=The text you want to analyze&intype=direct&informat=text&outformat=json-ld&algo=spFinancialEmoticon" http://localhost:8080/SAGaToNIF/Service
```

An example would look like this:

```
curl --data "input=I feel :)&intype=direct&informat=text&outformat=json-ld&algo=emoticon" http://localhost:8080/SAGaToNIF/Service

{
 "@context": "http://demos.gsi.dit.upm.es/eurosentiment/static/context.jsonld",
 "analysis": [
 {
 "@id": "http://www.gsi.dit.upm.es/ontologies/analysis#SAGA",
 "@type": [
 "marl:SentimentAnalysis"
],
 "marl:maxPolarityValue": 1.0,
 "marl:minPolarityValue": -1.0
 }
],
 "entries": [
 {
 "nif:isString": "I feel :)",
 "opinions": [
 {
 "@id": "_:Opinion1",
 "marl:hasPolarity": "marl:Positive",
 "marl:polarityValue": 1.0,
 "marl:describesObjectFeature": "Overall"
 }
],
 "strings": [
 {
 "nif:anchorOf": " :)",
 "nif:beginIndex": 7,
 "nif:endIndex": 8,
 "opinions": {
```



```

 "@id": "_:Opinion",
 "marl:hasPolarity": "marl:Positive",
 "marl:polarityValue": 1.0
 }
 }
]
}
}
}

```

### A.2.2 Using Java

On Java, you can test the API using `HttpClient`. A request would look like this:

```

HttpClient httpClient = HttpClient.createDefault();
HttpPost httppost = new HttpPost("http://demos.gsi.dit.upm.es/tomcat/SAGAtoNIF/Service")
 ;

ArrayList<BasicNameValuePair> params = new ArrayList<BasicNameValuePair>(4);
params.add(new BasicNameValuePair("input", "The text that you want to analyze"));
params.add(new BasicNameValuePair("intype", "direct"));
params.add(new BasicNameValuePair("informat", "text"));
params.add(new BasicNameValuePair("outformat", "json-ld"));
params.add(new BasicNameValuePair("algo", "spFinancialEmoticon"));

httppost.setEntity(new UrlEncodedFormEntity(params, "UTF-8"));

//Execute and get the response.
HttpResponse responseService = httpClient.execute(httppost);
HttpEntity entity = responseService.getEntity();

if (entity != null) {
 InputStream instream = entity.getContent();
 try {
 BufferedReader in = new BufferedReader(new InputStreamReader(instream));
 String inputLine;
 StringBuffer marl = new StringBuffer();

 while ((inputLine = in.readLine()) != null) {
 marl.append(inputLine);
 marl.append("\n");
 }
 in.close();
 String responseInString = marl.toString();
 // Use responseInString as you like
 } finally {
 instream.close();
 }
}
}

```

### A.2.3 Web service interface

The web service interface provides a web service interface to use and test the different services provided by SEAS. To use it, the user should go to `http://localhost:8080/SEAS/Controller`

The service looks as depicted in Figure A.1.

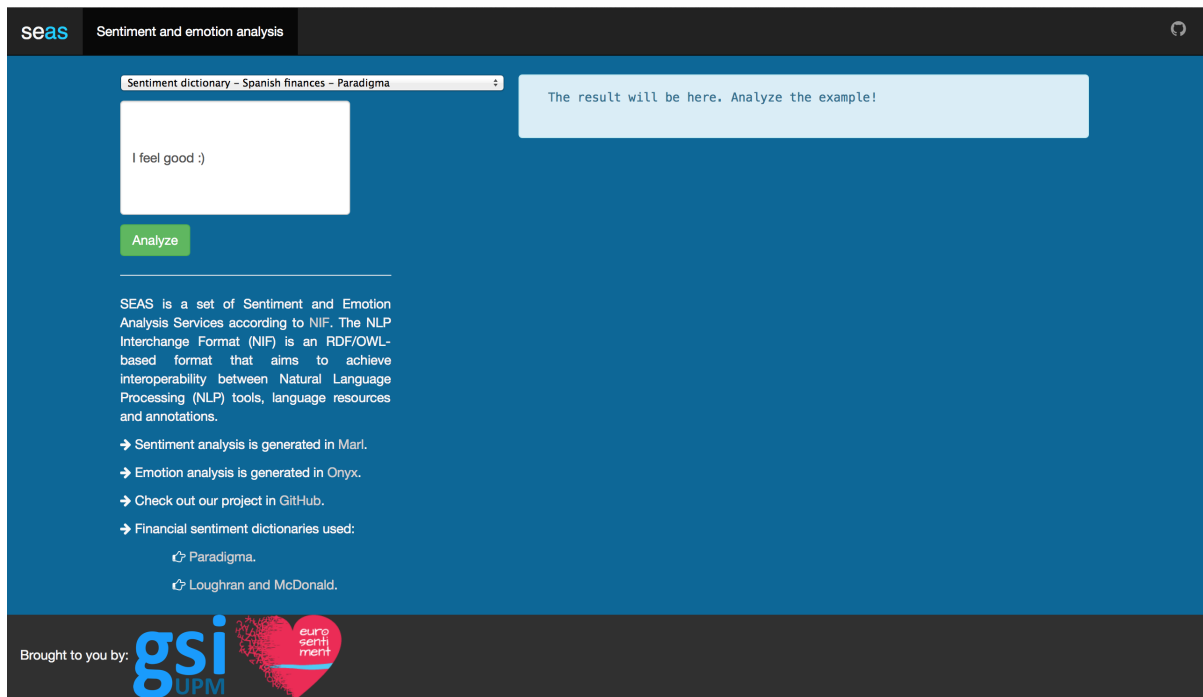


Figure A.1: Web service interface

The user only needs to select a service, write a text in the available textbox and click *Analyze*. The results are depicted in Figure A.2.

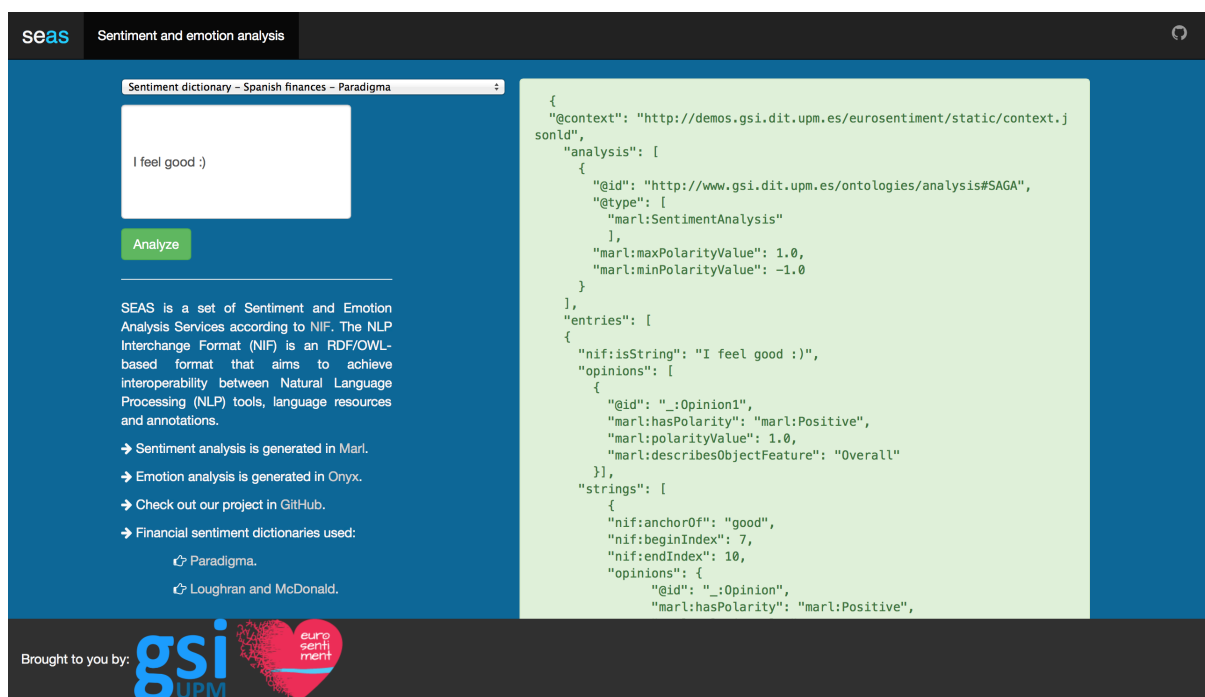


Figure A.2: Web service interface in action



## Installing and configuring SAGA

---

This tutorial goes through the process of installing and configuring SAGA to use it as a GATE plugin. SAGA's code is available at <https://github.com/gsi-upm/SAGA>

### B.1 Installation

#### B.1.1 Requirements

- Java 7
  - Download - <https://www.java.com/en/download/>
- GATE
  - Download - <https://gate.ac.uk/download/#latest>

#### B.1.2 Installation steps

There are two ways to install SAGA:

- Installation from GitHub repository
  - Download or clone the repository into your computer.
  - Unzip the folder called *saga* into the folder called *plugins* that is inside your GATE installation.
  - Open GATE. The new plugin should be available.
- Installation from GATE
  - open GATE → File → Manage CREOLE Plugins → Configuration tab → Click on the + symbol → add the repository name: GSI UPM url, `http://demos.gsi.dit.upm.es/SAGA/gate-update-site.xml` → Apply all → Available to install tab → Mark the SAGA plugin to install it → Apply all → Go to the Installed Plugins tab. There it is.

It is recommended to deploy SEAS's project as a local service in your computer to use this plugin.

## B.2 User manual

SAGA (Sentiment and Emotion Analysis integrated in GATE) is a set of processing and linguistic resources, written in Java, developed to run sentiment and emotion analysis over text using GATE platform.

If you are not familiar with GATE, check out these training modules<sup>1</sup> to understand what GATE can do.

Inside this plugin, the following processing resources are available.

### B.2.1 Sentiment and emotion analysis calling SEAS and Eurosentiment

To load this processing resource right click on Processing Resources → New → Sentiment and emotion analysis calling SEAS and Eurosentiment → Name it → OK. This process is depicted in Figure B.1.

Then, add this new PR to your current application or create a new one. To do so: right click on Applications → Create new application → Corpus Pipeline → Name it → OK.

The processing resource will have the following runtime parameters:

---

<sup>1</sup><https://gate.ac.uk/conferences/training-modules.html>

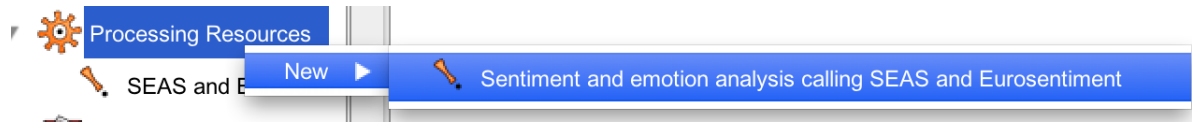


Figure B.1: New processing resource

**inputASName** is the Annotation Set that contains the annotation type to be analyzed.

**annotationType** is the annotation type to be analyzed.

**sentimentAnalysis** is a runtime parameter that sets if the PR is going to perform sentiment analysis with the chosen url or algorithm.

**emotionAnalysis** is a runtime parameter that sets if the PR is going to perform emotion analysis with the chosen url or algorithm.

**SentimentServiceURL** is the endpoint of the sentiment analysis service. If you deploy SEAS as a local service in your computer (Recommended): `http://localhost:8080/SAGaToNIF/Service` and `http://localhost:8080/RestrictedToNIF/RestrictedService`. You can use the demo available at GSI's website: `http://demos.gsi.dit.upm.es/tomcat/SAGaToNIF/Service` and `http://demos.gsi.dit.upm.es/tomcat/RestrictedToNIF/RestrictedService`. For more endpoints visit the Eurosentiment portal - `https://portal.eurosentiment.eu`

**EmotionServiceURL** is the endpoint of the emotion analysis service.

**APIKey** is the Eurosentiment token to use their services or other similar services that require an API KEY.

**ApiKeyName** is Eurosentiment (or other similar services) token name to use their services.

**sentimentAlgorithm** is the runtime parameter that sets the sentiment algorithm that the service is going to use. At the moment, you can use dictionary based algorithms.

**sentimentDictionary** is the runtime parameter that sets the sentiment dictionary that the service is going to use (in case that **sentimentAlgorithm** has been chosen). You can use the values *AUTO (Detects language)*, *Spanish finances Paradigma*, *English finances Loughran McDonald*, *Emoticon*, *Spanish finances and Emoticon*, *English finances and Emoticon*.

**emotionAlgorithm** is a runtime parameter that sets the emotion algorithm that the service is going to use. You can use *AUTO (Detects language)*, *onyx*, *ANEW2010All*, *ANEW2010Men*, *ANEW2010Women*.

**SentimentPolarityName** is the name of the sentiment polarity feature.

**SentimentValueName** is the name of the sentiment value feature.

**EmotionCategoryName** is the name of the emotion category feature.

**EmotionValueName** is the name of the emotion value feature.

### B.2.1.1 Example of use - Sentiment analysis over a finance domain

In this example we are going to see how to create a corpus inside General Architecture for Text Engineering (GATE), how to populate it and then we are going to set the corresponding runtime parameters of this processing resource to perform sentiment analysis over a finance domain.

- Create a new corpus and populate it: to do so, right click on Language resources → New → Gate Corpus → Name it → OK. Right click on the corpus → Populate → Go to the *saga* plugin folder → *resources* → *examples* → Choose *sentiment* → OK
- Set *emotionAnalysis* parameter to *false*
- Configure the runtime parameters as depicted in Figures B.2 and B.3(Be careful, the features inside the annotationType you choose to analyze will be substituted with the results of the analysis.).

|                                                                                                     |        |      |
|-----------------------------------------------------------------------------------------------------|--------|------|
|  annotationTypes | String | text |
|-----------------------------------------------------------------------------------------------------|--------|------|

Figure B.2: Finance example 1








|                                                                                                           |                     |                                                                      |
|-----------------------------------------------------------------------------------------------------------|---------------------|----------------------------------------------------------------------|
|  inputASName           | String              | Original markups                                                     |
|  sentimentAlgorithm    | SentimentAlgorithm  | Dictionary                                                           |
|  sentimentAnalysis     | Boolean             | true                                                                 |
|  sentimentDictionary   | SentimentDictionary | English_finances_Loughran_McDonald                                   |
|  sentimentPolarityName | String              | marl:hasPolarity                                                     |
|  sentimentServiceURL   | String              | http://demos.gsi.dit.upm.es/tomcat/RestrictedToNIF/RestrictedService |
|  sentimentValueName    | String              | marl:polarityValue                                                   |

Figure B.3: Finance example 2

- Run this application.
- Check the results depicted in Figure B.4.



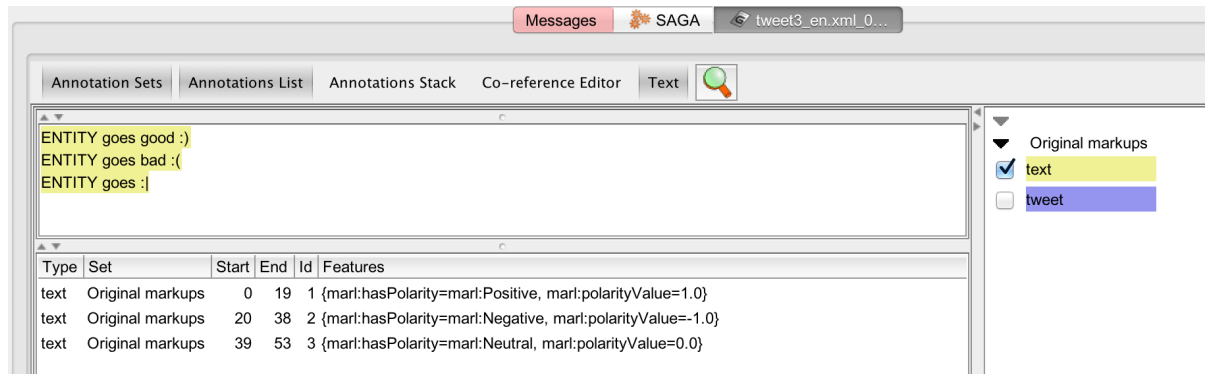


Figure B.4: Finance example result

### B.2.1.2 Example of use - Emotion analysis using Onyxemote

In this example we are going to see how to create a corpus inside GATE, how to populate it and then we are going to set the corresponding runtime parameters of this processing resource to perform emotion analysis calling Onyxemote service.

- Create a new corpus and populate it: to do so, right click on Language resource → New → Gate Corpus → Name it → OK. Right click on the corpus → Populate → Go to the *saga* plugin folder → *resources* → *examples* → Choose *emotion* → OK
- Set *sentimentAnalysis* parameter to *false*
- Configure the runtime parameters as depicted in Figure B.5 (Be careful, the features inside the annotationType you choose to analyze will be substituted with the results of the analysis.).

| Name                | Type             | Required | Value                    |
|---------------------|------------------|----------|--------------------------|
| annotationTypes     | String           |          | text                     |
| emotionAlgorithm    | EmotionAlgorithm |          | onyx                     |
| emotionAnalysis     | Boolean          |          | true                     |
| emotionCategoryName | String           |          | onyx:hasEmotionCategory  |
| emotionServiceURL   | String           |          |                          |
| emotionValueName    | String           |          | onyx:hasEmotionIntensity |
| euroSentimentToken  | String           |          |                          |
| inputASname         | String           |          | Original markups         |

Figure B.5: Emotion example

- Run this application.
- Check the results as depicted in Figure B.6.

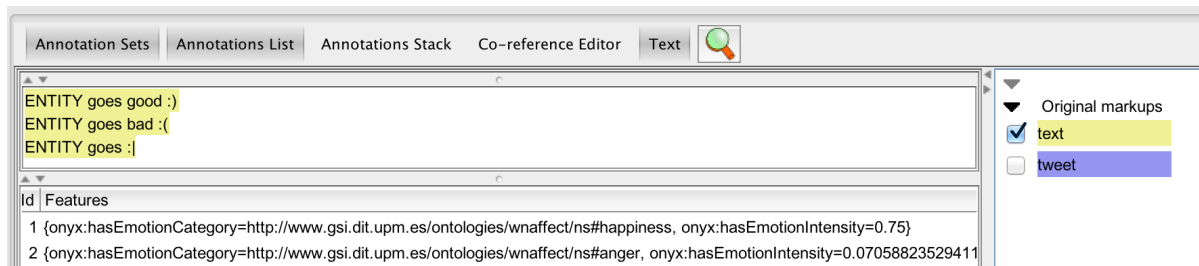


Figure B.6: Emotion example results

### B.2.1.3 Example of use - Eurosentiment services

In this example we are going to see how to create a corpus inside GATE, how to populate it and then we are going to set the corresponding runtime parameters of this processing resource to perform sentiment analysis calling one of the services available at Eurosentiment Portal.

- Sign up in the Eurosentiment portal, <https://portal.eurosentiment.eu/accounts/signup/>. Register yourself as a *Service Developer*.
- You will receive an access token in your mail. Put it in the runtime parameter called *APIKey*.
- Set the runtime parameter called *ApiKeyName* as *x-eurosentiment-token*.
- Set the runtime parameters called *SentimentServiceURL* or *EmotionServiceURL* with the ones offered in the Eurosentiment portal, <https://portal.eurosentiment.eu/service/list#>, that perform sentiment or emotion analysis (Figure B.7).

The PR configuration should look as depicted in Figures B.8 and B.9.

- Load your corpus: to do so, right click on Language resource → New → Gate Corpus → Name it → OK. Right click on the corpus → Populate → Go to the *saga* plugin folder → *resources* → *example* → Choose *eurosentiment* → OK
- Set *sentimentAnalysis* parameter to *true*
- Run the application.
- Check the results as depicted in Figure B.10.

[Home](#) » [Active services](#)

These services are available to the users of the ESRP. Please refer to the detail in each individual services for specific conditions. Note that the ESRP works an aggregation platform and cannot be responsible for the SLA of the individual services. For information on how to publish a new service [here](#)

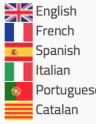












| Name                              | Description                                                     | Language                                                                                                                                            | Domain                                                                                                                                                                                       | Free                                 | Provider                                                                                                     |
|-----------------------------------|-----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|--------------------------------------------------------------------------------------------------------------|
| PT - Domain detection             | Domain detector for Hotels and Electronics domains              |  English<br>French<br>Spanish<br>Italian<br>Portuguese<br>Catalan |  Hotel<br> Electronics | <input checked="" type="radio"/> Yes | <br>Paradigma Tecnológico |
| ES - Domain detection             | Domain detection service developed by Expert System             |  English<br>German<br>Italian                                     |  Cars<br> Food         | <input checked="" type="radio"/> Yes | Expert System                                                                                                |
| ES - Language detection           | Language detection service developed by Expert System           |  English<br>German<br>Italian                                     |  Cars<br> Food         | <input checked="" type="radio"/> Yes | Expert System                                                                                                |
| ES - Sentiment & Emotion analysis | Sentiment & Emotion analysis service developed by Expert System |  English<br>German<br>Italian                                     |  Cars<br> Food         | <input checked="" type="radio"/> Yes | Expert System                                                                                                |

Figure B.7: Eurosentiment services










| Runtime Parameters for the "Sentiment and emotion analysis calling SEAS and Eurosentiment"              |                  |          |                              |
|---------------------------------------------------------------------------------------------------------|------------------|----------|------------------------------|
| Name                                                                                                    | Type             | Required | Value                        |
|  APIKey              | String           |          | XXXXXXXXXXXXXXXXXXXXXXXXXXXX |
|  annotationTypes     | String           |          | text                         |
|  apiKeyName          | String           |          | x-eurosentiment-token        |
|  emotionAlgorithm    | EmotionAlgorithm |          | auto                         |
|  emotionAnalysis     | Boolean          |          | false                        |
|  emotionCategoryName | String           |          | onyx:hasEmotionCategory      |
|  emotionServiceURL   | String           |          |                              |
|  emotionValueName    | String           |          | onyx:hasEmotionIntensity     |
|  inputASname         | String           |          | Original markups             |

Figure B.8: Runtime Eurosentiment parameters 1

### B.2.2 Predefined Sentiment Annotation

To load this processing resource right click on Processing Resources → New → Predefined Sentiment Annotation → Name it → OK. This process is depicted in Figure B.11.

|                       |                     |                                                                                  |
|-----------------------|---------------------|----------------------------------------------------------------------------------|
| sentimentAlgorithm    | SentimentAlgorithm  | Dictionary                                                                       |
| sentimentAnalysis     | Boolean             | true                                                                             |
| sentimentDictionary   | SentimentDictionary | English_finances_and_Emoticon                                                    |
| sentimentPolarityName | String              | marl:hasPolarity                                                                 |
| sentimentServiceURL   | String              | http://217.26.90.243:8080/EuroSentimentServices/services/server/access/sesse0328 |
| sentimentValueName    | String              | marl:polarityValue                                                               |

Figure B.9: Runtime Eurosentiment parameters 2

| Annotation Sets                |                  |       |     |    |                                                          | Annotations List | Annotations Stack | Co-reference Editor | Text |
|--------------------------------|------------------|-------|-----|----|----------------------------------------------------------|------------------|-------------------|---------------------|------|
| The iPad is a fantastic device |                  |       |     |    |                                                          |                  |                   |                     |      |
| Type                           | Set              | Start | End | Id | Features                                                 |                  |                   |                     |      |
| text                           | Original markups | 0     | 30  | 1  | {marl:hasPolarity=marl:Positive, marl:polarityValue=1.0} |                  |                   |                     |      |

Figure B.10: Eurosentiment results

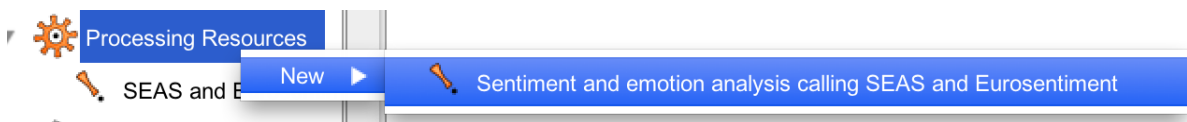


Figure B.11: New processing resource

Then, add this new PR to your current application or create a new one. To do so: right click on Applications → Create new application → Corpus Pipeline → Name it → OK.

The processing resource will have the following runtime parameters:

**Annotation Type** is the name of the annotation in which the sentiment polarity will be added.

**Input Annotation Set Name** is the name of the annotation set containing the Annotation Type.

**Sentiment Polarity Name** is the name of the annotation key.

**Sentiment Polarity** is the value of the annotation value.

For example, to annotate a corpus of negative documents with negative annotations, the parameter will be configured as depicted in Figure B.12.





| Name                                                                                                      | Type   | Required | Value            |
|-----------------------------------------------------------------------------------------------------------|--------|----------|------------------|
|  annotationType          | String |          | paragraph        |
|  inputASname           | String |          | Transferred      |
|  sentimentPolarity     | String |          | marl:Negative    |
|  sentimentPolarityName | String |          | marl:hasPolarity |

Figure B.12: Runtime parameters configuration for negative annotations



## Installing and configuring SEAS-Hadoop

---

This tutorial goes through the process of installing and configuring SEAS-Hadoop to integrate SEAS with the distributed processing platform called Hadoop. SEAS-Hadoop's code is available at <https://github.com/gsi-upm/SEAS/Hadoop>

### C.1 Installation

#### C.1.1 Requirements

- Java 7
  - Download - <https://www.java.com/en/download/>
- SEAS
  - Download - <https://github.com/gsi-upm/SEAS>
- Hadoop
  - Download - <http://hadoop.apache.org/#Download+Hadoop>

- Single Node Setup<sup>1</sup>
- Flume
  - Download - <http://flume.apache.org/download.html>
  - Setup - <http://flume.apache.org/FlumeUserGuide.html#setup>
- Pig
  - Download - <http://pig.apache.org/docs/r0.13.0/start.html#download>
  - Setup - <http://pig.apache.org/docs/r0.13.0/start.html#download>
- Elephant Bird
  - Download - <https://github.com/kevinweil/elephant-bird/>
  - Setup - <https://github.com/kevinweil/elephant-bird/>

### C.1.2 Installation steps

- Setup Flume to obtain data from Twitter:
  - To do so, go to the root of your Flume installation → go to *conf* folder → edit the file called *flume.conf* with the following:

Listing C.1: "Example of Flume configuration"

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.type = com.cloudera.flume.
 source.TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey = consumerKey
TwitterAgent.sources.Twitter.consumerSecret =
 consumerSecret
TwitterAgent.sources.Twitter.accessToken = accessToken
```

---

<sup>1</sup><http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html>



```
TwitterAgent.sources.Twitter.accessTokenSecret =
 accessTokenSecret

TwitterAgent.sources.Twitter.keywords = Finance related
 keywords go here
TwitterAgent.sources.Twitter.keywords.created_at =
 Creation date goes here

TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost
 :54310/path/to/your/hdfs/folder
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity =
 100
```

The following parameters are very important and must be set:

**Consumer key and consumer secret** , which are provided by Twitter when a user registers an application on its developers services.

**Keywords** to look for inside the tweets.

**Date** , when the tweets were created.

**HDFS path** to store fetched data.

With this, Flume is ready to obtain data from Twitter.

## C.2 User manual

SEAS-Hadoop allows us to obtain data from Twitter, store them in Hadoop distributed file system (HDFS), process them using Pig Latin scripts and then analyze it using the

sentiment and emotion analysis services provided by SEAS.

We are going to see how this can be done.

### C.2.1 Using Flume to obtain data from Twitter

Once Flume has been configured, we run it with the following command using the bash console:

#### Listing C.2: "Running Flume"

```
bin/flume-ng agent --conf ./conf/ -f conf/flume.conf -Dflume.root.logger=DEBUG,console -
n TwitterAgent
```

With this, Flume starts retrieving data from Twitter using the configuration set by us and stores these data in HDFS

If you want to see the obtained data, you can execute the following Hadoop command:

#### Listing C.3: "Hadoop ls command"

```
hadoop fs -ls /user/youruser/path/to/your/hdfs/folder
```

And the obtained data will be listed:

#### Listing C.4: "HDFS storing"

```
Found 3 items
-rw-r--r-- 1 youruser supergroup 40180 2014-07-31 15:56 /user/youruser/path/to/
your/hdfs/folder/FlumeData.1406814972678
-rw-r--r-- 1 youruser supergroup 18331 2014-07-31 15:56 /user/youruser/path/to/
your/hdfs/folder/FlumeData.1406814972679
-rw-r--r-- 1 youruser supergroup 26213 2014-07-31 15:57 /user/youruser/path/to/
your/hdfs/folder/FlumeData.1406814972680
```

### C.2.2 Sentiment and emotion analysis using Pig

When you think that Flume has obtained enough data, stop it.

As it was explained in section 2.10, Pig is an engine that allows the execution of data processing scripts over Hadoop. These scripts, also called data flows, are written in a data processing oriented language called Pig Latin.

Pig Latin scripts will be used to process the data that were retrieved and stored in HDFS by Flume. To do so, we will need to register Elephant Bird's User Defined Function (UDF)s to load JSON data and SEAS's UDFs to perform the sentiment or emotion analysis.

For example, the following script can be used to perform sentiment analysis over the obtained data from Twitter:

Listing C.5: "Example of Pig script for sentiment analysis"

```
-- Register the needed jars
REGISTER UDFs/seasudfs.jar;
REGISTER lib/json-simple-1.1.1.jar;
REGISTER lib/elephant-bird/pig/target/elephant-bird-pig-4.6-SNAPSHOT.jar;
REGISTER lib/elephant-bird/core/target/elephant-bird-core-4.6-SNAPSHOT.jar;
REGISTER lib/elephant-bird/hadoop-compat/target/elephant-bird-hadoop-compat-4.6-SNAPSHOT
.jar;

-- Load finance tweets from HDFS in json format
A = LOAD '/user/youruser/path/to/your/hdfs/folder/FlumeData.1406814972678' USING com.
 twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as (json:map[]);

-- For each tweet we use the text and the language
B = FOREACH A GENERATE json#'text' AS text, json#'lang' AS lang;

-- We keep those which are in english and limit to 10 tweets for testing.
C = FILTER B BY lang == 'en';
C = LIMIT C 10;

D = FOREACH C GENERATE text, es.upm.dit.gsi.udfs.EnglishSentimentAnalyzer(text) as
 polarity;
DUMP D;
```

To perform sentiment an emotion analysis, two UDFs are available:

**es.upm.dit.gsi.udfs.SentimentAnalyzer** will be called for each text that is going to be analyzed by the Pig Latin Script. This UDF will set the corrorresponding NIF parameters, make a POST request to SEAS sentiment analysis service, receive the response in a JSON and parse it. The analyzed text will be returned with its sentiment polarity, that can be *positive*, *negative* or *neutral*, in a tuple in the format (*text*, *polarity*).

**es.upm.dit.gsi.udfs.EmotionAnalyzer** will be called for each text that is going to be

analyzed by the Pig Latin script. This UDF will set the corresponding NIF parameters, make a POST request to SEAS emotion analysis service, receive the response in a JSON and parse it. The analyzed text will be returned with its most representative emotion category, that can be *happiness*, *sadness*, *anger*..., in a tuple in the format *(text, category)*.

# Bibliography

---

- [1] A. Westerski, C. A. Iglesias, and F. Tapia, “Linked Opinions: Describing Sentiments on the Structured Web of Data,” in *Proceedings of the 4th International Workshop Social Data on the Web*, 2011.
- [2] J. F. Sánchez-Rada and C. A. Iglesias, “Onyx: Describing Emotions on the Web of Data,” in *Proceedings of the First International Workshop on Emotion and Sentiment in Social and Expressive Media: approaches and perspectives from AI (ESSEM 2013)*, vol. 1096, (Torino, Italy), pp. 71–82, AI\*IA, Italian Association for Artificial Intelligence, CEUR-WS, December 2013.
- [3] S. Hellmann, “Nif 2.0 core ontology,” tech. rep., AKSW, University Leipzig, 2013.
- [4] T. U. of Sheffield, “Gate embedded,” 2014.
- [5] D. Borthakur, “Hdfs architecture guide,” 2008.
- [6] M. T. Jones, “Distributed data processing with hadoop, part 2: Going further,” 2010.
- [7] T. A. S. Foundation, “Flume 1.5.0.1 user guide,” 2012.
- [8] E. Cambria, B. Schuller, Y. Xia, and C. Havasi, “New avenues in opinion mining and sentiment analysis,” *Intelligent Systems, IEEE*, vol. 28, pp. 15–21, March 2013.
- [9] W. Crawford and J. Kaplan, *J2EE Design Patterns*. O’Reilly Media, 1 ed., 10 2003.
- [10] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities,” *Scientific American*, p. 4, 2011.
- [11] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damjanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters, *Text Processing with GATE (Version 6)*. 2011.
- [12] B. Pang and L. Lee, “Opinion mining and sentiment analysis,” *Found. Trends Inf. Retr.*, vol. 2, pp. 1–135, Jan. 2008.
- [13] S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer, “Integrating nlp using linked data,” in *12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia*, 2013.
- [14] T. White, *Hadoop: The Definitive Guide*. O’Reilly Media, Inc., 1st ed., 2009.
- [15] S. Hoffman, *Apache Flume: Distributed Log Collection for Hadoop (What You Need to Know)*. Packt Publishing, 7 2013.

- [16] A. Gates, *Programming Pig*. O'Reilly Media, Inc., 1st ed., 2011.
- [17] T. Khare, *Apache Tomcat 7 Essentials*. Packt Publishing, 3 2012.
- [18] A. Klein, O. Altuntas, T. Hausser, and W. Kessler, "Extracting investor sentiment from weblog texts: A knowledge-based approach," in *Commerce and Enterprise Computing (CEC), 2011 IEEE 13th Conference on*, pp. 1–9, Sept 2011.