D5.10
WP5
Application development and functionality report 2

R - Report, PU - Public

The UniteEurope Consortium:

Participant no.	Participant organisation name	Short name	Country
1 (Coordinator)	INSET Research and Advisory	INSET	Austria
2	Erasmus University Rotterdam - Department of Public Administration	EUR	Netherlands
3	SYNYO GmbH	SYNYO	Austria
4	Imooty Lab	IMOOTY	Germany
5	Malmö University - Institute for Studies of Migration, Diversity and Welfare	MHU	Sweden
6	ZARA, Zivilcourage & Anti-Rassismus-Arbeit	ZARA	Austria
7	City of Rotterdam	CITYROT	Netherlands
8	City of Malmö	CITYMAL	Sweden
9	University of Potsdam, Department for Public Management	UP	Germany

UniteEurope is co-funded by the European Commission under the FP7 Programme

Document Information

Contract Number:	288308	
Lead Beneficiary:	IMOOTY	
Deliverable Name:	Application development and functionality report 2	
Deliverable Number:	5.10	
Dissemination Level:	PU	
Contractual Date of Delivery:	April 30, 2014	
Delivery Date:	April 30, 2014	
Authors:	Blaise Bourgeois	IMOOTY
	Johannes Neubarth	IMOOTY
Checked by:	Bernhard Jäger	SYNYO
	Dr. Bernhard Krieger	UP
	Dr. Sonja Kabicher-Fuchs	INSET

Table of contents

Figures

Tables

1 Summary

Recent years have witnessed the remarkable popularity of Web 2.0 concepts and social media, in which millions of people communicate and collaborate. Global participation platforms and social networks like Facebook or Twitter allow their users to discuss any political issue freely and interactively, thereby offering an enormous potential for improving governance and policy modelling. As a result, there is a big demand for tools which facilitate the usage of these vast resources of information.

UniteEurope aims at presenting a solution here, by building a multilingual social media analytics and decision support tool. We have applied traditional techniques in the field of social media analytics to the domain of integration policy, enabling us to collect content which was generated by citizens of all societal groups, and which is related to integration. Although the identification of relevant data was already very difficult in terms of technical feasibility, we managed to support content acquisition for nine different languages, making UniteEurope attractive for multiple European countries. In addition, we detect content that is relevant for certain European cities, which is crucial when we target municipal administrations and local NGOs.

The user will access our data in an intuitive web application. Its main features are filter options for finding information on specific integration issues, sending out alerts according to urgency criteria, and gaining statistical insights. We believe that an analytics tool with a strong focus on integration improves the communication between citizens and policy makers, enabling the latter to make better decisions.

In the following sections we give a final overview on the application development process of the project. Since UniteEurope aims at social media analytics, we list in section 2 the problems which usually need to be solved for the realisation of such a tool. Afterwards, we examine which of these requirements are applicable for our project, and where we expect additional problems. Also, the general software architecture is recapitulated.

The target group of section 3 are software and web engineers, who are interested in how our software was developed. The tools and techniques listed here are not specific to this project, but can also be interesting for any other projects where good practices of the IT world are going to be adopted.

Section 4 illustrates how content from the web is downloaded, analysed and persisted. In addition, we calculate and explain the latest statistics on the analysed data. Section 5 will then present the UniteEurope web application, which is responsible for giving our users access to the content[1].

In section 6, we interpret our development results and discuss how they are connected to the whole project. Section 7 concludes this deliverable.

[1] See also Deliverable 5.11, which focuses on visualisations in the user interface.

2 Introduction

Recent years have witnessed the remarkable popularity of Web 2.0 concepts and social media, in which millions of users communicate and collaborate. Driven by these new possibilities, the web has become more social and interconnected. Global participation platforms and social networks like Facebook or Twitter and a large number of local blogs and web communities offer an enormous potential for improving governance and policy modelling. As a result, there is a big demand for tools which facilitate the usage of these vast resources of information. UniteEurope aims at presenting a solution here, by building a scalable social media analytics and decision support tool. The tool will collect data which was generated by citizens of all societal groups, and which is related to integration issues. In the following, we will quickly review how existing tools solve the task of social media analytics, and then present the challenges which are specific for UniteEurope.

2.1 Social Media Analytics

Traditional social media analytics tools provide companies with an up-to-date overview of how their products, brands and competitors are perceived by the web community. Typical applications include:

- Trend scouting: Which products or features become increasingly popular on the market?
- Brand image analysis: How is a brand or product viewed by customers? How is this view influenced by the company, e.g. via promotional campaigns?
- Opinion leader detection: Which individuals or communities influence the market?
- Competitor monitoring: Which market moves made by competitors were (un-)successful? What have they planned in the future?

In order to answer these questions, an analytics tool must combine functionalities from several related fields of science, such as data mining, information retrieval or natural language processing, commonly subsumed under the term “web mining”. In his very concise book, [1] outlines the most common tasks in a web mining scenario:

T1. First, web content must be retrieved from the online sources which were defined in an earlier step. There are three different ways to do so:
    a. Given a set of URLs, a web crawler downloads the corresponding documents, which can be in any format (HTML, XML, PDF etc). In addition, the crawler might automatically detect links in the downloaded content, and follow them.
    b. Some documents are only accessible after entering data into a form. For example, a newspaper might allow users to browse through its archive of articles via search terms, but does not allow downloading the whole archive. In this case, form emulators allow for retrieving search results automatically and repeatedly. The downside is that for every source, a new emulator must be defined.
    c. Sources can also actively support the automated retrieval of content by offering APIs (Application Programming Interfaces). These allow for machine-

to-machine exchange of data via standardised web services. Again, for every API a matching client must be implemented.

T2. After downloading the content, it must be preprocessed, because the following analysis steps assume a unified input structure.
   a. All documents must be parsed, i.e. transformed into a tree-like representation which separates text from structure. While this step is straightforward for well-defined formats like JSON, other inputs like HTML are error-prone and must first be normalised.
   b. Afterwards, all information which is needed for the analytics task is extracted from the parse tree, such as the textual content or the publishing date.
   c. Based on the document text, duplicates and spam are removed if they would otherwise compromise usability or distort the analytical statistics.
   d. Finally, linguistic preprocessing takes place, such as stopword removal[2], tokenisation[3] or part-of-speech tagging[4].

T3. Starting from the unified input produced by previous steps, various analyses can be derived.
   a. A search index is created, which returns for a given query all the documents which contain words from the query. This simple lookup function can be made arbitrarily complex by supporting boolean operators (AND, OR, NOT) or searching for specific entity types (person and place names, email addresses etc).
   b. It is also useful to classify documents. One possibility is to use text genres, e.g. poetry vs. prose. Another would be sentiment analysis, deeming a text either positive, negative or neutral. Analysers are usually trained on texts which were classified by hand, yielding an algorithm which can automatically classify new documents later.
   c. If no training data is available, clustering algorithms can group similar documents into sets, based on word frequencies. While the clusters might not represent a meaningful class, they are useful for finding similar documents.
   d. On a more abstract level, statistical analyses can aggregate the analysed content. For example, the weekly number of published documents containing a certain product name represents a good trend indicator for this product.

T4. Due to the complex nature of web mining, it is recommended to define input-output test cases for the previous tasks, so that errors are detected early.

T5. Finally, the analysis results are displayed to the user. Depending on his objectives, results can be listed exhaustively, or aggregated by classes with their percentage values. Of course, this also influences how the data is visualised, e.g. as a search interface or a pie chart. Alternatively, the data can also be made accessible from an API, leaving it to the user how he wants to process it further.

[2] Stopwords are functional words that occur very frequently, but which do not convey information by themselves, for example “the”, “you”, “of”, “and” etc.

[3] Tokenisation splits a text into single words (tokens), based on spaces and punctuation.

[4] This step assigns a part-of-speech (verb, noun, adjective etc.) to every token.

2.2 UniteEurope

Based on the typical web mining tasks presented in the previous section, we will now investigate how they apply to UniteEurope.

Beginning with “T1: Content retrieval”, it is important to note that most of the online sources were already defined in Work Package 3. According to this list, we need to support URL-based web crawling (T1.a) for XML-based feeds and a small set of HTML-based feeds. Since the feeds are supposed to contain all relevant information, we do not follow links in the documents. Also, none of the sources require emulating forms (T1.b). Social media APIs, however, are strongly required (T1.c). Interestingly, we do not only use them for downloading content, but the Wordpress and Blogger.com APIs are used for finding new web feeds.

From task T2 (“Preprocessing”), we adopt the first and second step: Input documents are parsed and normalised, and relevant information is extracted. Furthermore, duplicates should be detected, so they can later be hidden from the user (T2.c). Spam removal seems needless, because we only store integration-related content in the first place. From the linguistic preprocessing (T2.d), we only keep the tokenisation. Stopwords must be kept in order to enable phrase search (“country **of** birth”), and part-of-speech tagging is currently not required by the following analysis steps.

We invest a lot of work into task T3, because the quality of analysis directly influences usability of the tool. Users must be offered a wide range of search functionalities (T3.a) in order to leverage the full potential of our collected data. Named entity recognition, however, is out of the scope of this project: Although it would be very helpful to distinguish, for example, between ZARA (the clothing company) and ZARA (the Vienna-based NGO), we would need large amounts of manually annotated training data, which goes beyond our time resources.

One of the biggest challenges are the classification problems (T3.b), of which we are facing four:

1. Language detection (classes: English, German, Swedish, …)
2. Geotagging (classes: Berlin, Vienna, Rotterdam, Unknown, …)
3. Integration relevance (classes: relevant, irrelevant)
4. Integration area (classes: Education, Labour, Anti-discrimination…)

Especially tasks 3. and 4. are very difficult. Traditional social media tools used for market research often succeed by simply searching for product names. For the term of integration, on the other hand, there is not even consensus in the scientific community how to define it, let alone how to isolate it from related topics like immigration. And the fact that UniteEurope is supposed to be **multilingual** makes it even harder, because classification models must be trained for each supported language.

Classification by sentiment will not be attempted, because at one project meeting we discussed that assessing posts as “positive” or “negative” would sound like a judgement and would deprive us of our neutral position. This is an interesting contrast to brand image monitoring tools, which heavily rely on finding out the customer’s sentiment.

Also, clustering of documents (T3.c) is not necessary, since the abovementioned classifications are quite comprehensive. Statistical analyses (T3.d), however, constitute an essential complement to the search functions, and will allow the user to trace integration issues over time, measure the impact of integration campaigns etc.

Multiple levels for testing the tool (T4) were established. On the technical side, all source code (not only for the analysers) is extensively tested via unit tests. In addition, it must be possible to track back and revise classification results. On the user side, we refer to the user tests in Deliverable 6.3.

UniteEurope offers two features which are essential for social media analysis, namely visualisation and access to the content. Tag clouds, pie charts and comparison by Source Type, Integration Area, country, language, keyword or several other Annotation Item of the UniteEurope Grid, make it possible to analyse and process the content[5]. All this information is also exportable to a csv file to follow the analysis in other offline context (Excel files…). To improve the quality of the tools and focus on the aim of the project, empirical components have been had to it. For example, the Library component is a network measure and case library that each user of UniteEurope is filling regarding his professional experience. This valuable library gives the opportunity to each user to get, follow and exchange results and experience with the UniteEurope "Community". Based on the same logic, the Connect component allows the access to the profile of the other institutions that are using UniteEurope tools.

2.3 Software architecture

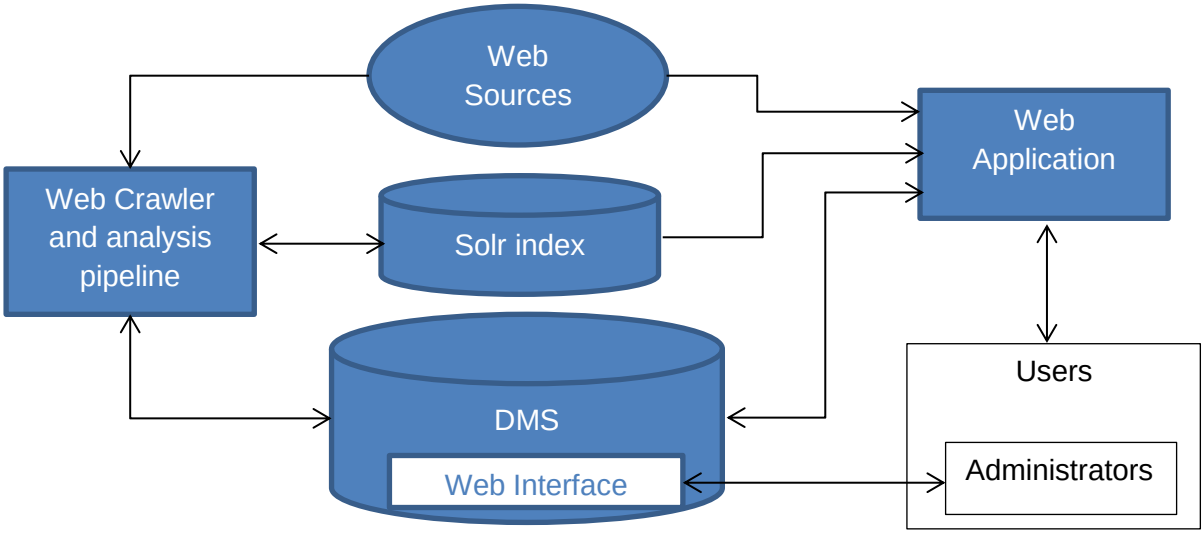

Figure 2.1: Software architecture

[5] Elaboration and description of each visualisation are done in D3.5, D3.10, D4.5, D4.9, D5.3 and D5.11

We will now quickly recapitulate the architecture of UniteEurope (Figure 2.1), because it is fundamental for the following chapters. From a data flow perspective, the Web Crawler is responsible for retrieving, preprocessing and analysing documents published by the Web Sources (T1 – T3.c). Analysis results are then stored in a search index (Solr, T3.a). The results are visualised by the UniteEurope Web Application and provided to the users (T5, T3.d). In addition, the Web Application allows the users to query some of the Web Sources directly (“Live Search”, see section 5.2).

In order to make the web mining process highly configurable, we added a database which stores all configuration data (DMS, Data Management System). It contains the list of Web Sources, keywords used for querying the APIs, user account data etc. Accordingly, both the Web Crawler and the Web Application have read/write access to it. Furthermore, the UniteEurope administrators can access the DMS via a separate web interface.

3 Software development process

In this chapter, we give a quick overview on the different stages in a software project, and present the tools which are used by Imooty during the implementation.

3.1 State of the art

Before the actual implementation starts, it is good practice to create a **model** for the problem domain. Such a model helps to identify entities in the domain and the relationships between them. The entities can then be implemented as **classes** in any object-oriented programming language. This step is usually carried out inside an **integrated development environment (IDE)**, containing at least a source code editor and tools for building and running programs.

Due to the complex nature of large software projects, the results are prone to containing errors. Various techniques have therefore been added to the development process which shall guarantee robustness and scalability. For example, a common error is unexpected input – the user or another program may pass parameters that cannot be processed, and the program crashes. Another example is regression errors. These are introduced by the developers themselves when they try to improve source code at one place and unintentionally cause failures at a different place. Such mistakes can be avoided by exhaustively testing the code with **unit tests** [2], feeding it with possible inputs and defining the expected behaviour.

Another challenge is how to coordinate multiple individuals working on the same project. Apart from traditional project management techniques, there exist also technical means which facilitate collaboration. One of them is the adherence to **coding conventions** [3]. These are normally represented by a set of rules which define if a line of code is well-formed according to some standard. Nonconforming code can automatically be detected and returned to its author. The rules can also specify a minimum amount of **documentation** that must be present for each class, method or variable. Altogether, such conventions increase readability of source code, making it easier to maintain and extend. Another important tool is a **version control system (VCS)** [4], which allows developers to work on different parts of a program simultaneously. Every time some work is finished, the changed parts can be merged back to the central version, so that the other developers can use it. VCS also allows to easily maintain several versions of a program (e.g. for different operating systems), while being able to commit bug fixes to all versions at once.

The next step in the development process is building and deploying the software. For large projects, **build automation** [5] tools are used for automatically compiling the code, packaging the binaries together with all required libraries, and deploying the package on the target system. Especially the automatic provision of software libraries can greatly reduce the amount of work, making it possible to reuse functionality implemented by third parties. Of course, license conditions must be respected.

After deployment, the end users of the tool can test its functions and report errors or missing features via a **bug tracking system**. The bug tracker alerts the developers who start fixing the bugs and thereby start a new release cycle.

Error reports can also be created automatically, with the help of **monitoring software**. This includes hardware monitoring, e.g. server load or hard disk usage, as well as verifying that programs are still running and working flawlessly.

3.2 Methodology and Results

Models of the UniteEurope domain are written in the Unified Modeling Language (UML) [6]. An example for some of the most important entities is shown in Figure 3.1.

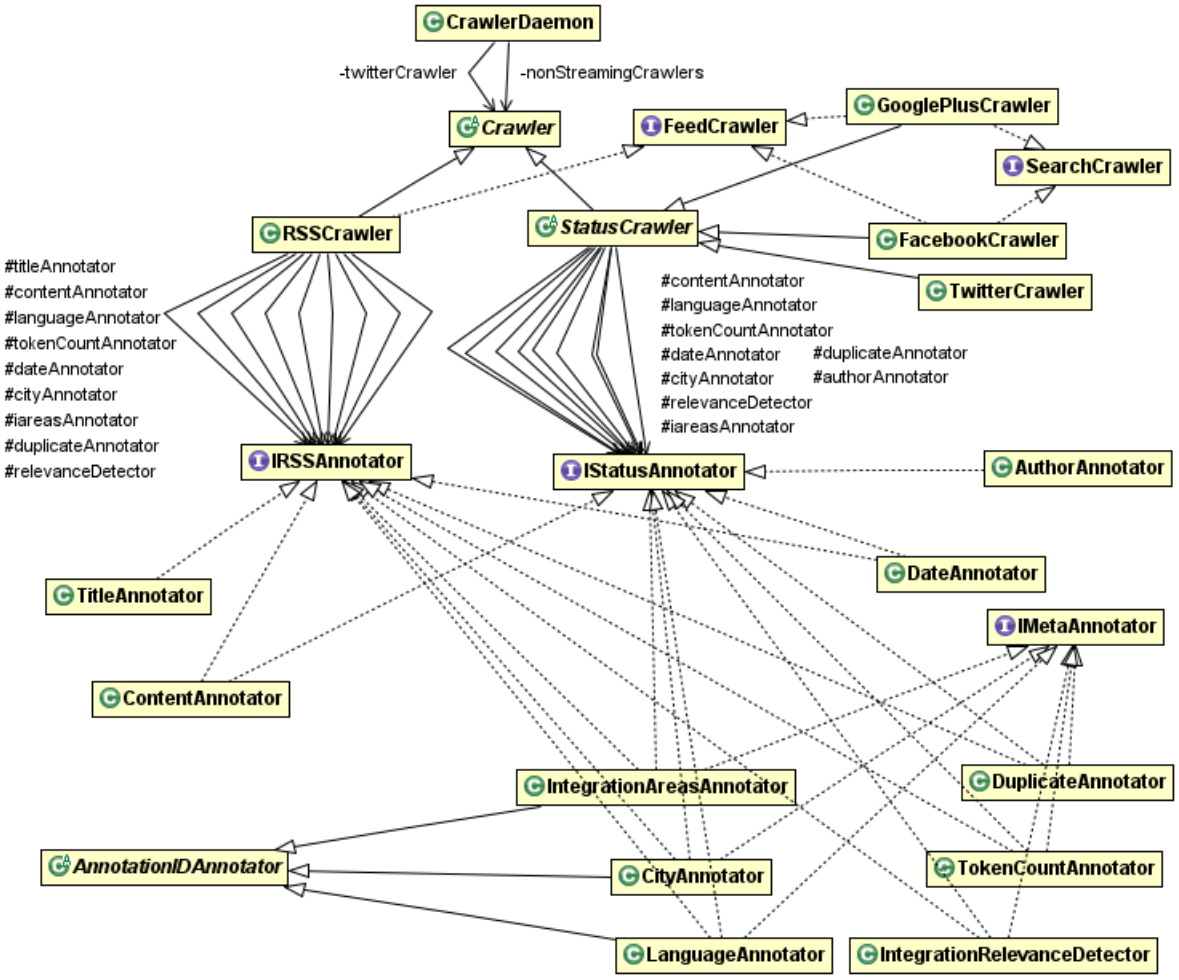

Figure 3.1: UML diagram

To give a rough idea of how to interpret this diagram, we pick two examples:

- A `StatusCrawler` is a web crawler which downloads and analyses status updates from social networks. It serves as the mother class for all our API clients, i.e. for the Twitter-, Google+ and Facebook-Crawler. Every `StatusCrawler` inherits functionality from the more abstract class `Crawler`, such as a function which reads feed information from the DMS. Every `StatusCrawler` has nine different annotators, for annotating the content of a status update, its language etc.

- A `LanguageAnnotator` detects and annotates the language of a post. It implements the `IRSSAnnotator` and `IStatusAnnotator` interfaces, because it is applicable both for RSS feed entries as well as status updates. The `LanguageAnnotator` is also a `MetaAnnotator`, meaning that it depends on the output of previous annotators (the language can only be determined when the title/content is known). Finally, it inherits functionality from the abstract class `AnnotationIDAnnotator`, because every language has a corresponding annotation ID in the DMS.

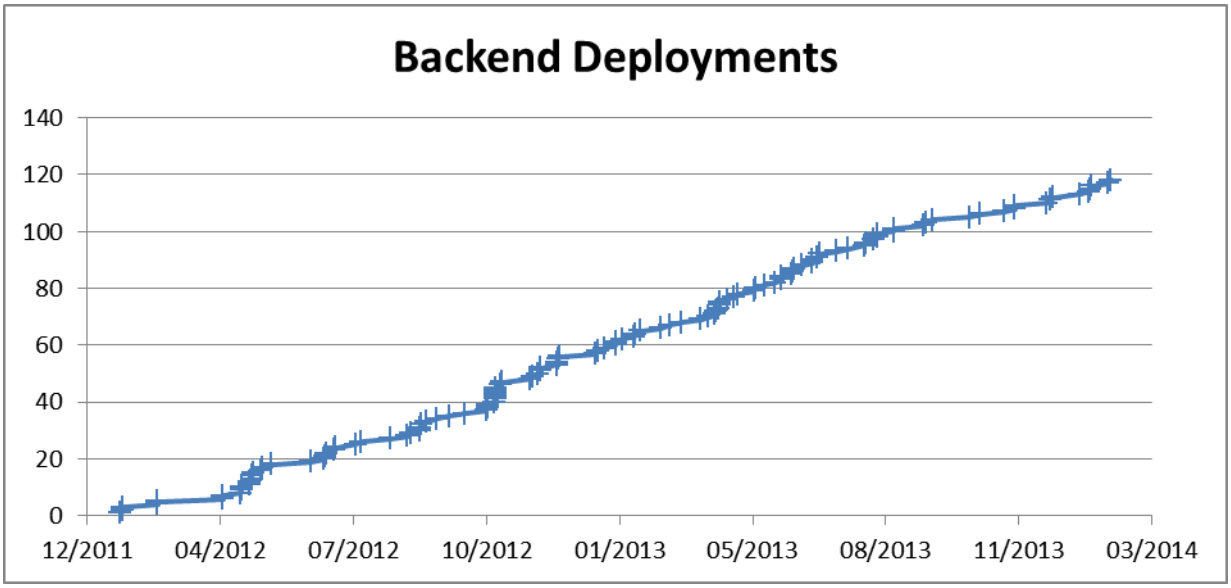

Figure 3.2: Backend Deployment Rate

The UniteEurope Web Application is developed in PHP using the Dreamweaver IDE [7]. For the Web Crawler, we use Java and Eclipse [8]. The Eclipse IDE makes it very easy to write and execute unit tests [9] for testing the code. Additionally, the Eclipse Checkstyle plugin [10] enables quick validation of coding conventions. We use the Imooty rule set containing about 50 rules, for example we disallow magic numbers [11] and we enforce code documentation via the Javadoc standard [12].

The version history of the crawler source code is managed by a Subversion [13] repository. This makes it possible to view and restore all versions since implementation started on 23.12.2011. Since this date, 118 versions have been deployed on the development server (see Figure 3.2), so we have about one deployment per week. The crawler is deployed as a `.jar` file containing all required libraries. These include for example a client implementation for the Twitter API [14] or for adding records to the Solr index [15]. The file is automatically built via Maven [16], and must be copied manually to the target server. On the server, several shell scripts are available for starting different Java daemons, like the crawler itself, or a process which re-annotates historical data, or a utility program which continuously analyses the geolocation of Twitter user profiles. All daemons are configured by a properties file called `uniteeurope.properties`, described in D6.2, section 8.

Builds that have not been tested yet are first deployed on the development server in order to find obvious bugs. After one or two weeks, debugged versions are made available on the production server for the users to start testing usability. If they find any problems, users report them in the bug tracker at http://ww2.uniteeurope.org/bugzilla3/. For example, one user requested that the (de-)selection of stream types in the Integration Monitor is kept over multiple search actions.

Both the development and production server are monitored via Nagios [17]. In addition, the DMS displays recent log file entries of the UniteEurope software components, and calculates statistics based on the number of downloaded posts.

4 Retrieving and processing large amounts of data

4.1 State of the art

As we have declared in section 2.2, the first step in our web mining task is to retrieve content from a given list of feeds and from social media APIs. Most of the **feeds** comply to the RSS [18] or Atom [19] format. Web feeds are usually accessed with a feed reader such as http://feedly.com. For the **APIs**, the picture is not so clear. Although every API relies on standard formats (XML, JSON), the way how these formats are applied is different every time. For example, while Google+ makes the author of a post available at a field called "actor", in Facebook the same field is named "from". Therefore, most web mining tools resort to implementing a new client for every single API [1]. Alternatively, companies such as Gnip [20] offer a unified API for all kinds of social media sources.

The text of the retrieved documents is then linguistically preprocessed. **Normalisation** and **tokenisation** of the text are straightforward – the former mostly transforms the text to lower case, while the latter determines word boundaries. Tokenisation would however be more difficult when we add languages like Chinese which do not delimit words. Additionally, it is often required to determine **word stems**. For example, when searching for texts containing "child", stemming allows the user to find documents containing "children". One approach for stemming is a simple lookup table, which maps all inflected forms to their corresponding stem. This method becomes infeasible for languages with a rich morphology, where rule-based stemmers [21] are more appropriate (but less accurate).

Another step which seems trivial at first is the removal of **duplicates**. Duplicates can occur when the same text is published at different locations, e.g. when press agency releases are copied by multiple newspapers. Documents with exactly the same text can simply be identified via calculation of a hash value, but otherwise we need to determine the *degree* of similarity and define a threshold. Usual approaches involve n-gram shingling [22] or even clustering algorithms [1, p. 106].

After the preprocessing is finished, input documents must be analysed and the results persisted, so that the users can fully exploit them. As mentioned in section 2.1, a **search index** is created, which "lists for each word in the collection all documents that contain it. […] An inverted index makes it easy to search for 'hits' of a query word" [23, p. 532]. The index can be made arbitrarily complex, from enabling phrase search and search operators (AND, OR, NOT) to searching derived words via stemming or even finding results in different languages.

Finally, the user might want to view only documents belonging to a certain group. For example, financial journalists are often more interested in stock market news than sports news. Various statistical algorithms have been proposed for the task of **document classification** [23] chapter 16. They all share the assumption that documents of the same class have similar properties, such as the words they are using. A common approach is to define a training set with maybe 1000 texts which are classified by hand. A classification

algorithm “learns” on these data, creating probabilistic models which describe how likely a word is to appear in documents of a certain class. The models can then be used to classify any other documents. Evaluation is performed by letting the algorithm learn on 90% of the training data, and measuring its classification accuracy on the remaining 10%.

In consideration of all the diverse preprocessing and analysis tasks, it seems natural that efforts have been made to unify and formalise this pipeline of tools. In particular, it would be desirable to define a standard format for inputs and outputs of any such tools, so that they can be re-used easily and combined with other components who conform to the standard. Examples for such **frameworks for the management of unstructured information** are UIMA [24] and OpenNLP [25].

Whichever approach we select for analysing our input documents, at some point we need to persist the results to the hard disk, because we cannot keep everything in memory. Traditionally, this is achieved via **databases**. A short overview of database technologies has been given in Deliverable 4.2, section 3. Briefly speaking, relational databases are appropriate for storing a limited amount of relations between a large number of entities, and when multiple relations need to be edited inside the same “transaction”. On the other hand, NoSQL databases are better suited for large data volumes which can be processed by several machines in parallel, but where transactions are not required[6].

4.2 Methodology

The UniteEurope tool currently supports sources of three different stream types. First, the **web feeds** defined in Work Package 3 are crawled and relevant information is extracted, such as title, content and publishing date. In addition, we crawl feeds that were automatically detected using the Wordpress and Blogger.com APIs for searching blogs. The whole procedure is described in D5.8, section 4.2.

Second, we collect status updates from **social media APIs**. Three different APIs are currently supported: Twitter, Facebook and Google+, described in D5.8, sections 4.3.1ff. We used existing third-party libraries for connecting to these APIs. The remaining implementation effort on our side mainly consists in creating one or more accounts for each API, and transforming the library-specific post structures into our own. The biggest disadvantage of this approach is that APIs change quite frequently. As soon as these changes become incompatible with our tool, we need to update the library and deploy a new version of the crawler. The only solution would be using an aggregation tool like the abovementioned Gnip, but this exceeds our current budget.

Apart from these structured stream types, we investigated the possibility of crawling sources which only offer content in HTML. Consider, for example, the Dutch news portal http://www.nujij.nl/. We first needed to determine the individual news departments, such as “Politics”, “Economy” or “Sport”. These are manually defined as HTML feeds in the DMS. Afterwards, a new **web scraper** [26] class was implemented. When this class is crawling http://www.nujij.nl/Politiek, it will first download the corresponding document and collect the

[6] This is a very abridged summary. We refer to the literature for a broader overview.

URLs of all recent politics articles. The path where to find the URLs in the HTML must be hard-coded in Java[7]. Every article is then downloaded and relevant fields are extracted: Title, content, publishing time, comments, publishing time of each comment etcetera. Again, the path to each field is hard-coded.

We developed web scrapers for two other sources: http://forums.marokko.nl, a Dutch web forum for Moroccan immigrants, and http://wijblijvenhier.nl/, a Dutch magazine focussing on Islam in the Netherlands. Web scraping is our only possibility to access these sources which do not offer standardised web feeds. It enables us to retrieve all comments written by visitors of the page. On the downside, the implementation takes a lot of time, and must be updated whenever structural changes occur on the target sites.

The first step in our analysis pipeline is the **detection of a document's language**. That is because most following steps are language-specific. Furthermore, all documents not written in one of the 9 UniteEurope languages[8] can be discarded at this point, which saves us unnecessary processing. As explained in D6.1, section 3.4.2., we use the lc4j [27] library as a basis. Languages are modelled as probability distributions over character n-grams (1- to 5-grams). For example, the word "cocoon" yields the following n-grams:

> *cocoo (1), ocoon (1), coco (1), ocoo (1), coon (1), coc (1), oco (1), coo (1), oon (1), co (2), oc (1), oo (1), on (1), c (2), o (3), n (1)*

Since the ~70 models shipped with lc4j were rather small, we had to increase their coverage. This was achieved by collecting a handful of Wikipedia articles for each respective language and training new classifiers on these data. In the future, we should measure classification accuracy of the language detector. We expect room for improvement especially for short texts.

Next in line are the traditional preprocessing steps. We use the Lucene [28] library for **normalisation, tokenisation and stemming**. The Lucene stemmers rely on rule sets for each supported language, which are basically rough morphological heuristics. Stemming errors are very frequent, e.g. "compete" and "competence" are both stemmed to "compet" (*overstemming*), while "racism" and "racist" come out of the stemmer unchanged, so that the common word stem "racis" is not detected (*understemming*). We did not investigate the influence of these errors. Furthermore, the Serbian, Croatian and Bosnian language are not supported at all[9]; only the non-stemmed texts can be used. It should be noted, however, that the stems are only used for enhancing the matching of keywords (see below). User queries in the web application are not stemmed. Instead, the users need to employ the Solr search operators, e.g. "immigrant*" will find documents containing "immigrant" and "immigrants".

Afterwards, we proceed with filtering documents which are not **relevant for the topic of integration**. These need no further analysis as they are immediately discarded. An integration-related text must have a minimum number of integration-related or -associated

[7] The HTML parsing was done with the jsoup library: http://jsoup.org/cookbook/extracting-data/selector-syntax

[8] Bosnian, Croatian, Dutch, English, German, Polish, Serbian, Swedish, Turkish

[9] We did not find any free Java stemmers for these languages. The analysers available from Lucene are listed here: http://wiki.apache.org/solr/LanguageAnalysis

keywords in it, depending on the size of the text (see D6.1, section 3.4.4). At this point, stems can be used for matching more keywords, e.g. the input word “immigrate”, which is not in the DMS, matches the keyword “immigration”. The relevance calculation also yields a relevance score, which can be used later for ranking documents (see Figure 5.17).

Note that we decided not to employ one of the sophisticated classification methods mentioned in the previous section. Instead, we follow a rule-based approach by defining the set of keywords and matching them in the input text (the matching is described in D5.9, section 5.2). The decision was necessary as it seemed impossible to create training data by manually classifying texts, due to the inherent vagueness of the integration topic and its related areas. The keywords thus serve as an abstraction level, reducing the question of “Is this text relevant for integration?” to the question “Is this keyword (mostly) used in integration contexts?”.

Next in line are two trivial annotation steps, which specify the **number of input words** as well as the **publishing date**. If no date is given, we use the time when the crawling happened.

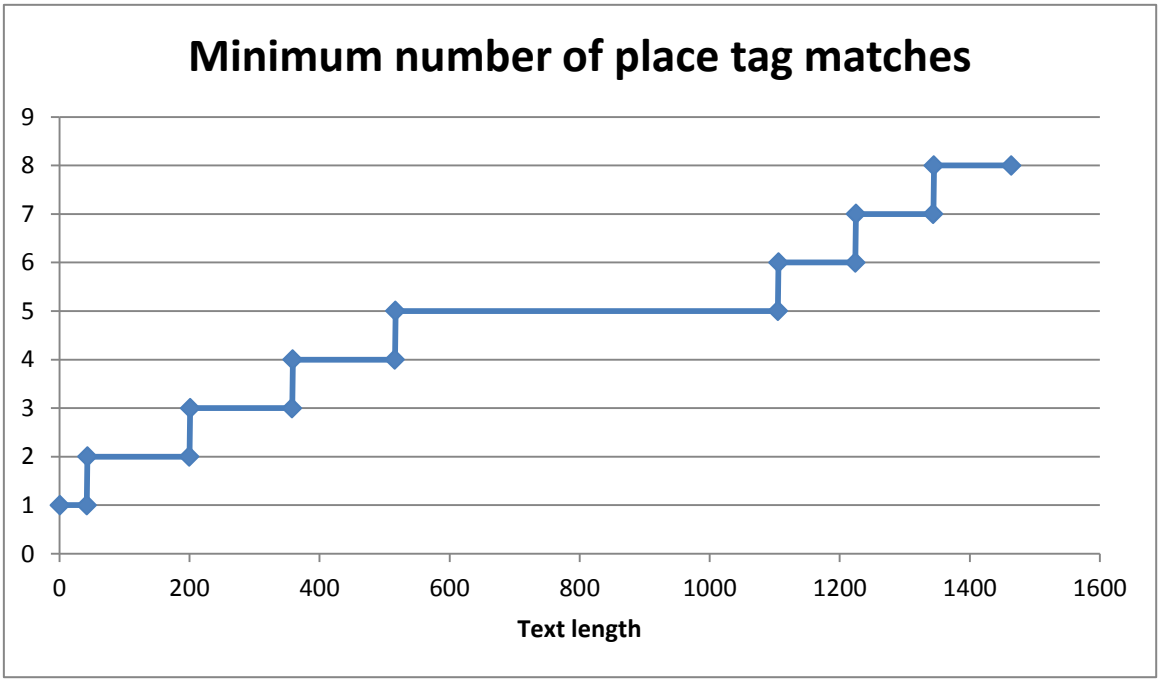

Figure 4.1: Geotagging with place tags

Afterwards, we try to enrich the input document with **geographical information**, as described in D6.1, section 3.4.3. There are two different annotations created, one of which is based on meta-information, the other is based on the textual input:

1. `cityAnnotation`: Use geographical tags added by the author of the post (this is only possible for content from social media APIs). If no tag is given, and the post is a tweet, try to localise the Twitter user’s profile data.

2. `cityAuto, districtAuto`: For each partner city[10], the DMS contains a large collection of place tags. For example, “Brandenburger Tor” maps to “Berlin”. We try to match place tags in the input document. If the number of matches exceeds certain thresholds (see Figure 4.1), the value of `cityAuto` is set to the corresponding city; otherwise it is set to “Unknown”. The thresholds were estimated by manual inspection of a small number of texts, a more thorough investigation might lead to higher annotation accuracy. In addition to cities, we also attempt to detect which city district a text is talking about.

After a document is geotagged, we apply another filter. It is based on the observation that for short texts (< 50 words) it is very hard to determine integration relevance based on keyword matching. In the beginning of the project, many irrelevant short posts were saved in our repository. Since we didn’t want to add an exception to the integration relevance calculation, we decided to constrain the data volume by **requiring short texts to have a geotag**; otherwise they are filtered out[11].

Next, we analyse the **integration areas** covered by a document. As explained in in D5.7, section 5.6, a document is tagged with all integration areas of all keywords that are matched in the text. Again, we can use stems for matching derived forms of a word.

Duplicate detection is the last analysis step in our pipeline. The task is quite intensive in terms of computing time and memory, so we want to make sure that all filters are applied beforehand. The duplicate detector itself is not a filter; instead, every original document (“primary”) is annotated with pointers to all its duplicates. Analogously, every duplicate receives a pointer to its primary. This allows us to hide duplicates when displaying search results to the user, but the user still has the possibility to see duplicates by request.

The duplicate detection algorithm was shown in D5.7, section 5.7. It estimates the similarity of two documents by measuring the overlap of the sets of bigram shingles. As we need to keep the whole content of all documents under comparison in memory, we restrict the detector to a time window of 14 days. Furthermore, only the first 500 words are considered. Alternative algorithms like clustering (see section 4.1) should also work, but the shingling approach needed less time to implement and yields good results (see next section).

The question remains if we had been better off using a **framework for unstructured information management**, like UIMA. At the moment, every analyser reads and writes annotations from/to a simple hash map. The sequence of analysis steps is hard-coded, and the dependencies between them are implicit. However, it was very fast to implement – according to our experience, UIMA entails a high administrative workload. Describing and updating the input and output behaviour of a UIMA component can take more time than is actually saved by reusing code. Furthermore, our hash map approach is so simple that it would be very easy to integrate more analysers or switch to a different exchange structure.

[10] Berlin, Vienna, Rotterdam, Malmö

[11] Combined with the thresholds in Figure 4.1, this means that texts with 42 words or less need at least one place tag match for being stored in our index, while texts between 43 and 49 words need two matches.

We conclude this section with an update on the **database structure**. In previous deliverables, we have argued why the HBase database, combined with a Solr search index, serves our needs better than a relational database (D4.2, section 3.3.1; D5.1, section 2). At the beginning of the project, however, we could not foresee that the data volume would be comparatively small, amounting to 4500 documents per day. At this speed, it would take almost 20 years for our first hard disk to be full. But the strength of HBase and the underlying Hadoop framework consists in its ability to distribute data and processing tasks over multiple hard disks and servers. Such a distributed system is not easy to maintain, and the configuration efforts are only worthwhile if the cluster will at some point contain more than one machine. In addition, the support for our wrapper library Lily (D4.2, section 3.3.3) via its mailing list[12] has severely declined, making it impossible to upgrade to the newest version.

We have therefore decided to use only **Solr** for the rest of the project duration. The Lily adapter is archived and the Hadoop processes are stopped, but both can be reactivated later if necessary. Until then, the Solr index delivers amazing performance, supporting a wide range of search operators (D5.4, section 5.3).

4.3 Results

From all the combined sources, we retrieve about 4500 integration-related posts per day. Figure 4.2 shows how the posts are distributed per source.

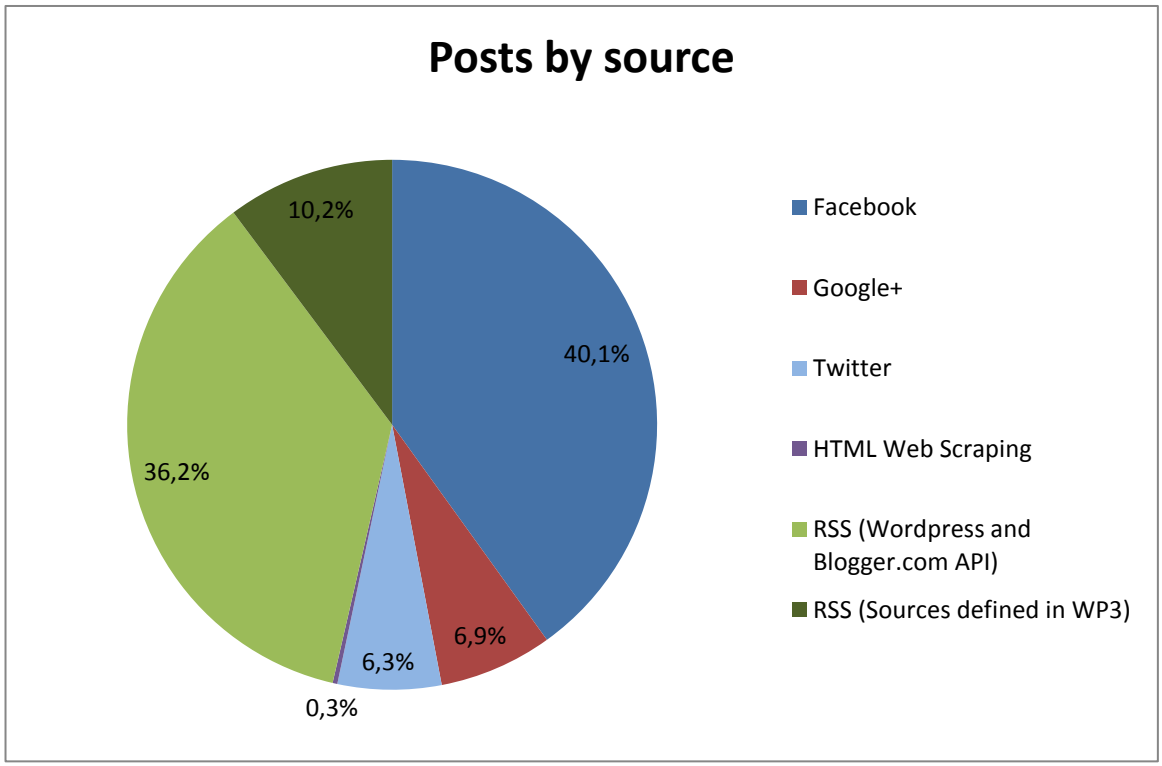

Figure 4.2: Posts by source

[12] https://groups.google.com/forum/#!forum/lily-discuss

The results indicate that both the RSS feeds defined in WP3, as well as the automatically collected feeds from the Wordpress and Blogger.com APIs, deliver important content for the tool.

As for the APIs, Facebook is clearly the most fruitful source, while Google+ and Twitter are dead even. We assume that from all three APIs, almost all relevant, public status updates are stored. This is because we retrieve posts by searching for keywords, see D5.8, section 4.4. The search interval for every keyword is optimised on how often a search yields new relevant content, see some examples below:

- "racism": Search in Facebook every 94 seconds, search in Google+ every 11 minutes
- "immigration": Facebook = 2.5 minutes, Google+ = 4.3 minutes
- "asylum seeker": Facebook = 1 hour, Google+ = 12 minutes

Now the smallest search interval we allow a keyword to have is 3 seconds, due to API restrictions on the number of queries. However, the smallest optimised search interval in our system is 19 seconds. Since this is well above our artificial limit, we claim that all available content is present in our index. Unfortunately, this does not hold for comments. While the Facebook API currently does not support searching for comments, Google+ has not been examined yet.

One of the biggest assets of UniteEurope is the support for multiple languages. Figure 4.3

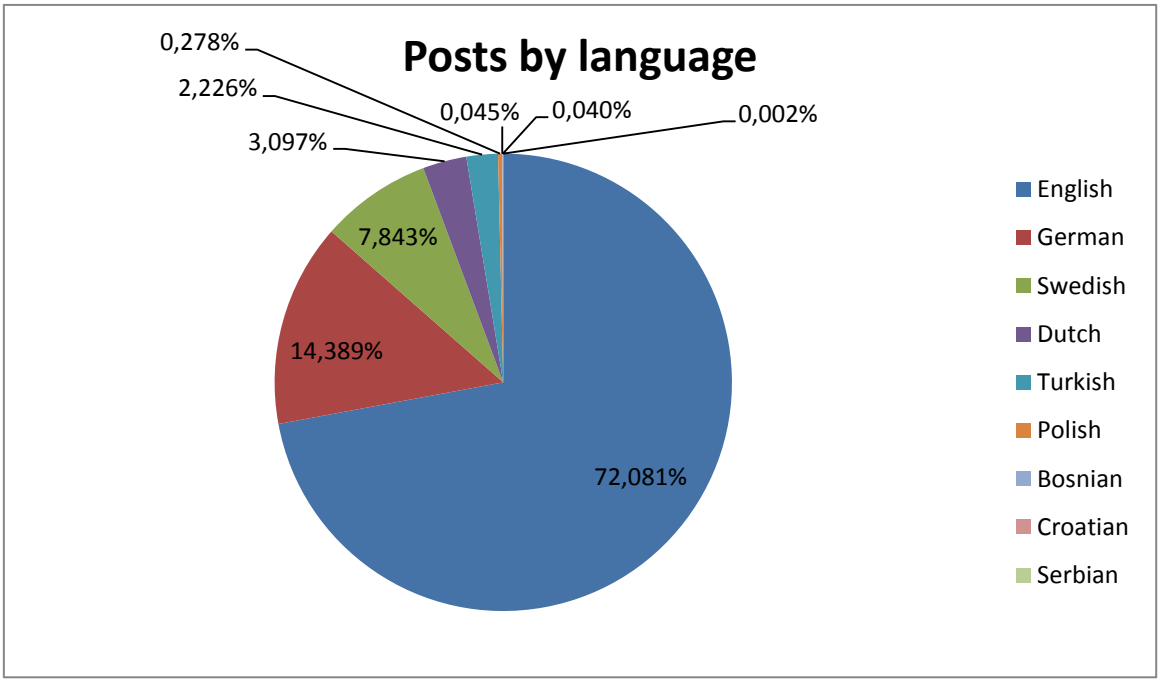

Figure 4.3: Posts by language

shows how posts are distributed over languages according to our detector. The high ratio of

English texts is due to the fact that we are storing a lot of content about integration issues in all parts of the world, and there are many English speakers living in all these parts. Alternatively, if we were to store only documents which can be geotagged with a location in Europe, we would lose a lot of content, because geotagging is not always possible: An author can talk about integration without naming any particular place. We decided to store everything for now, and leave it to our users if they want to see only posts which our tool located in Europe, or if they prefer filtering the global content by their own search queries.

Next, we inspect the geotagging results. Unfortunately, it is not possible to perform a deep evaluation, because of the following reasons:

- It would be interesting to know how many of the API status updates were tagged with a location by their authors. But we only use these tags if they point to one of the four partner cities, and ignore tags for all other cities like Paris, London etc., so the question remains unanswered.
- The same holds for place tagging: In order to evaluate the coverage of this approach, we would need place tags for all cities in the world.
- Neither is it possible to estimate the correctness of our place tagger. For doing this, we would need a testing corpus of posts which were manually tagged with a city, based on their textual content. However, such a corpus is not available.
- The filter which requires short posts to have a geotag artificially increases the percentage of geotagged posts in our index.

Under these conditions, it is clear that Figure 4.4 is only an indicator for the usability of the tool, and does not reveal anything about the correctness or coverage of the two geotagging approaches. The diagram shows the percentage for posts where both approaches yield no result (<unknown>), or where at least one of them assigns a city to the post. The number of posts where the two approaches disagree (<mixed results>) is very small.

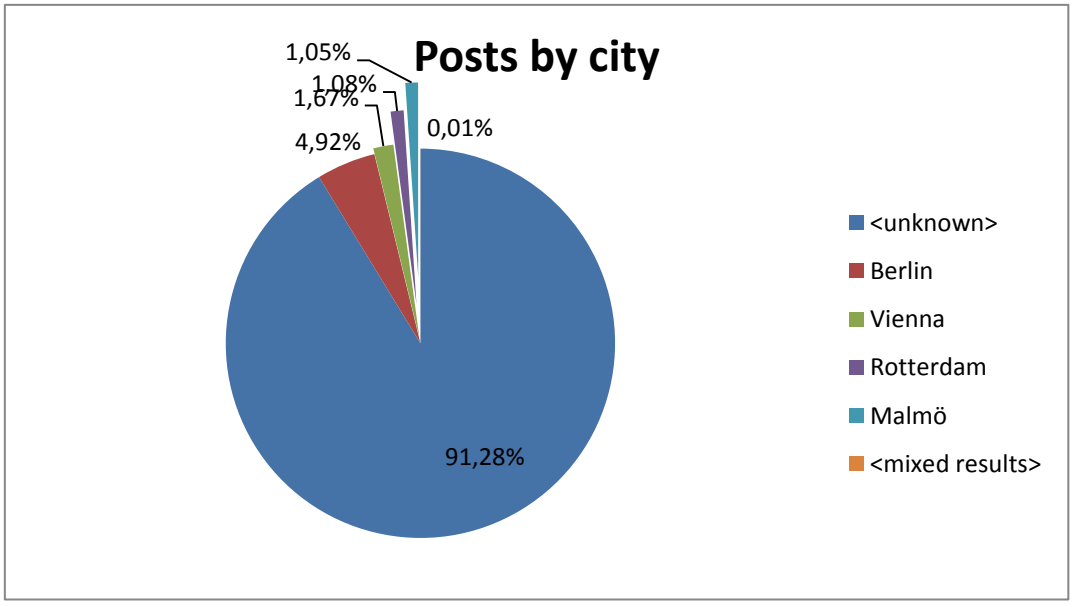

Figure 4.4: Posts by city

Analysing the integration areas covered by a post is quite simple. As stated in section 4.2, we select all areas of all keywords matched in the text. Of course, this results in extreme multi-classification, see Figure 4.5.

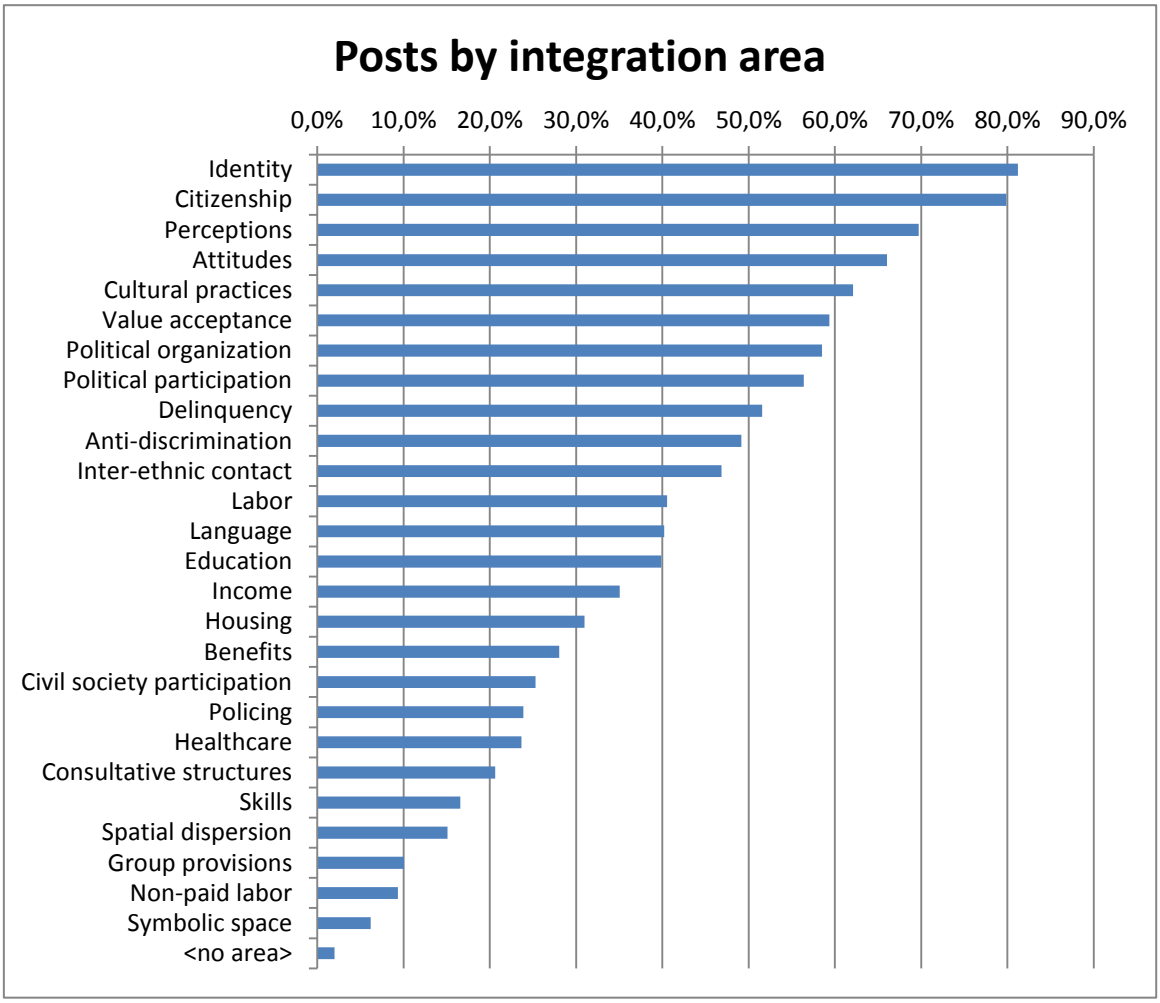

Figure 4.5: Posts by integration area

Finally, we have a closer look at the duplicate detector results. Until now, the detector seems to be very efficient; when working with the data, we have almost never come across examples where a duplicate was not detected, or where two different texts were identified as copies. What is surprising is that actually 40% of all posts in our index are duplicates. A quick review of the data shows that most of these are caused by crossposting: Many users of social networks decide to post the same text in multiple places. Targets are for example their own timeline (in multiple networks), the timeline of friends, groups and organisations, and traditional blogs.

5 Web Application

In this chapter we will present as in the State of the Art current solution before to present our methodology and introduce the result.

5.1 State of the art

PHP language, which has been selected at the beginning of the project by UniteEurope, is one of the most used and mature programming language dedicated to web application. It had immediately a big success explain because it was an interpreted language unlike C++ or Java, had the ability to be used directly with HTML, is based on Open Source technology and the powerful LAMP[13] server architecture and finally was built on a functional nature which is easier to learn than the Object Oriented Programming. Over the year, PHP become always more complex, vulnerable to security threats and less performance than other solution e.g. Ruby on Rail and Django. But it remains a very powerful language which affords a high quality framework that is contently improve and well documented (e.g. Zend, CakePHP…). UniteEurope choose Zend Framework to develop his tools.

In between a new solution appeared and seems to establish it slowly: Node.js. Since the popularisation of Smartphone and Tablet web developers thought their project as cross-platform solution and are always search after better performance. JavaScript framework is become a part of the Web application but this was just until the creation of Node.js, a client side language. Node.js defines itself as “a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices” [29] seems to be a very good alternative to PHP.

[13] **L**inux **A**pache **M**ysql **P**hp

The Node.js, as a server, is combining the JavaScript language on the client side and the server side. It is an asynchronous solution what makes it much faster as other solution and extremely fast due to its non-blocking I/O mechanism and the Google Chrome V8 engine

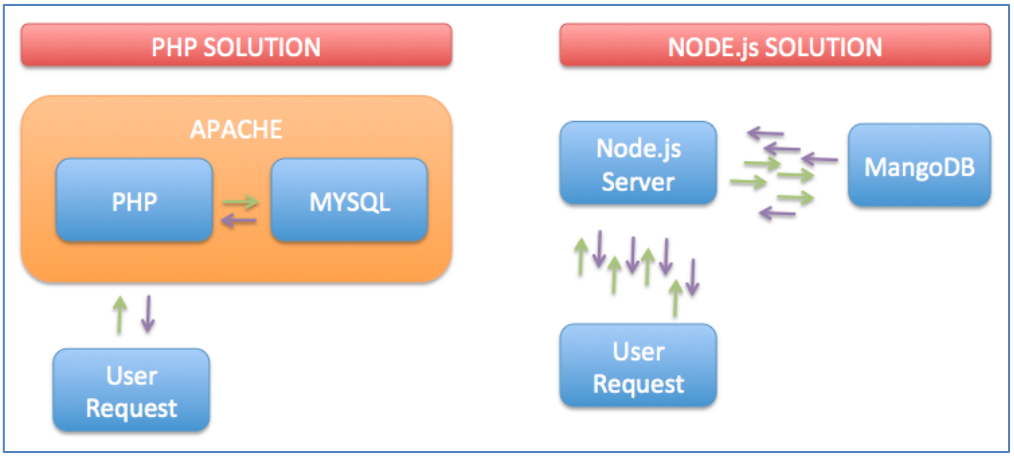

Figure 5.1: php vs node logic

technology. It is natural with NoSQL databases like MongoDB [30]. Both PHP and Node.js are functional languages with a relatively later addition of OOP to PHP. The biggest difference is that PHP's flow is blocked until the response of the remote server whereas node.js can handle multiple requests with just one thread of execution. Node.js is also a very modular tool what in one side make it very quick and easy but could be an inconvenient regarding the rigor of writing required and the project structure. In comparison, the clean and readable MVC pattern [31] used by the biggest PHP framework is probably a clear advantage to PHP.

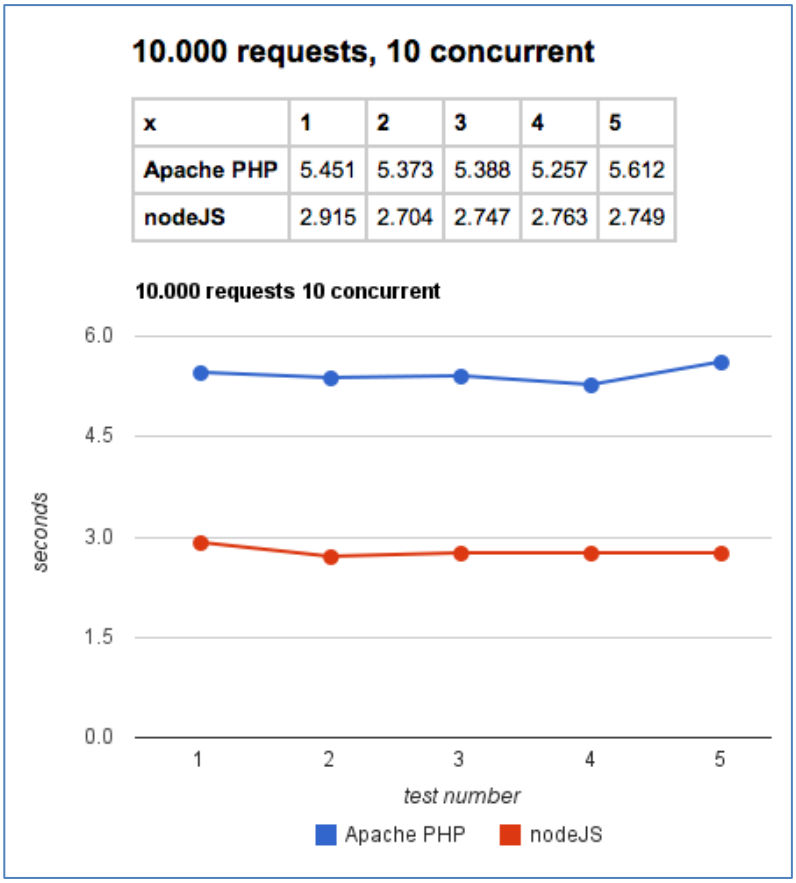

Figure 5.2: Benchmark Apache PHP / Node.js

The benchmark performance of both solutions [32], reproduced in Figure 5.2, is clearly in favour of node.js.

PHP was an outstanding technology in its day and node.js seems to establish it slowly. It's already the fastest growing platform by the pace of contributions. Regarding the UniteEurope project, PHP is still an adapted solution but node.js would have been probably a powerful alternative development solution that should have impacted the different tools functionalities.

5.2 Methodology

The development of UniteEurope tools was built following a clear methodology supported by the relation with the consortium. We first define the need with the scientist, conceptualize the architecture and begin the implementation of the different modules wished. Therefor the UniteEurope platform is the result of an efficient collaborative work.

The aim of the project was to structure, first, in a clear and efficient architecture a Grid Model pattern defined by the scientist and used to get and classify contents, and secondly to afford a convenient way to access the content crawled. We decided that we want to offer three websites, each one serving a different target group:

1. The project website introduces the project to persons who want to learn more about it, representing it to the outside world

2. The UniteEurope application makes all tools accessible to our users
3. The DMS lets administrators configure the tool and manage user rights

For each website, we employ the Zend Framework and the Pimcore CMS in order to unify the global application development.

The structure was duplicated in two entities: the development architecture and the production architecture. The both entities are fully self-ruling what involve that the grid model content and the content of the index are different in the both entities. The development process, described in Figure 5.3, is firstly done on the Development architecture. When the version is running fine, the update required (Script PHP, Database…) are uploaded on the production server.

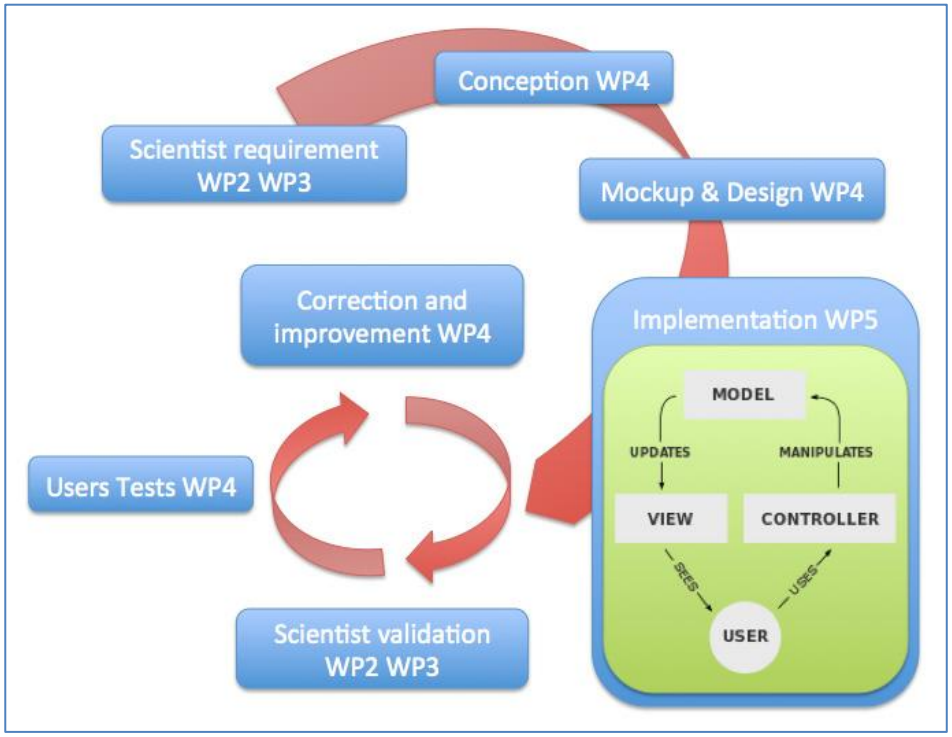

Figure 5.3: Work methodology applied

The methodology applied for the development of the tools is basically structure on the MVC Pattern[14] required and the collaboration with the consortium.

The improvement of the tools and correction are processed with Bugzilla [33], tools that allow end User and tester to collect bugs. The user tests conducted in WP6 also allowed us to correct and improve the tools until the final version.

[14] http://www.it-ebooks.info/read/2327/

5.3 Results

5.3.1 Web application architecture

As describe in previous deliverables[15], the web application architecture is built on the Data Management System available under http://dms.uniteeurope.org and the Application website tools itself. All is developed with Zend Framework 1.12.

Figure 5.4: Web Application Structure

The IT architecture is duplicated in two independent entities what allow to develop the project without endanger the Prototype usability.

[15] D4.2 - Framework architecture specification D5.1 - Tool architecture documentation

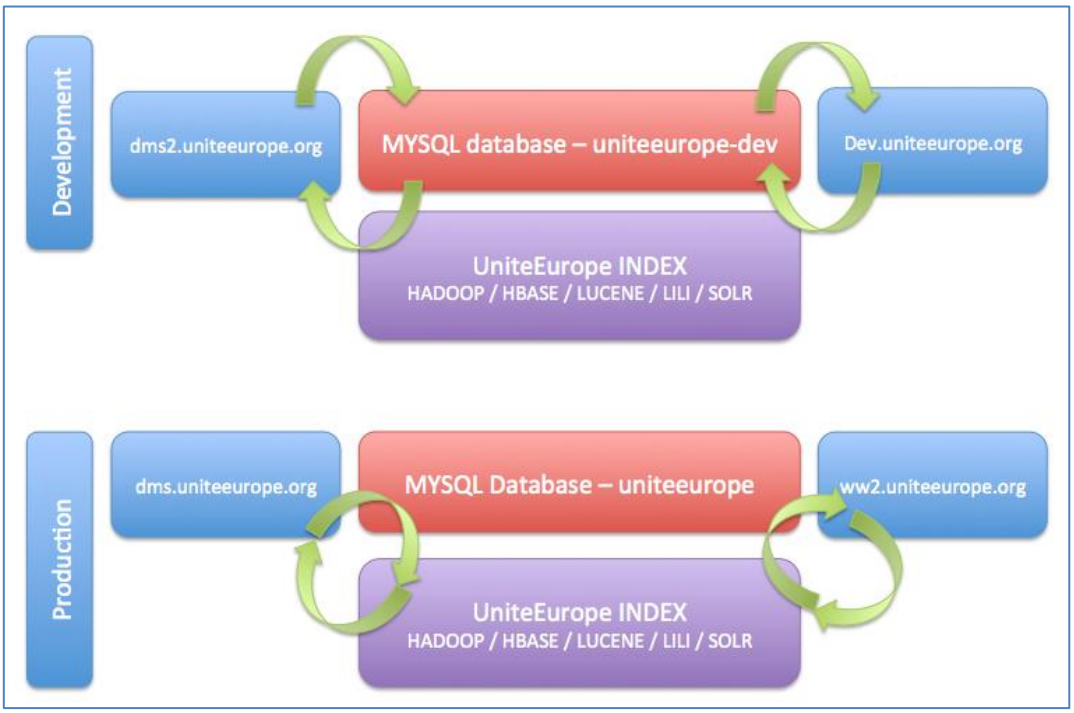

Figure 5.6: IT architecture overview

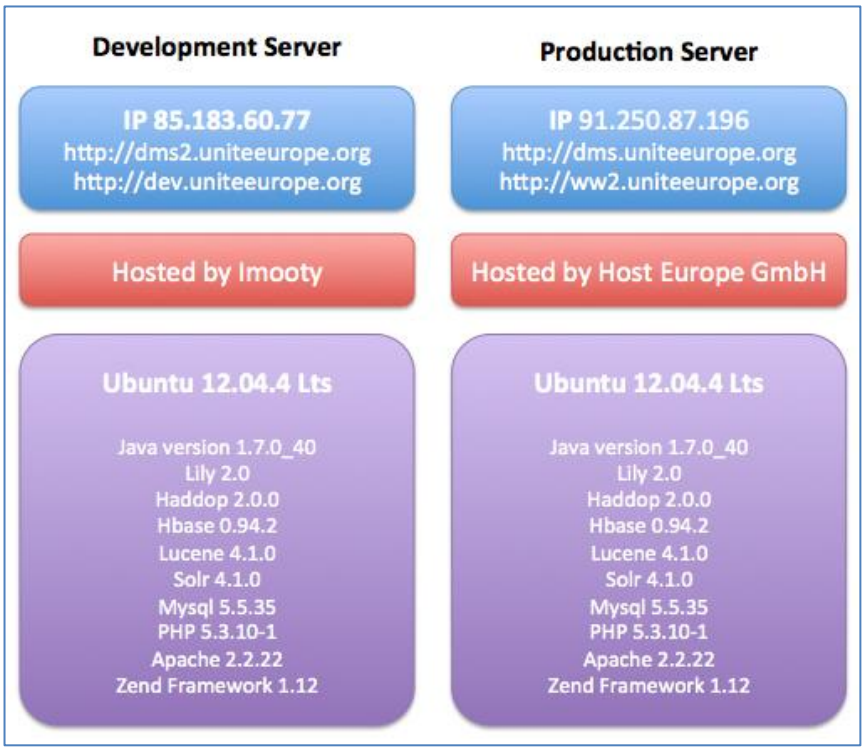

Figure 5.5: Server description

The architecture has been clearly described in several deliverables[16] during the project.

5.3.2 The Grid Model Pattern

The Grid Model Pattern, defined in a deep collaboration process with the researcher of the consortium, allows the structuration of the content and therefore the valorisation of the content.

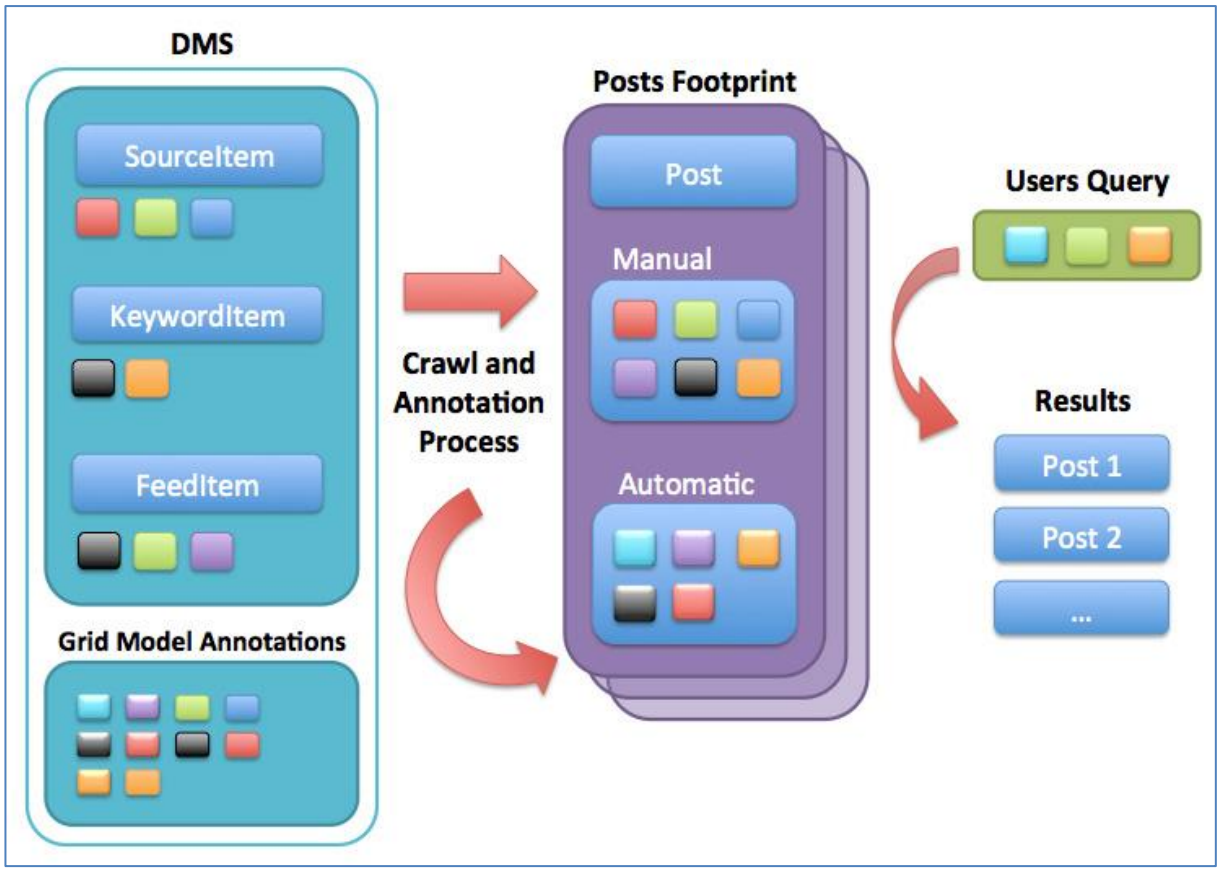

Figure 5.7: Grid Model logic

As described in Figure 5.7, the combination of Manual parameter, applied in the DMS, and the automatic annotation done by the Web Crawler, allow us to create a Footprint for each post saved. Every Meta Parameter that composed this Footprint allows us to interpret the content and match it following the requirement of the end users.

Figure 5.8 lists every annotation collected from the manual definition.

[16] D4.2 - Framework architecture specification D5.1 - Tool architecture documentation

Defined in the DMS		
Name	**Type**	**Example**
sourceID	Integer	5
feedID	Integer	6
streamType	String	Facebook, Rss, Twitter…
country	String	AT
cityAnnotation	String	VIE
cityFocus	String	Regional
languageAnnotation	List<String>	[DE,FR, …]
sourceType	String	Newspaper, Blog, Microblog
integrationFocus	Boolean	true, false
topics	List<String>	[Culture, Finance, …]
textType	String	Article, Comment

Figure 5.8: Annotations defined in the DMS

Generated by the web crawler		
Name	**Type**	**Example**
annotationIDs	List<Integer>	[1,5,6, …]
annotationIDsAuto	List<Integer>	[1,7,3, …]
date	Date	2012-01-05T08:41:17Z
title (optional)	Text	Immigration on the rise
content	Text	This article will give an extensive overview of…
tokenCount	Integer	514
cityAuto	String	ROT
districtAuto	String	ROT_Rotterdam Centrum
iareas	List<String>	[Identity, Citizenship…]
languageAuto	String	EN
keywordMatches	String (JSON)	{"keywordMatches": [{"id": 3502, "begin": 7, "end": 13, "stem": false, "affin": "related"}]}
lily.id	String	USER.http://derstandart\\at/index.html
authorID (optional)	String	KarinZauner
authorName (optional)	String	Karin Zauner
keywordType	String	Content
relevance1	Double	0.015
duplicateOf	String	primary, USER.http://derstandart\\at/index.html
duplicates	List<String>	[USER.http://derstandart\\at/index2.html]
duplicatesCount	Integer	1

Figure 5.9: Annotations generated by the web crawler

Figure 5.9 displays the list of every automatic annotation applied to each post. The automatics annotations are preferred to the manual annotation. For example, *LanguageAuto* is used to display in the web application the post per language. Or the duplicate annotation allows us to identify if the post is the first post recognize (Primary), which are the duplicate post and how many duplicate has been recognized.

To compare, in the *Comparative Analytics* component of the tools, posts from different language, a third level of the Grid model has been implemented. The aim of this level is to interconnect the *integration related* Keywords of the different languages.

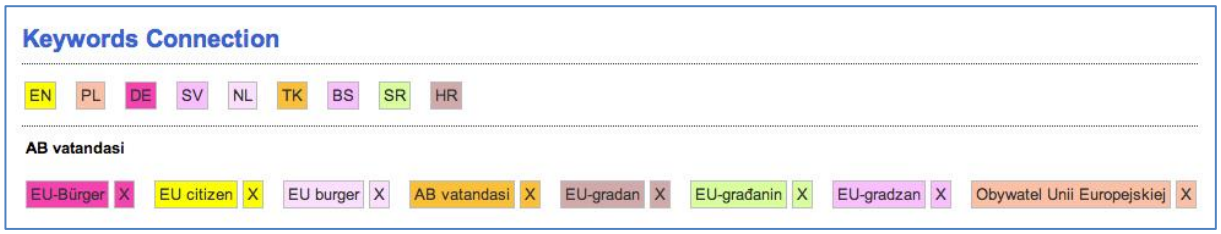

Figure 5.10: Keyword connections in the DMS

5.3.3 Web application functionalities

5.3.3.1 Data Management System (DMS)

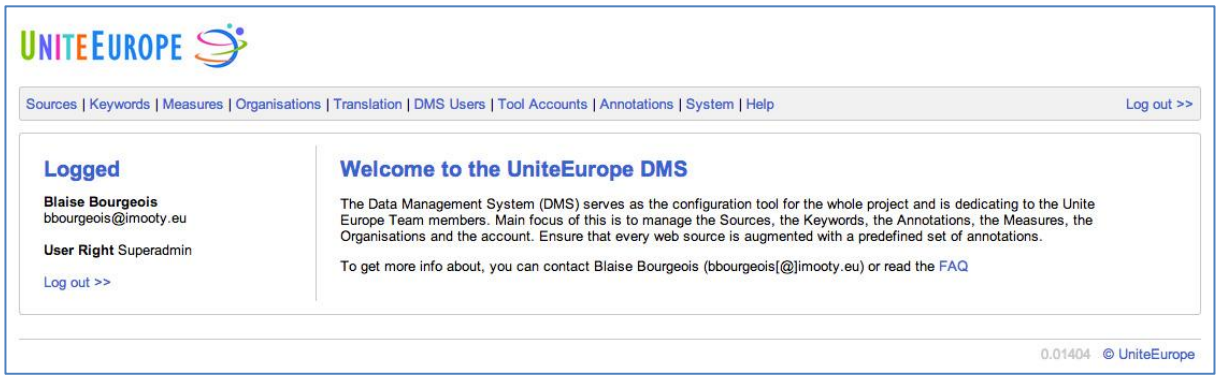

Figure 5.11: http://dms.uniteeurope.org

The Data Management System (DMS) serves as the configuration tool for the whole project, and is targeted at the UniteEurope team members. Its main focus is to manage the sources, the keywords, the annotations, the integration measures, the organisations and the user accounts. It also ensures that every web source is augmented with a predefined set of annotations.

The DMS has four complementary aims:

1. Manage the **Grid Model**
2. Afford **Monitoring** functionalities
3. Manage **Account**
4. Process extended **Crawl**

The DMS is fully structured with the MVC[17] logic.

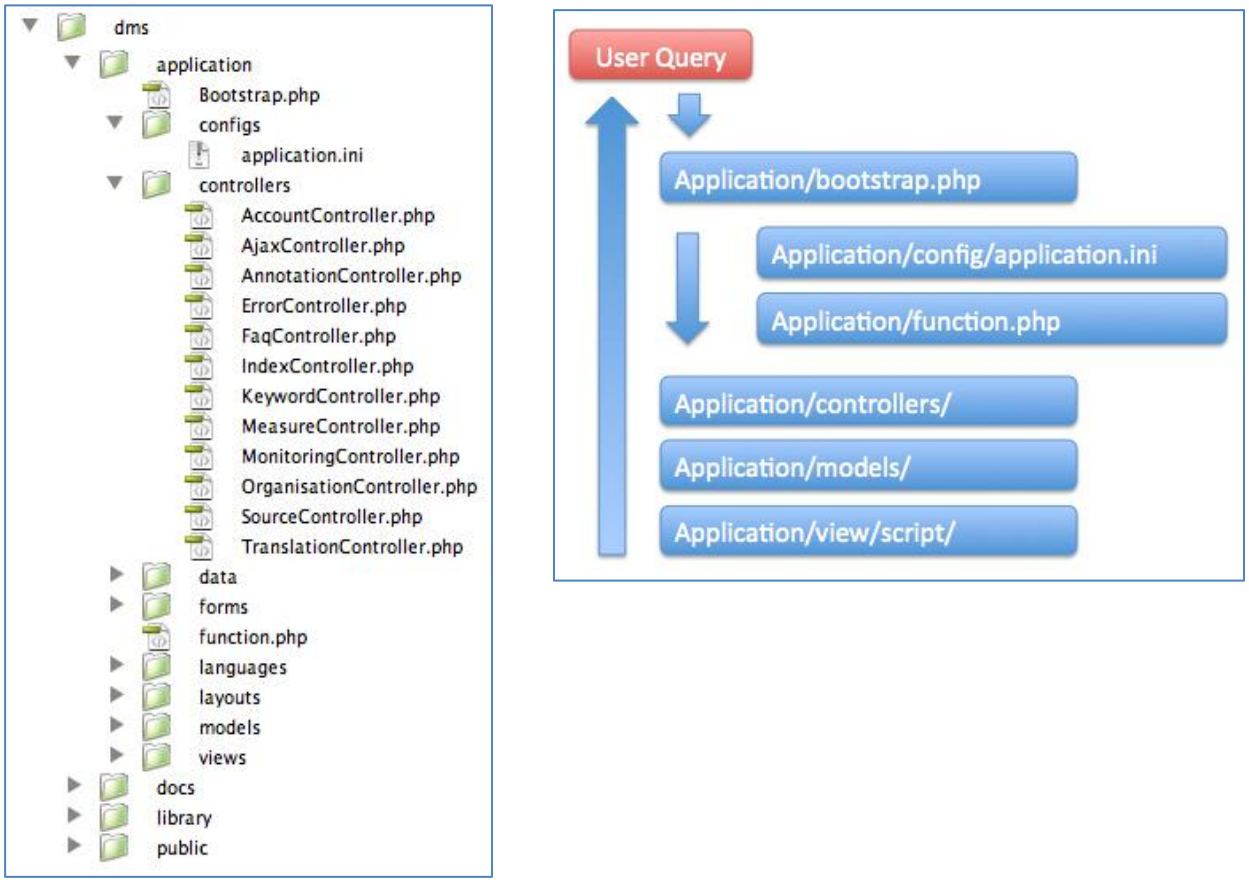

Figure 5.12: MVC structure

Therefore, we will describe the different component and the MVC structure applied briefly but with a focus on the structure and logic. Every class of the MVC structure are also well documented with PHPdocumentor [34].

[17] Model View Controller

Source (Grid Model)	**Aim and function** The *source* component allows team members to add, delete or edit source. For each source, he can annotate it and define Feed (e.g. Rss Feed, social Media Feed…) Each stream can be manually annotated and an automatic crawler is getting stream of the source. A crawler that search constantly new source is also available in this component. **Controller** application/controllers/SourceController.php **Model** application/models/source.php (Class Object) application/models/sourceMapper.php (Class Process) application/models/feed.php (Class Object) application/models/feedMapper.php (Class Process) **View** application/view/script/source/index.phtml application/view/script/source/crawler.phtml application/view/script/source/feedmonitoring.phtml application/view/script/source/feed.phtml

Keyword (Grid Model)	**Aim and function** The *keyword* component allows team members to add, delete, edit and annotate keyword. The *Review interface* allows Team members to see in the crawled post, the keyword that have define the annotation of the post. Aim of this function is to access quickly and directly the result of the Web Crawler to correct easily the keyword definition if needed. A third function enabled to connect keyword to each other keyword. This is dedicated to the Comparativ Analytics tools and affords the possibility to compare post without consideration of the language. **Controller** application/controllers/KeywordController.php **Model** application/models/keyword.php (Class Object) application/models/keywordMapper.php (Class Process) **View** application/view/script/keyword/index.phtml application/view/script/keyword/connection.phtml application/view/script/keyword/monitoring.phtml

Annotations (Grid Model)	**Aim and function** The *Annotation* component allows team members to add or edit Annotation and ItemAnnotation. The annotation are the Meta parameter that compose the Grid Model (e.g. Language, Country…). A subsection page *Relation* allows user to affect to the item that compose the grid model which annotation should be activated. **Controller** application/controllers/AnnotationController.php **Model** application/models/annotation.php (Class Object) application/models/annotationMapper.php (Class Process) application/models/itemAnnotation.php (Class Object) application/models/itemAnnotationMapper.php (Class Process) **View** application/view/script/annotation/index.phtml application/view/script/annotation/relation.phtml

Measure (Grid Model)	**Aim and function** The *measure* component allows team members to add, delete or edit measure and case. **Controller** application/controllers/MeasureController.php **Model** application/models/measure.php (Class Object) application/models/measureMapper.php (Class Process) application/models/measureCase.php (Class Object) application/models/measureCaseMapper.php (Class Process) **View** application/view/script/measure/index.phtml application/view/script/measure/case.phtml application/view/script/measure/create.phtml

Organisation (Grid Model)	**Aim and function** The *organisation* component allows team members to add, delete or edit organisation. The suggest page allow access to organisation that has been submitted by End User in their user account. **Controller** application/controllers/OrganisationController.php

	Model application/models/organisation.php (Class Object) application/models/organisationMapper.php (Class Process) **View** application/view/script/organisation/index.phtml application/view/script/organisation/suggest.phtml

Translation (Grid Model)	**Aim and function** The *translation* component allows team members to translate every element of the Grid model and access the frontend dashboard translation. **Controller** application/controllers/TranslationController.php **Model** application/models/translation.php (Class Object) application/models/translationMapper.php (Class Process) **View** application/view/script/translation/index.phtml

DMS Users (Account)	**Aim and function** The *DMS user* component allows the management of the UniteEurope team. **Controller** application/controllers/AccountController.php **Model** application/models/dmsUser.php (Class Object) application/models/dmsUserMapper.php (Class Process) **View** application/view/script/account/index.phtml

Tool Accounts (Account)	**Aim and function** The *Tool Account* component allows team members to add, delete or edit Account. Each Account can then manage his own user. **Controller** application/controllers/AccountController.php **Model** application/models/account.php (Class Object) application/models/accountMapper.php (Class Process) **View** application/view/script/account/user.phtml

System (Monitoring)	**Aim and function** The *System* component afford a clean overview of the current status of the system. The Crawler status afford an overview about the status of the crawler and display the log if needed. A Red/Green colour convention is used to identify if the crawler is running well or not. The Backend log are displayed and a statistic overview display the amount of Source and Keyword per language. Finally charts display the amount of post by language by day during the last 2 weeks for each stream type (Facebook, Html, Rss, Twitter and Google+). Access to the third tool used lile PhpDocumentor, Bugzilla and Nagios are also available from this page. **Controller** application/controllers/MonitoringController.php **Model** application/models/solrMapper.php (Class Process) application/models/monitoringMapper.php (Class Process) **View** application/view/script/source/index.phtml

Help (Monitoring)	**Aim and function** The *help* component allows team members to read a quick description of the DMS. **Controller** application/controllers/faqController.php **Model** none **View** application/view/script/faq/index.phtml

Several Crawlers have been implemented to monitor the tools or process dedicated task.

library/crawler/crawlerUpdate SourceTypeFeed.php	**Aim** Compare the Source SourceType and the Feed SourceType. Update the Feed SourceType if different with the Source SourceType. **Frequency** Run every hour
library/crawler/crawler.php	**Aim** Collect every Feed (Rss and Social Media) for a new source if the annotation `CrawlStatus` was set to `active` **Frequency**

	Run every minute
library/crawler/countPost.php	**Aim** Count post per source and per feed **Frequency** Run every day at 23h30
library/crawler/countCase.php	**Aim** Count amount of case per organisation **Frequency** Run every day at 23h40
library/crawler/crawlerSocial Media.php	**Aim** Search for all integration keywords in the Blogger.com and Wordpress APIs. The results are added as new RSS feeds (for posts and comments), and are connected to the Blogger.com and Wordpress sources. **Frequency** Run every day at 23h
library/crawler/crawlerSource. php	**Aim** Identification of new source that publish article about integration topics in Google or Yahoo News. **Frequency** Run every 6 hours

The full documentation of the PHP class is done with phpDocumentor and directly available from the DMS.

We invite you to access directly the DMS by using this access information. This access will give you the right to access the DMS but no interaction with the system will be possible.

http://dms4.uniteeurope.org **Login**: EuCommission **Password**: 2013Dms4Eu

5.3.3.2 UniteEurope website and application Tools

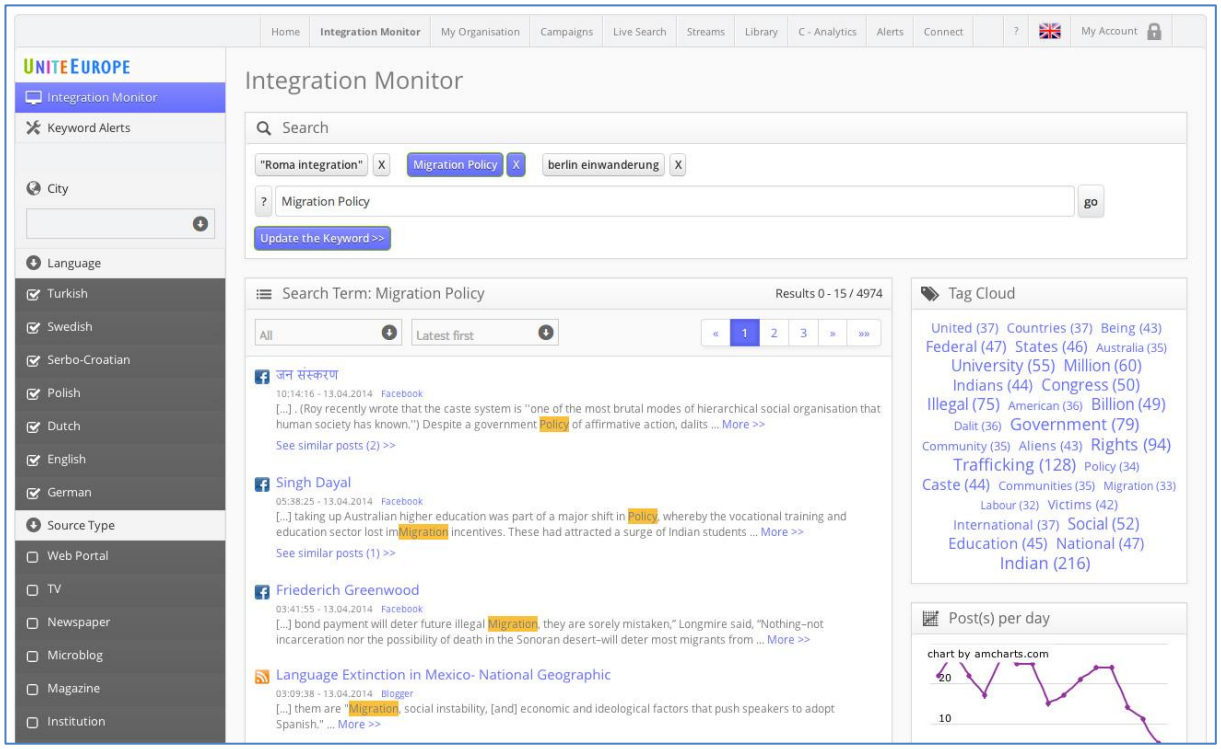

Figure 5.13: http://ww2.uniteeurope.org

The tool, available at http://ww2.uniteeurope.org, has been developed following the same method as the DMS and is built according to the MVC logic. The Website structure is done in Pimcore what allows a convenient and centralized administration of every part of the application Website (Project website and Application).

The first step of the application process is the control of the tools User's User Right status, which is processed in `action.php` file. The user is then following this status redirected to the authorized page. Zend_Acl [35] is used to process the user right management. The user right (see D4.3) are either *Manager* or *Editor* and can access the different component define in his account.

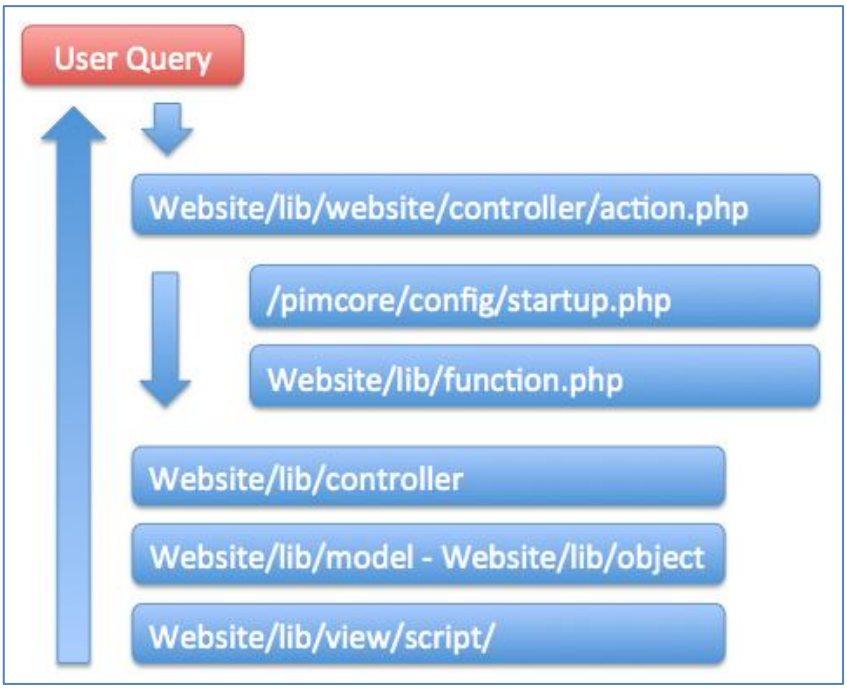

Figure 5.14: Pimcore MVC workflow

Each page created in Pimcore is defined by a controller, an action (which is a function of the controller class) and a template.

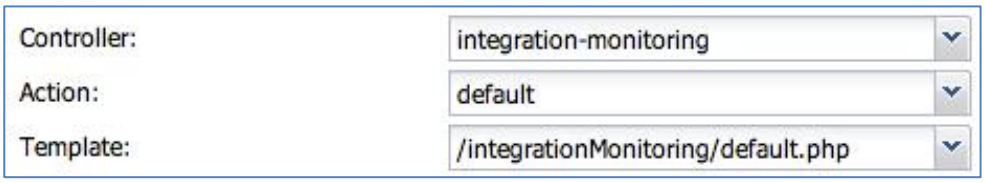

Figure 5.15: Pimcore MVC definition

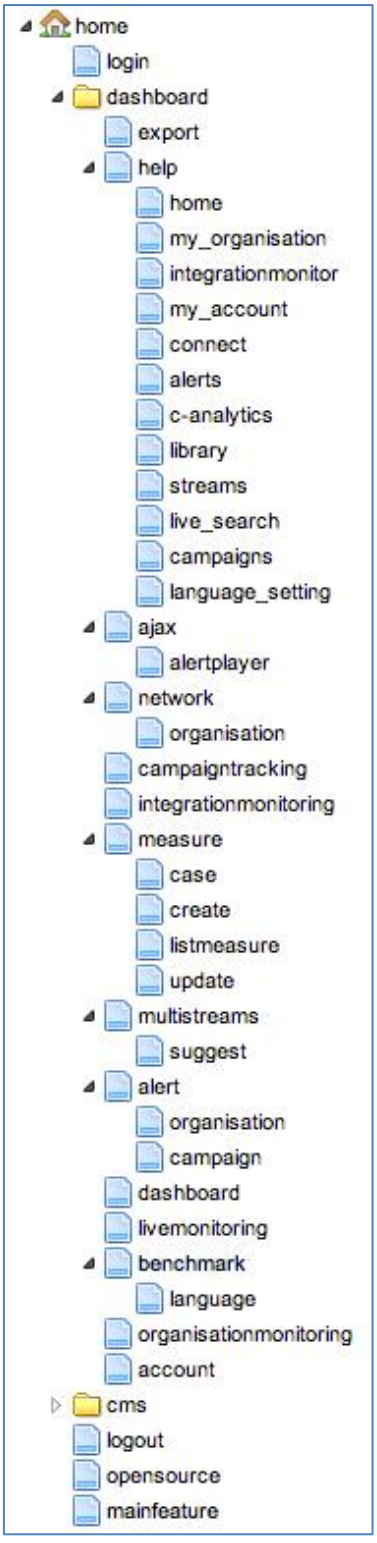

Figure 5.16: Sitemap

Before to present the detail of every part of the tools, a clear overview of the structure is available as sitemap. The sitemap (Figure 5.16) is directly editable in Pimcore and allow a convenient access to each page of the project. The *Dashboard* folder contains the tools component whereas the *Cms* folder and the page available at the root of the sitemap, are public page editable in Pimcore.

The description of each object class and process class is done in PHPDocumentor.

Dashboard	**Aim and function** The *Dashboard* summarizes every part of the tools and afford a quick overview on the Alert results, the latest news of the My Organisation component and the latest organisation and case added in UniteEurope. **Controller** website/controllers/dashboardController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/AlertMapper.php website/Model/MeasureMapper.php website/Model/MeasureCaseMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/dashboard/default.php

Integration Monitor	**Aim and function** The *Integration Monitor* component is the main page to access content from the index. User can search in the UniteEurope index and filter it by Language, Source Type, Stream Type and Integration Area. He can create Alert and filter by period and relevance. Tag-clouds, Pie and Charts afford statistical overview. **Controller** website/controllers/IntegrationMonitoringController.php **Model** website/Model/SourceMapper.php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/AlertMapper.php website/Model/AccountMapper.php **View**

	Layout website/view/layout/application.php *Template* website/view/script/integrationmonitoring/default.php

My Organisation	**Aim and function** The *My Organisation* component display post found in the UniteEurope Index by searching terms define in the Account by UniteEurope KeyAccount manager during the subscription. Content is filtered by Language, Source Type and Stream Type. Alert can be personalised and the content is also filtered by period and relevance. Charts afford statistical overview. **Controller** website/controllers/organisationController.php **Model** website/Model/SourceMapper.php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/AlertMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/organisationMonitoring/default.php

Campaigns	**Aim and function** The *Campaign* component allow access of Campaign alert content. User can filter it by City, Source Type, Stream Type, Period and relevance. Charts and Source Type repartition afford statistical overview. **Controller** website/controllers/CampaignTrackingController.php **Model** website/Model/SourceMapper.php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/AlertMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/campaignTracking/default.php

Live Search	**Aim and function** The Live search function allows direct access to Google+, Twitter, Facebook,

	Yahoo news and Wordpress. User can search and save search terms. The access to this content depend on API limitation. To extend this limitation, several account have been created for each API provide. For exemple, the Twitter API is limited with 450 query on a period of 15 minutes [36]. **Controller** website/controllers/livemonitoringController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/facebookMapper.php website/Model/twitterMapper.php website/Model/googlePlusMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/liveMonitoring/default.php

Streams	**Aim and function** The *Stream* component allow a quick access to selected Organisation news provider. Content is always integration content. To create a community, an each UniteEurope account is connected to an Organisation. UniteEurope teams members can also be inserted directly in the DMS. The activation of each organisation as a Stream organisation is done by the UniteEurope team members in the DMS. A *suggest* section allow each user to submit to the UniteEurope team members some organisation. **Controller** website/controllers/multiStreamsController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/OrganisationMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/multiStream/default.php *Template* website/view/script/multiStream/suggest.php

Library	**Aim and function**

	The *Library* is a directory composed by case edited by the community of UniteEurope users. Each User can create an case. Case can be filtered by Integration Area, Measure Type, Location, Evaluation and Target Group. **Controller** website/controllers/dashboardController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/MeasureMapper.php website/Model/MeasureCaseMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/measure/case.php *Template* website/view/script/measure/create.php *Template* website/view/script/measure/default.php *Template* website/view/script/measure/measure.php *Template* website/view/script/measure/update.php

Comparative Analytics	**Aim and function** The *Comparative Analytics* component allows to compare by country and language the integration keyword of the Grid Model. The different set of data can be filter by time. **Controller** website/controllers/benchmarkController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/KeywordMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/benchmark/default.php

Alerts	**Aim and function** The *Alert* component allow user to add, delete and edit his alert. The template_email_alert.php view is used to design the Email alert. A dedicated crawl (/website/cli/alert.php) is dispatching the alert when the requirement

	defined by the user is validated (Amount of new news reaches and Alert period). **Controller** website/controllers/alertController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/AlertMapper.php website/Model/MeasureMapper.php website/Model/MeasureCaseMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/alert/alert.php *Template* website/view/script/alert/organisation.php *Template* website/view/script/alert/campaign.php *Template* website/view/script/alert/template_email_alert.php

Connect	**Aim and function** The *Connect* component is the Organisation directory. User can search and filter organisation by country. And access quickly his own organisation. By defining itself as Public User, each user profile is listed in his organisation profile. **Controller** website/controllers/networkController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/MeasureMapper.php website/Model/MeasureCaseMapper.php website/Model/OrganisationMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/network/default.php *Template* website/view/script/network/organisation.php

Export	**Aim and function** The Export function is used to generate .csv file. Each result of pie charts, tagclouds and diagrams can be exported as .csv file. 1000 Post with their annotation could be also exported as .csv file. **Controller** website/controllers/ExportController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/AlertMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/csv.php *Template* website/view/script/export/csv.php

Ajax	**Aim and function** The *Ajax* class is used to prepare content required as with Ajax. Alert charts and duplicate post player are currently using Ajax. **Controller** website/controllers/ajaxController.php **Model** website/Model/SourceMapper..php website/Model/SolrMapper.php website/Model/ItemAnnotationMapper.php website/Model/AnnotationToTableMapper.php website/Model/UserMapper.php website/Model/AlertMapper.php website/Model/MeasureMapper.php website/Model/MeasureCaseMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/ajax.php *Template* website/view/script/ajax/default.php *Template* website/view/script/ajax/alertPlayer.php

My Account	**Aim and function** The *My Account* component is used to administrate the account and user profil. User can edit his profile, define the language of the dashboard, edit the account organisation profile and create new user. The My Account functions

	are depending of the User Right of the User. **Controller** website/controllers/accountController.php **Model** website/Model/ItemAnnotationMapper.php website/Model/UserMapper.php website/Model/OrganisationMapper.php website/Model/AccountMapper.php **View** *Layout* website/view/layout/application.php *Template* website/view/script/account/default.php

Default	**Aim and function** The *Default* controller is used to manage the login access and every public page. **Controller** website/controllers/defaultController.php **Model** website/Model/userMapper.php website/Model/monitoringMapper.php **View** *Layout* website/view/layout/help.php *Layout* website/view/layout/public.php *Template* website/view/script/template/public/default.php *Template* website/view/script/template/public/help.php *Template* website/view/script/template/public/home.php *Template* website/view/script/template/public/login.php *Template* website/view/script/template/public/contentVertical.php

To access the content of the index, a model class called `SolrMapper` has been implemented. This class manages the access to the Solr index and processes the query of the user by using the PHP Solr search engine extension [37]. The search operators supported by us are displayed in Table 5.1. More operators offered by Solr could be added by request.

Example	matches...
`discrimination`	texts containing “discrimination”
`discrimination Europe`	texts containing “discrimination” **and** “Europe”
`discrimination OR Europe`	texts containing “discrimination” **or** “Europe”
`discrimination -Europe`	texts containing “discrimination”, but **not** the word “Europe”
`“discrimination against”`	texts containing the **phrase** “discrimination against”
`discriminat*`	texts containing “discriminate” or “discrimination” or “discriminated” etc. (**wildcard**, does not work inside phrases)
`discriminat* (“asylum seekers” OR refugees) -”foreign workers”`	search operators can be combined

Table 5.1: Solr search operators

After the first user test, we noticed that the users prefer to see the extract of the text that contains the search terms, instead of the first lines of the post. Therefore, Solr’s highlighting function has been integrated in the query process and is displaying a teaser of 200 words that contains the search term. This method has the disadvantage to get teaser where the html code could not be efficiently cleaned but display distinctively the search term context.

```
$query ->setHighlight(true)
       ->addHighlightField("content"
       ->setHighlightSnippets(1,"content")
       ->setHighlightFragsize(200,"content");
```

The relevance issue defined in relation with the researcher of the consortium is using the Solr Extended Dismax query Parser [38] in which we apply the formula motivated in D5.7, section 4.2.

```
$query ->addParam('defType','edismax');
       ->addParam('boost','recip(ms(NOW/HOUR,date),3.16e-
11,0.08333,0.08333)');
```

```
        ->addParam('boost','relevance1');
```

Figure 5.17: Ordering search results by relevance and recentness

To generate the different charts and diagram required, we use the faceting overview of Solr [39]. For example, the query below will allow us to draw a multiline diagram in which we can compare the amount of article per IntegrationArea (*iareas* in the Solr index) by day during a period of 14 days.

```
$query ->addFacetField('iareas')
       ->addFacetDateField('date')
       ->setFacetDateStart('NOW/DAY-14DAYS')
       ->setFacetDateEnd('NOW/DAY+1DAY')
       ->setFacetDateGap('+1DAY');
```

The latest implication regarding the content selection is that duplicates of older posts are not displayed. If a post has duplicates, the user can access them in a small window below the post. The full documentation of every PHP class, controller, model are available in the DMS in the section *System*.

We invite you to access directly the tools by using the following access information.

http://ww2.uniteeurope.org
Login: EUFP7-2013
Password: EUFP7-2013

6 Implications

In many respects, UniteEurope has turned out to be similar to other projects which deal with social media analytics. For example, the separation of the project into three different modules (DMS for configuration, web crawler for processing of content, and web application for user interaction) can probably be found in many other analytics tools on the market, even if the exact implementation is realised in different ways. Furthermore, our decision to adopt a **user-centred design** [40] approach is in line with good practices for software development. An early integration of the users in the design process helped us to define functional requirements for the tool, not only from a technical perspective, but covering the needs of social scientists, city administrations and NGOs. An iterative development cycle assured that after every implementation step, feedback on bugs and missing features was given, resulting in improvements of the usability of the tool. This did not only include the visualisation level (such as the highlighting of search terms), but also the functional level (e.g. the possibility to order search results by relevance) and the content processing (e.g. the inclusion of web-scraping capabilities).

On the other hand, there were obviously a lot of challenges which are specific to UniteEurope, making it stand out in the domain of social media tools. The most important one was the difficulty to determine what content is actually **relevant** for the user. Our keyword matching approach, including two different word classes (integration-related and -associated) as well as stemming, seemed to offer a quick and easy way to develop a running prototype in the given time frame. Unfortunately, the amount of work for defining the keywords and a sensible relevance formula was much higher than expected. And what is worse is that we cannot evaluate the approach, due to missing gold-standard data.

Accordingly, it seems that annotating training data for an automatic document classifier (see section 4.2) would have been a better strategy. While this is probably true, it must not be forgotten that the costs involved in such a strategy increase linearly with the number of languages supported, and that the **multilingual** character of UniteEurope is one of its unique selling points. Concretely, we would have to agree on guidelines which define if a text is integration-related, and for the nine languages found in our tool, we would have to educate nine different native speakers to follow these guidelines, so that a certain inter-annotator agreement[18] is reached. Each annotator would then be asked to classify hundreds of documents in his language. In contrast, translating the keywords to all nine languages was rather inexpensive in comparison. And the inter-lingual connection between these keywords allows us to statistically compare their usage over time (*Comparative Analytics*). All things considered, the trade-off between the two classification approaches is clearly one of the interesting outcomes of the project.

Another remarkable feature of our tool is its ability to localise content, required by municipal administrations and local NGOs who need to deal with integration issues in their own region. In contrast to the relevance calculation, keyword matching seems to be very effective for **geolocalisation**. A preliminary inspection of the city filter shows that it produces many posts

[18] The agreement can be measured with Cohen's Kappa score, see [41], chapter 16.

whose content refers to the selected city. Also, the fact that only 9% of all posts are geotagged is not a problem, because we expect that people are more likely to talk about global integration issues instead of local ones. If we can identify this localised content correctly, our work is done. Again, a gold standard would allow for a more detailed evaluation.

Recalling what we said at the end of section 4.2, it seems that the general **interest in the integration topic** is lower than expected. While commercial analytics tools need to distribute data on large server clusters, UniteEurope gets along with a single machine. This is not surprising, however, given that the search for “IPhone” returns ten times more search results in Google than the word “integration”. Considering that 40% of all posts in our index are duplicates (see section 4.3), and that most content cannot be geotagged, it is probably possible for a single user to track all relevant content for the city he lives in.

Finally, an important question for future developments is how to include **comments** from social networks (see section 4.3). In addition, Polish, Serbian, Bosnian and Croatian are underrepresented in our index; we should therefore **add more sources** for them. And the three Serbo-Croatian languages are still missing **stemmers**, so there is a lot of room for improving their keyword matchers.

On the frontend side, we might reconsider the usage of an **asynchronous application platform** like Node.js. This would increase the tool’s performance and allow us to implement real-time functions like interactions between users. However, it would also require us to find a stable, matching MVC framework.

7 Conclusion

The UniteEurope project has set out to develop a social media analytics tool which collects citizen-generated content related to integration issues. The previous sections have illustrated how the overall task was split into smaller units, and distributed over various modules. Starting from the traditional concept of *Web Mining*, we designed an architecture with components for the retrieval, analysis, storage and visualisation of web content.

During the project's duration it became clear that the field of integration overlaps with many similar policy areas like immigration, anti-discrimination etc., and is therefore hard to define. However, we managed to integrate requirements and feedback from diverse user groups. At the time of writing, final improvements are underway in order to incorporate the results from the user tests in D6.3.

For the future, we are confident that the software is robust enough to keep running until the end of the project and beyond. Many well-established development techniques were applied in order to make the tool stable, or easy to fix if need be. The same techniques also make the software adaptable to future requirements, be it for dealing with the problems illustrated in the previous section, or extending UniteEurope to other policy fields.

REFERENCES

[1] A. Graubner-Müller, *Web Mining in Social Media: Use Cases, Business Value and Algorithmic Approaches for Corporate Intelligence*. Social Media Verlag, 2011.

[2] Ambysoft, “Introduction to Test Driven Development (TDD),” 2014. [Online]. Available: http://www.agiledata.org/essays/tdd.html.

[3] “Google Java Style,” 2014. [Online]. Available: http://google-styleguide.googlecode.com/svn/trunk/javaguide.html.

[4] S. Yeates, “What is version control? Why is it important for due diligence?,” 2014. [Online]. Available: http://oss-watch.ac.uk/resources/versioncontrol.

[5] wiseGEEK, “What Is Build Automation?,” 2014. [Online]. Available: http://www.wisegeek.com/what-is-build-automation.htm.

[6] OMG, “UML,” 2014. [Online]. Available: http://www.uml.org/.

[7] Adobe, “Dreamweaver,” 2014. [Online]. Available: http://www.adobe.com/de/products/dreamweaver.html.

[8] The Eclipse Foundation, “Eclipse,” 2014. [Online]. Available: https://www.eclipse.org/.

[9] Vogella, “Installation of JUnit,” 2014. [Online]. Available: http://www.vogella.com/tutorials/JUnit/article.html#eclipse.

[10] D. Schneider, L. Ködderitzsch, and et al, “eclipse-cs,” 2014. [Online]. Available: http://eclipse-cs.sourceforge.net/.

[11] Wikipedia, “Magic number (programming),” 2014. [Online]. Available: http://en.wikipedia.org/wiki/Magic_number_%28programming%29.

[12] Oracle, “Javadoc Tool,” 2014. [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html.

[13] Apache Software Foundation, “Subversion Apache,” 2014. [Online]. Available: http://subversion.apache.org/.

[14] Twitter4J, “Twitter4J,” 2014. [Online]. Available: http://twitter4j.org/en/index.html.

[15] CassandraTargett, “Solrj,” 2014. [Online]. Available: https://wiki.apache.org/solr/Solrj.

[16] Apache Software Foundation, “Maven Apache,” 2014. [Online]. Available: http://maven.apache.org/.

[17] Nagios Enterprises, “Nagios,” 2014. [Online]. Available: http://www.nagios.org/.

[18] RSS Advisory Board, “RSS Advisory Board,” 2014. [Online]. Available: http://www.rssboard.org/rss-specification.

[19] M. Nottingham and R. Sayre, “The Atom Syndication Format,” 2014. [Online]. Available: https://tools.ietf.org/html/rfc4287.

[20] Gnip, “Gnip,” 2014. [Online]. Available: http://gnip.com/.

[21] M. F. Porter, “Snowball: A language for stemming algorithms,” 2001. [Online]. Available: http://snowball.tartarus.org/texts/introduction.html.

[22] A. Z. Broder, S. C. Glassman, and M. S. Manasse, “Syntactic Clustering of the Web,” 2014. [Online]. Available: http://www.std.org/~msm/common/clustering.html#Defining similarity of documents.

[23] C. D. Manning and H. Schütze, *Foundations of Natural Language Processing*. Cambridge: MIT Press, 1999.

[24] Apache Software Foundation, “Apache UIMA,” 2014. [Online]. Available: http://uima.apache.org/.

[25] Apache Software Foundation, “Apache OpenNLP,” 2010. [Online]. Available: http://opennlp.apache.org/index.html.

[26] B. Hartley, “I Don’t Need No Stinking API: Web Scraping For Fun and Profit,” 2010. [Online]. Available: http://blog.hartleybrody.com/web-scraping/.

[27] M. Olivo, “lc4j, a language categorization Java library,” 2014. [Online]. Available: http://olivo.net/2009/08/lc4j/.

[28] Apache Software Foundation, “Apache 2.0 licenced,” 2014. [Online]. Available: http://lucene.apache.org/.

[29] Node.js, “Node.js,” 2014. [Online]. Available: http://nodejs.org/.

[30] MongoDB, “MongoDB,” 2014. [Online]. Available: https://www.mongodb.org/.

[31] C. Pitt, *Pro PHP MVC*. Apress, 2014.

[32] “node-js-vs-apache-php-benchmark,” 2014. [Online]. Available: https://code.google.com/p/node-js-vs-apache-php-benchmark/wiki/Tests.

[33] bugzilla.org, “Bugzilla,” 2014. [Online]. Available: http://www.bugzilla.org.

[34] PhpDocumentor, “phpDocumentor,” 2014. [Online]. Available: http://www.phpdoc.org/.

[35] Zend Technologies Ltd., “Zend Acl,” 2014. [Online]. Available: http://framework.zend.com/manual/1.12/en/zend.acl.introduction.html.

[36] Twitter, “REST API v1.1 Limits per window by resource,” 2014. [Online]. Available: https://dev.twitter.com/docs/rate-limiting/1.1/limits.

[37] PHP Group, “Apache Solr,” 2014. [Online]. Available: http://www.php.net/manual/en/book.solr.php.

[38] E. Pugh, “ExtendedDisMax,” 2013. [Online]. Available: http://wiki.apache.org/solr/ExtendedDisMax.

[39] C. Grobmeier, “SolrFacetingOverview,” 2014. [Online]. Available: http://wiki.apache.org/solr/SolrFacetingOverview.

[40] D. A. Norman and S. W. Draper, *User Centered System Design: New Perspectives on Human-Computer Interaction*. CRC Press, 1986.

[41] J. Bortz, *Statistik für Sozialwissenschaftler*. Springer, 1993.

ANNEX A: Server Installation Instruction

Installation of a UniteEurope Server

In general, there are two options for the installation process.

Option 1: Use only Solr as data repository. All CRUD operations are directly performed from a Solr client (such as Solrj or via the REST API). The installation and maintenance process is much easier than for option 2. However, we don't know how well Solr scales for large data volumes.

Option 2: Use a full Hadoop/HBase cluster as data repository. HBase scales very well for arbitrary amounts of data, since it can be distributed over multiple machines. We would still use Solr for searching documents, but since they are already in HBase, we do not need to store large document fields (such as content and title) in Solr; these only need to be indexed. The crawler will add, delete and update data via the Lily wrapper library. The UniteEurope web interface will find relevant record IDs via Solr and then look them up in HBase.

We first describe the installation procedure for Option 1, because it is a precondition for Option 2, which is specified afterwards.

Option 1: UniteEurope installation with a Solr repository

The following configuration steps are exemplary for a server with the internal IP `192.168.1.22`, public IP `91.250.87.196` and hostname `unitEUrope`. When choosing option 2, the server's operating system must be compatible with the requirements of Cloudera Manager. We recommend Ubuntu 12.04. LTS.

1. Create a new user "hduser" for running the UniteEurope Java processes: `adduser hduser`
    a. Enable password-less sudo for hduser by adding the following line to /etc/sudoers:
    `hduser ALL=(ALL) NOPASSWD: ALL`
    b. Adapt the .bashrc file to your needs.
2. As root, install the latest JDK for Java 7 SE
    a. `mkdir /usr/lib/jvm`
    b. In your browser, go to http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html
    c. Accept the license agreement (!), and start downloading the Linux x64 JDK (.tar.gz for Ubuntu). When the download starts, figure out the actual URL of the downloaded file (in Firefox: left-click on the download progress bar -> left-click on the download information -> "Copy download link").
    d. Download the JDK on the server:
        i. `cd /usr/lib/jvm`

ii. `curl -O -L 'http://download.oracle.com/otn-pub/java/jdk/7u51-b13/jdk-7u51-linux-x64.tar.gz?AuthParam=1392831242_dd4f384bd5b69ca3af30e422a6423e8c'`
When the download begins, you can cancel the download in your browser.

e. Remove the "?AuthParam" part in the downloaded file's name, e.g.
`mv jdk-7u51-linux-x64.tar.gz\?AuthParam\=1392... jdk-7u51-linux-x64.tar.gz`

f. Unzip and untar the file.

g. `ln -s jdk1.7.0_51/ jdk1.7.0`

h. `chown -R root:root jdk1.7.0_51/`

i. Append the following to the .bashrc of all users:
```
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0/
export PATH=${PATH}:/usr/lib/jvm/jdk1.7.0/bin
```

3. Install MySQL:
   a. `apt-get install mysql-server`
   b. We use the public IP in /etc/mysql/my.cnf to make the DB accessible from anywhere:
   `bind-address = 91.250.87.196`
   c. Create a dump of a previous database:
   `mysqldump --user=root --password --databases uniteeurope --opt > mysql_uniteeurope_20130723.sql`
   d. Load the dump on the new server:
   `mysql -u root -p < mysql_uniteeurope_20130723.sql`
   e. `mysql -u root -p`
   f. `create user 'cmsadmin'@'%' IDENTIFIED BY 'some password';`
   g. `GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER ON `uniteeurope`.* TO 'cmsadmin'@'%' IDENTIFIED BY 'some password';`
   h. `create user 'crawler'@'%' IDENTIFIED BY 'some password';`
   i. `GRANT SELECT, INSERT, UPDATE, DELETE ON `uniteeurope`.* TO 'crawler'@'%' IDENTIFIED BY 'some password';`
   j. `GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER ON `uniteeurope`.`DmsAnnotatorDaemon` TO 'crawler'@'%' IDENTIFIED BY 'some password';`
   k. `GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER ON `uniteeurope`.`DmsKeywordCrawlStatistics` TO 'crawler'@'%' IDENTIFIED BY 'some password';`
   l. `flush privileges;`
4. Install Solr:
   a. `wget` http://apache.openmirror.de/lucene/solr/4.4.0/solr-4.4.0.zip
   b. unzip the file

c. Copy the log4j.properties from a previous Solr installation in `solr-4.x.y/example/resources/log4j.properties`
d. Copy the `solr-4.x.y/example/solr/collection1/conf/schema.xml` from a previous Solr installation. ATTENTION: Depending on your choice of option 1 or 2, the schemas must have different fields. For option 2, see schema_lily.xml.
e. Copy stopwords_all2.txt from a previous Solr installation
f. Adapt auto(Soft)Commit settings in `conf/solrconfig.xml`. Also, the default solrconfig.xml comes with default field "text" (df) which we do not have in our schema.xml, so replace it with "title".
g. Securing Solr by password does not work, because the UniteEurope frontend does not support it. So we have to close the port instead, and use Solr on the command line or via an SSH tunnel.
h. Start Solr:
    i. `cd /home/lily/programme/solr-4.x.y/example`
    ii. `java -Xms128M -Xmx3000M -jar start.jar &> logs/stdout.log &`
i. Open the Solr web interface at http://192.168.1.22:8983/solr/#/ (the port can be configured in `solr-4.x.y/example/etc/jetty.xml`)

5. Create the directory `/home/hduser/PROD` with all necessary files and folders:
    a. `mkdir log`
    b. Copy .properties files from a previous installation, incl. logging properties.
        i. In uniteeurope.properties, adapt the IP values to your current environment.
        ii. In the logging properties files, adapt the logfile path parameters.
    c. Shell scripts for starting the Java processes
    d. Jar files
6. Create new Twitter accounts (one account can track up to 400 keywords), and update the Twitter authentication details in uniteeurope.properties.
7. Install Gisgraphy. It is used by the TwitterAuthorLocationDaemon, which extracts geographical information from Twitter user profiles.
8. Add the server's IP to the list of allowed IPs on the Facebook App page: https://developers.facebook.com/apps/400590523352979/advanced?ref=nav
9. Add the server's IP to the list of allowed IPs on the Google+ Plus App page: https://code.google.com/apis/console/?pli=1#project:1011739132975:access
10. As user hduser, start the shell scripts:

```
./start_CrawlerDaemon.sh 1.0.xyz 1.0.xyz
./start_AnnotatorDaemon.sh 1.0.xyz 1.0.xyz
./start_TwitterAuthorLocationDaemon.sh 1.0.xyz 1.0.xyz
```

Option 2: UniteEurope installation with an HBase repository

1. Add the option "noatime" to the / file system in `/etc/fstab`, according to the Hadoop book[19], p. 275.
2. Edit `/etc/hosts` so that the $hostname resolves to the internal IP and NOT to the loopback address (127.0.0.1), according to the HBase book, p. 48:
   `192.168.1.22  unitEUrope`
3. `apt-get install ntp`
   `/etc/init.d/ntp stop`
   `ntpdate 0.ubuntu.pool.ntp.org`
   `/etc/init.d/ntp start`
4. Make sure the DHCP client configuration enables reverse-lookup from the internal IP to the hostname. Edit `/etc/dhcp/dhclient.conf :`
   `send dhcp-client-identifier 1:00:22:15:f3:df:e9;` ("1:" followed by hardware address)
   `send host-name "your_hostname";`
5. Install Cloudera Manager and CDH. In this step, we follow http://www.cloudera.com/content/cloudera-content/cloudera-docs/CM4Ent/latest/Cloudera-Manager-Installation-Guide/cmig_install_path_A.html?scroll=cmig_topic_6_5
   a. Download Cloudera Express 4 (or whatever product currently contains CDH) from https://ccp.cloudera.com/display/SUPPORT/Downloads
   b. `chmod 700 cloudera-manager-installer.bin`
      `chown root cloudera-manager-installer.bin`
   c. `./cloudera-manager-installer.bin`
   d. After the installation has finished, check all files in `/var/log/cloudera-manager-installer`
   e. Also check logs in `/var/log/cloudera-scm-server`
   f. If everything is fine, proceed to the web interface: http://192.168.1.22:7180/
   g. Follow the dialog and install CDH:
      i. Choose the CDH 4.2.x or according to http://docs.ngdata.com/lily-docs-current/414-lily/432-lily.html. Other versions will cause problems!
      ii. You may have to add another parcel repository URL, like http://archive.cloudera.com/cdh4/parcels/4.2.2/
      iii. Do not select Solr.
   h. Copy changed settings from a previous Cloudera Manager installation to the current one. Examples (list may not be complete):
      i. set `replication = 1` if running on only one machine
      ii. `DataNode -> Advanced -> DataNode Logging Safety Valve:`
          `log4j.logger.org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace = WARN`
      iii. `HDFS -> Service-Wide`
           `-> Check HDFS Permissions = false`

[19] http://www.amazon.de/Hadoop-Definitive-Guide-Tom-White/dp/1449311520/

iv. Disable warning about underreplication of blocks:
`HDFS -> Monitoring -> Service-Wide -> Thresholds -> Under-replicated Block Monitoring Thresholds -> "Never" / "Never"`

i. Change the Cloudera Manager admin password.

6. In the CDH web interface, stop all services (including the *mgmt* services). Afterwards, stop Cloudera Manager:
```
service cloudera-scm-agent stop
service cloudera-scm-server stop
service cloudera-scm-server-db stop
```
7. Cloudera will automatically install an outdated Java SDK which causes problems due to a bug in `sun.net.www.ParseUtil`. You can fix this by renaming the outdated Java and creating a link to Java 7.
```
ls –la /usr/lib/jvm:
drwxr-xr-x 11 root root Jul 22 17:23 bak.j2sdk1.6-oracle.bak
lrwxrwxrwx  1 root root Jul 31 12:42 j2sdk1.6-oracle ->
jdk1.7.0_25/
lrwxrwxrwx  1 root root Jul 22 12:32 jdk1.7.0 -> jdk1.7.0_25/
drwxr-xr-x  8 root root Jun  6 06:07 jdk1.7.0_25
```
8. Restart the server. After restart, check logs in `/var/log/cloudera*`, and start all services in the CDH web interface.
9. Create a new user "lily" for running the CDH, Solr and Lily processes:
`adduser –gid XYZ lily`
(XYZ must correspond to the ID of group "hadoop" in `/etc/group`, e.g. 999).

10. Add the following to `/etc/security/limits.conf`, according to the HBase book, p. 50:

```
hdfs  -       nofile  32768
hbase  -       nofile  32768
lily  -       nofile  32768

hdfs  soft    nproc   127847
hdfs  hard    nproc   127847
hbase  soft    nproc   127847
hbase  hard    nproc   127847
lily  soft    nproc   127847
lily  hard    nproc   127847
```

11. In the file `/etc/pam.d/common-session`, add the following:

```
session required  pam_limits.so
```

12. You can test the two previous steps with:

```
su lily
ulimit –a
```

13. Install and configure a firewall:
    a. `apt-get install ufw`
    b. `ufw allow 22`
    c. `ufw enable`
    d. In addition, allow the following ports:

Port	From	Description
22	*	SSH
3306	*	MySQL
53	192.168.122.1	DNS
953	192.168.122.1	DNS
8983	192.168.122.1	Solr
7180	*	Cloudera Manager Web Interface
7182	192.168.122.1	Cloudera Manager Web Agent->Server heartbeats
7184	192.168.122.1	Cloudera EventCatcherService
7185	192.168.122.1	Cloudera EventCatcherService
8084	192.168.122.1	Cloudera EventCatcherService
7432	192.168.122.1	Cloudera Manager PostgreSQL database
9000	192.168.122.1	Cloudera Manager Agent Internal
9001	192.168.122.1	Cloudera Manager Agent Control + Status

9994	192.168.122.1	Cloudera Host Monitoring internal
9995	192.168.122.1	Cloudera Host Monitoring external
8090	192.168.122.1	Cloudera Host Monitoring external
9996	192.168.122.1	Cloudera Service Monitoring internal
9997	192.168.122.1	Cloudera Service Monitoring external
8086	192.168.122.1	Cloudera Service Monitoring
9998	192.168.122.1	Cloudera Activity Monitoring internal
9999	192.168.122.1	Cloudera Activity Monitoring external
8087	192.168.122.1	Cloudera Activity Monitoring external
10101	192.168.122.1	Cloudera Alert Publisher
8020	192.168.122.1	NameNode
8022	192.168.122.1	NameNode
50010	192.168.122.1	NameNode
50020	192.168.122.1	DataNode
50070	192.168.122.1	DataNode
50075	192.168.122.1	DataNode
8021	192.168.122.1	JobTracker
9290	192.168.122.1	JobTracker Thrift Plugin
50030	192.168.122.1	JobTracker
50060	192.168.122.1	TaskTracker
4867	192.168.122.1	TaskTracker
57236	192.168.122.1	TaskTracker
60000	192.168.122.1	HBase Master IPC
60010	192.168.122.1	HBase Master HTTP
60020	192.168.122.1	HBase Region IPC
60030	192.168.122.1	HBase Region HTTP
50090	192.168.122.1	Secondary NameNode
2181	*	Zookeeper Client port
9010	192.168.122.1	Zookeeper JMX
33417	192.168.122.1	Zookeeper
51852	192.168.122.1	Zookeeper

12020	192.168.122.1	Lily clients

e. Finally, close all other ports:
```
ufw default deny
```
f. From a different machine, scan for remaining open ports:
```
nmap -v -p 1-10000 -sT 91.250.87.196
```

14. Install Lily as user “lily”:
    a. `wget 'http://lilyproject.org/release/2.0/lily-2.0.tar.gz'`
    b. Unzip the file
    c. Copy relevant config file changes from a previous server, for example:
        i. Replace localhost with internal IP in `hadoop.xml`, `hbase.xml`, `mapreduce.xml`, `zookeeper.xml` and `repository.xml`.
        ii. Increase the ZooKeeper session timeout.
        iii. Jobtracker is not listening on port 9001, but 8021.
    d. Copy Lily index definition `uecrawler.xml` from existing server to `lily-2.0/samples`.
    e. Copy all necessary Lily libraries to the HBase lib directory:
```
while read line ;
do echo $line ;
cp $line /opt/cloudera/parcels/CDH/lib/hbase/lib ;
done < lib/copy_to_hbase.txt
```
    f. Restart the HBase service.
    g. `lily-2.0/service/lily-service start`
15. In `/home/hduser/PROD`:
    a. From a previous UniteEurope installation, add a file `./META-INF/services/org.apache.hadoop.fs.FileSystem`
    b. In uniteeurope.properties, set `LilyRepository.useLily = true`
16. Create the record type `Post` and all necessary field types programmatically. It seems that this cannot be done from a remote machine, because port 8020 is only accessible from internal IPs. Instead, use `org.uniteeurope.uecrawler.LilyRecordUtilitiesTest.java#test-CreatePostFromScratch()`.
    Make sure that you are using the appropriate Lily client library in the file pom.xml while compiling!
17. As user lily, create the Lily index:
```
cd programme/lily-2.0/bin/
./lily-add-index -c /home/lily/programme/lily-2.0/samples/uecrawler.xml -n Post -sm classic -s shard1:http://192.168.122.1:8983/solr -z 192.168.122.1:2181
```
18. In Cloudera manager, decrease the logging level at `Datanode -> advanced`
19. Start the crawler as in the last step of Option 1.