FP7-ICT-2011.4.2
Contract no.: 288342
www.xlike.org

Deliverable D4.3.1

Early event extraction prototype

Editor:	
Author(s):	Gregor Leban, JSI, Janez Brank, JSI;
Deliverable Nature:	P
Dissemination Level: (Confidentiality)	PU
Contractual Delivery Date:	M24
Actual Delivery Date:	M24
Suggested Readers:	XLike project partners
Version:	1.0
Keywords:	event detection, topic tracking, cross-lingual document clustering, event registry

Disclaimer

Full Project Title:	XLike – Cross-lingual Knowledge Extraction
Short Project Title:	XLike
Number and Title of Work package:	WP5
Document Title:	4.3.1 Early event extraction prototype
Editor (Name, Affiliation)	Gregor Leban, JSI
Work package Leader (Name, affiliation)	JSI
Estimation of PM spent on the deliverable:	9

Executive Summary

The deliverable presents an early version of a prototype for automatic event extraction. It starts by introducing the concept of an event and the core features that are used to describe it. The events are the main building block for our main goal, which is building a global event registry that would contain automatically identified events that occurred across the world and are being written about in different languages.

In order to identify events we developed a clustering algorithm that is able to group articles based different article features. Each identified cluster of articles initially represents a separate event. Once a cluster of articles is identified we extract key information about the event – title of the event, time when it happened, location and main entities involved in it. Since each cluster contains only articles in the same language we also need to identify and merge it with clusters in other languages that are describing the same event. We achieve this using the cross-lingual document linking approach described in D4.1.1. After the processing of an event is complete, we store it in the event registry. Event registry provides an API that allows the users to search for events based on several criteria, to visualize and aggregate the search results and to view individual event information. Based on the extracted event information we are also able to compare events and identify other related events.

The prototype described in this document depends on former parts of the project – WP1, WP2 and WP3. It is assumed here that the language processing pipeline developed in these work packages is prepared and functioning.

The developed prototype is currently available at http://eventregistry.ijs.si/.

Table of Contents

List of Tables

List of Figures

Abbreviations

API	Application programming interface
BIC	Bayesian information criterion
BOW	Bag-of-words
B/S	Browser/Server
LOD	Linked open data
TF-IDF	Term frequency – inverse document frequency
UML	Unified modelling language

Definitions

Article	An instance of a news report.
Entity	*Person*, *Organization* or *Location* contained in the *Title* or *Content* of an *Article*
Concept	*Entity* or a general thing that can be recognized and annotated in the articles (such as hunger strike, lawyer, military, cost, debate, etc.)
Cluster	A group of articles describing the same event
Event	An event is anything significant that is occurring in the world
Event registry	A database of identified events

1 Introduction

There are thousands of news articles written and published every day by news agencies all across the world. They are written in various languages and discuss all possible topics. A large percentage of these articles are discussing world events – current, past and future. There is no generally accepted definition of an event, but one intuitive definition is that an event is any significant happening in the world. Two instances of an event are, for example, Felix Baumgartner's jump from a helium balloon on October 14, 2012 and bombings during the Boston marathon on April 15, 2013.

The way people today consume news and learn about world events leaves much to be desired. Firstly, despite being interested in events, the actual "unit" of content that we consume is an article. If we wish to have broader, more complete and objective knowledge of the event we need to manually check multiple news sources and find different articles about the event. Secondly, although most articles about events answer the main questions about who, where, when and what, this information remains hidden in the text and requires the reader to manually extract it by reading the article. The lack of explicit knowledge does not only mean inefficient consumption of news for the reader, but also prevents us from having any kind of search engine for finding events based on these criteria. Lastly, our options for exploring related or similar events are very limited. News sources do sometimes provide links to related articles but they are typically bound to the same news source and contain just a few articles that were manually selected by the editors.

There is a lack of information also on the side of the news publishers. They have no analytics that could in each moment identify the events that are currently trending by other similar publishers. Similarly, they are unable to see how good/bad their coverage is of individual topic areas compared to other news publishers.

In this deliverable (D.4.3.1) we will present an early prototype for event extraction. Our goal is to analyse the published articles and identify in them the events that are being reported. The task is closely related to the task of topic detection and tracking (TDT) [JA98] which was a DARPA sponsored initiative to finding and following new events in the stream of news articles. The TDT problem consists of three major tasks: 1) segmenting news stream into different stories, 2) identifying those articles that are describing a new event, and 3) given a small sample of articles about an event finding all other articles describing the same event. Similarly as in TDT, we also want to identify groups of articles that are describing the same event. In addition, however, we also want to process the groups of articles in order to extract from them relevant event information, such as time of the event, location, what the event is about, event type, etc. Due to importance of cross-linguality in the XLike project we also want to see articles in different languages discussing the same event being recognized and represented as a single event. Additionally, we also want to build a global repository of all recognized events and provide search functionality for identifying events based on various search criteria. We want to enable the users to view individual events as well as see an overview of several events using different kinds of visualizations.

In the next section we will describe in details the individual parts of the pipeline. We will start by describing the data collection process and the ways in which each article is processed before it is being used. We will present the clustering algorithm that we use for identifying the groups of articles in the same language that describe a single event. Next, the cluster merging approach will be described that can identify if articles in different languages are discussing the same event. When groups of clusters are identified we then extract from them the relevant event information – title, date, location, entities, etc. We also describe the event registry and its main API calls that can be used to search for events. In the end we list the future work and end with a conclusion.

2 Event detection pipeline

In order to detect and store events we use the pipeline shown in Figure 1. The pipeline contains four main parts:

1. Input data; provides input data that can serve for event extraction
2. Pre-processing steps; collected articles are individually processed by linguistic tools developed in WP2 and WP3 in order to extract available semantic information
3. Event construction; in this phase we identify groups of articles that correspond to individual events. From each identified group we extract available event information (time, location, entities, and type).
4. Event storage & maintenance; after the events are constructed they are stored in a database of events and are indexed across different fields to provide search functionalities.

In the rest of the section we will now describe in more detail individual parts of the pipeline together with the expected inputs and outputs.

2.1 Input data

In order to identify events we need to use some learning data. In our case, the learning data is a feed of articles that we obtain from the Newsfeed service (http://newsfeed.ijs.si/) [D1.3.1]. The service constantly monitors and collects articles from more than 75.000 RSS feeds. The feeds represent various worldwide news sources (majority) or blogs (minority).

2.2 Pre-processing steps

Before we can try extracting events from the collected articles we want to extract as much semantic information from the articles as possible. Currently, there are four main tasks that we rely on: semantic

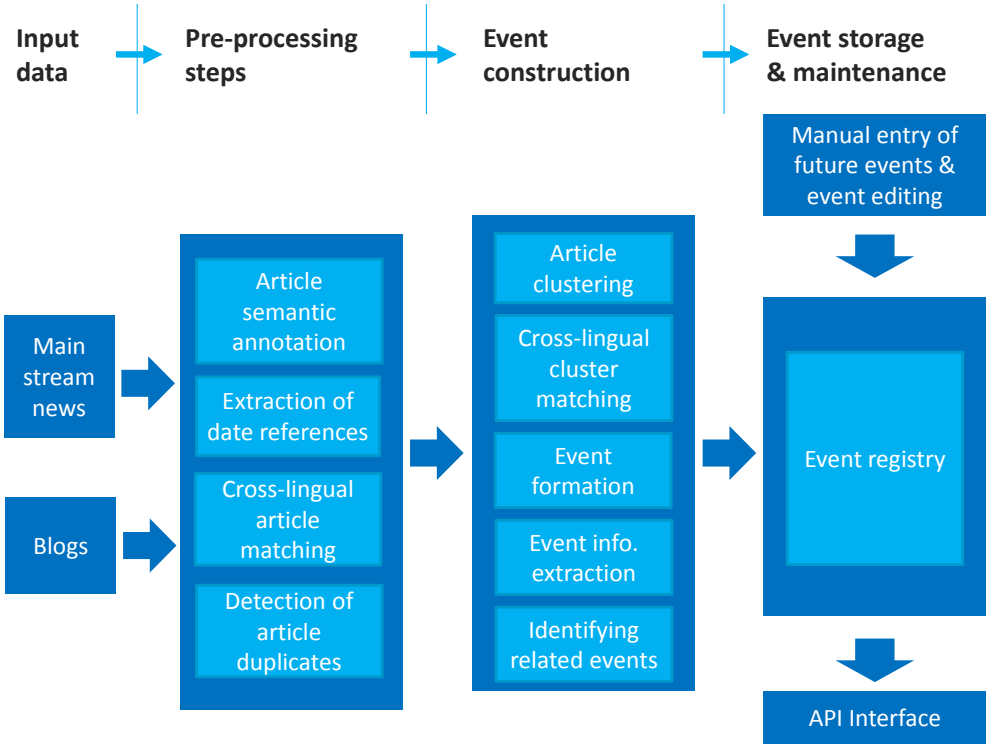

Figure 1 Pipeline used for detection and storage of events

annotation of article text, cross-lingual article matching, extraction of date references, and detection of article duplicates.

Semantic annotation of article text

Semantic enrichment of text is provided by services developed in WP2 and WP3. For the purpose of event extraction, the most important part of enrichment is the detection of named entities [D2.1.1, D3.1.1] and the ontology based word-sense-disambiguation [D3.2.1]. When extracting information about an event, the detected named entities enable us to understand what the event is about and where it occurred.

Cross-lingual article matching

The Cross-Lingual Similarity Service (CLSS) (described in [D4.1.1]) can be used for providing cross-lingual functionality in event extraction. The service can compute an approximate similarity between English, German, Spanish and Chinese news articles. Computation of the cross-lingual similarities is based on an aligned set of basis vectors obtained by one of two methods: latent semantic indexing (LSI) and a generalized version of canonical correlation analysis (CCA) by using an aligned multi-lingual corpus. Given a newsfeed article as an input, it returns ids of top 10 most similar articles for each language in JSON format (see example below). The service uses one day buffer of articles which is suitable for event detection since an event is typically reported for a short time period.

```
{"id":66701562,
  "similar_articles_spa": [{"id":66701512,"sim":0.1364}, …],
  "similar_articles_deu": […],
  "similar_articles_fra": […],
  …
}.
```

Extraction of date references

This component is responsible for identifying date references in text of the articles. Detecting date references is important for the purpose of determining the time when the event happened. Date detection was not yet developed as a part of any other work packages so we had to implement it for the purpose of event extraction. Our implementation is based on a set of regular expressions developed for each language separately. Based on the article language, the appropriate set of regular expressions is selected and used to identify date occurrences. Since events can either occur at a single point in time (Bombing attack in Syria) or across a certain time period (starting and ending date, e.g. 27th July – 12th August for London Summer Olympic Games) we developed a separate group of expressions for detecting date ranges and a group for expressions for detecting single date occurrences.

An example of a single regular expression for detecting a date range from English articles is provided below:

```
\b(?P<day1>\d{1,2})(?:st|nd|rd|th)?(?:(?:\sof\s)|(?:-|–|—)|(?:\.\s?)|\s)?
(?P<month1>(?:jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)[a-z]*)\.?(?:(?:\s? (?:-|–|—
|,|')?\s?)?(?P<year1>\d{2}|\d{4}))?(?:\s?(?:to|through|until|and|(?:-|–|—))\s?)
(?P<day2>\d{1,2})(?:st|nd|rd|th)?(?:(?:\sof\s)|(?:-|–|—)|(?:\.\s?)|\s)?
(?P<month2>(?:jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)[a-z]*)\.?(?:(?:\s? (?:-|–|—
|,|')?\s?)?(?P<year2>\d{2}|\d{4}))?\b
```

The expression can identify date mentions in the form of “15th of September 2003 to 20th of November 2012” including numerous variations. Similarly, an example of a regular expression for matching a single date occurrence looks like:

```
\b(?P<month>(?:jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)[a-z]*)\.?(?:(?:\sthe\s)|(?:-|–|—
)|\s)(?P<day>0?[1-9]|[12][0-9]|3[01])(?:st|nd|rd|th)?(?:,?\s'?(?P<year>\d{2}|\d{4}))?\b
```

This expression can be used to identify date occurrences such as “September 22, 2013”.

As can be seen from both examples, both types of expressions also match incomplete date references (such as September 2012 or 15th of July). Incomplete dates cannot represent a valid date for an event, but as it will be seen in section "Extracting event date" we can alleviate this problem if we are able to identify similar articles with complete date references.

Another problem that occurs when extracting date occurrences is ambiguity caused by different formats in which dates can be written (m/d/y in U.S. vs. d/m/y in Europe). When month or day digit is above 12 there is no ambiguity. In other cases we have to, however, mark the date as ambiguous since we cannot assure the correct date interpretation.

The procedure for identifying the date occurrences works in two steps. In first step we search for possible date ranges. Detected dates, if found, are marked as seen before we run the next step in which we search for individual date mentions, which were not yet detected in the first step. All detected dates are normalized into the form YYYY-MM-DD. If year or day information is missing, the corresponding part is left empty. Information about the detected dates is then stored in XML format inside <article> XML tag. Here is an example of a detected date and date range in XML format:

```
<date date="2012-09-" ambiguous="false" />
<daterange dateStart="2012-01-13" dateEnd="2012-01-18" ambiguousStart="false" ambiguousEnd="false"
/>
```

Detection of article duplicates

By analysing the articles from Newsfeed we found that it provides many duplicated articles, which contain the same or almost the same content as other articles. Duplicated articles are sometimes generated when a news source updates and republishes their own article, or by a news source that simply copies an existing article from a different publisher. Here is an example of two such articles:

Article 1:
Title: Record profit signals healthier Fannie Mae
Body: WASHINGTON (AP) -- Fannie Mae said something Thursday that would have been unthinkable a...
News source: San Francisco Chronicle

Article 2:
Title: Record profit signals healthier Fannie Mae
Body: WASHINGTON (AP) -- Fannie Mae said something Thursday that would have been unthinkable a...
News source: Omaha World-Herald

In principle, there is no issue in having multiple copies of the same articles in the event detection system. There are however some parts of the pipeline that can be negatively affected by such duplicated content, especially if the duplicates are numerous (one of our tests identified articles that had more than 100 duplicated articles). In article clustering, for example, that will be described in the next section, the articles are grouped by their similarity using a clustering method. In case a cluster contains multiple copies of the same article, this can significantly skew the cluster's properties – its centroid, variance and the medoid article. Additionally, the duplicated articles can overestimate the support for cluster merging in the cross-lingual cluster merging (Section 2.4).

In order to determine if an article is a duplicate or not we implemented the following procedure. Given a new article, we first compute a hash code using the article title. We then check a dictionary containing existing hash codes and find previously seen articles with the same hash code (meaning, the previous articles with the same title). Many duplicated articles keep the same title, so finding articles with the same title provides us candidates, which we wish to explore further. In the next step we check the actual article content of each of the candidates. Since news sources sometimes add small modifications (such as adding the title of the news source) we do not want to directly compare article content. Instead, we transform the article contents into the bag-of-words form and compare the individual word frequencies. If the discrepancy between two articles is less than 5 words then we mark the article as a duplicate and further steps in the pipeline can take this information into account.

Since titles are also sometimes modified we do a similar comparison by computing the hash code on the tested article content. We then find other articles with same hash code for their content and test the candidates by comparing the bag-of-words. As before, if difference between articles is less than 5 words then article is marked as a duplicate.

2.3 Article clustering

Article clustering is performed using a NewsCluster, which is a web service for microclustering a stream of news documents, based largely on the approach of C. Aggarwal *et al.* [AY06, AY10, AHWY03]. The service maintains a set of documents partitioned into a number of (relatively small) clusters. Old documents are periodically discarded, thereby keeping the cluster structure focused on the current state of the data stream.

Architecture

The NewsCluster web service runs as a simple HTTP server. It accepts incoming HTTP requests containing the text of new documents that need to be added to the clustering, and it reports the resulting (document ID, cluster ID) pairs to a set of zero or more "listeners", i.e. other web services whose URLs are passed to the NewsCluster service as command line parameters. The NewsCluster service periodically saves its state to disk and performs other maintenance operations, such as discarding old documents and deleting clusters that fall below a minimum size threshold.

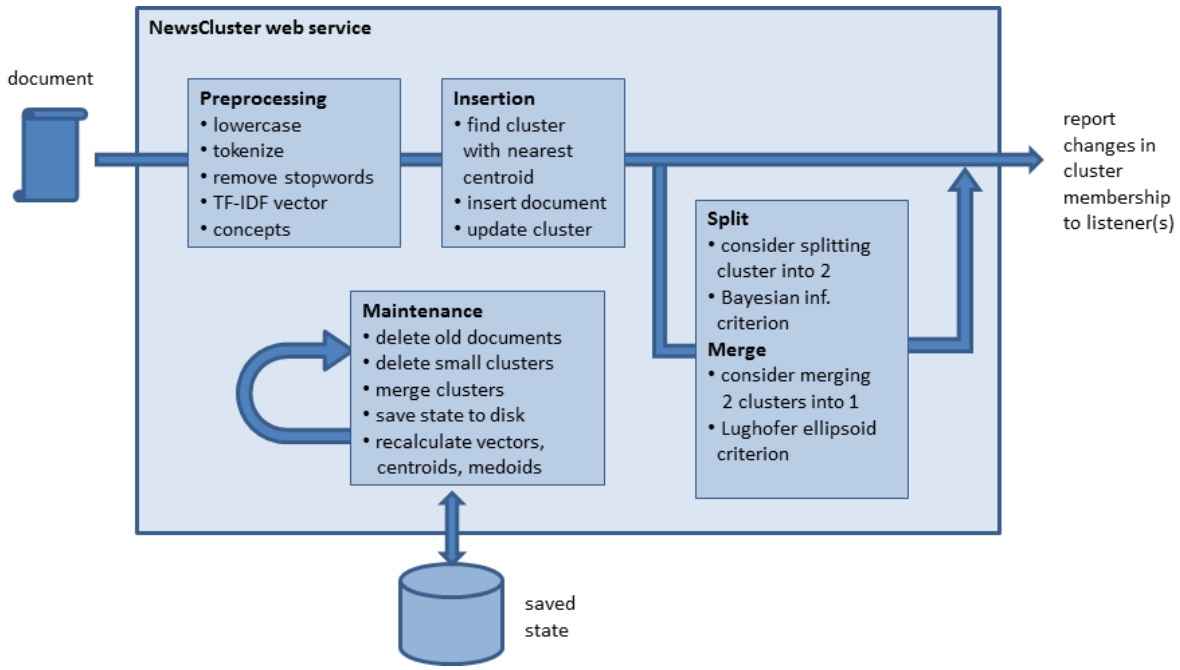

Figure 2 Architecture of the clustering service

If we look at it more closely, the incoming HTTP request is expected to provide the following information:

- a unique document ID: this can be an arbitrary string, but if NewsCluster finds that a document with the same ID already exists in its database, the new document provided by the request is ignored;
- the full text of the document;
- optionally, a title of the document, but this is used only for diagnostic purposes and does not participate in the feature construction process;
- the language of the document: NewsCluster processes documents from each language separately from those of other languages, meaning that they will not end up in the same clusters, and the

feature space (including document frequencies of features etc.) is constructed separately for each language;

- optionally, the request may also contain a set of ⟨*concept ID*, *weight*⟩ pairs; such concepts are treated as additional features in the document representation (on which see more below), on top of the features which NewsCluster obtains from the full text using the bag-of-words principle.

Document representation

For the purposes of clustering, each document is represented by a TF-IDF feature vector. First, the text of the document is split into words using a slightly adapted version of the Unicode word breaking rules [Dav12]. Next, stopwords are removed if a stopword list for the language of the document is available; currently, the NewsCluster service has stopword lists for English, German, French, Spanish, Italian, Portuguese, and Dutch.

The remaining words are used as the basis of a bag-of-words representation: the document is represented by a sparse feature vector containing one component for each word that appears anywhere in the document set so far. The value of the component corresponding to word t in the feature vector representing the document d is defined as $TFIDF(t, d) = TF(t, d) \cdot IDF(d)$, where $TF(t, d)$ is the *term frequency* (the number of occurrences of term t in the document d) and $IDF(t) = \log(N / DF(t))$ is the *inverse document frequency*, obtained from N (the total number of documents currently maintained in the collection) and $DF(d)$ (the *document frequency* of t, i.e. the number of documents in which t occurs at least once).

Optionally, the NewsCluster service can use n-grams (sequences of up to n adjacent words) as features, in addition to individual words.

If the incoming request contained the optional set of ⟨*concept ID*, *weight*⟩ pairs, a TF-IDF vector is also formed from these pairs. In this case, the *weight* is used as the TF, and a DF (and hence IDF) is computed for each concept just as it would be for features arising from the full text.

This results in two feature vectors, $\mathbf{x}^t$ arising from terms and $\mathbf{x}^c$ arising from concepts; they are concatenated into the final feature vector of the document: $\mathbf{x} = \langle \mathbf{x}^t, \mathbf{x}^c \rangle$. Each part is normalized separately such that after normalization, $||\mathbf{x}|| = 1$ and $||\mathbf{x}^c|| = w\ ||\mathbf{x}^t||$. Here, w is a user-provided parameter (*conceptWeight* on the command line) that defines the relative importance of concepts to terms in the final feature vector.

Internally, the service stores not only the normalized TF-IDF vector of each document, but also its TF vector and the original full text of the document.

Document weighting

The influence of a document in any clustering-related operation is weighted by a coefficient that decreases exponentially as the age of the document increases. Following [AY06], we use the concept of *half-life*, which is the time period in which a document's weight decreases by one-half. Thus, if t is the timestamp of a given document, the weight of this document at time T will be $(1/2)^{(T-t)/H}$, where H is the half-life period. The default value of H is one day, but this can be modified by the user through a command-line parameter. The NewsCluster service also allows the weight decay to be disabled by setting H to 0; in this case, every document always has a weight of 1, regardless of its age.

Cluster representation

Consider a cluster C containing documents $d_1, ..., d_n$, represented by their respective feature vectors $\mathbf{x}_1, ..., \mathbf{x}_n$ and weights $w_1, ..., w_n$. We maintain the following aggregate statistics for each cluster:

- the sum of weights, $W = \sum_{i=1..n} w_i$
- the sum of squares of weights, $W_2 = \sum_{i=1..n} w_i^2$
- the weighted sum of vectors, $\mathbf{S} = \sum_{i=1..n} w_i \mathbf{x}_i$
- the squared norm of $\mathbf{S}$, i.e. $||\mathbf{S}||^2 = \mathbf{S}^T\mathbf{S}$

- the weighted sum of squares, $S_2 = \sum_{i=1..n} w_i \, ||x_i||^2$ (currently, all x_i are normalized to unit length, meaning that $S_2 = W$, but in general this might change if we add more features which might not participate in the same normalization).

These aggregate statistics allow us to efficiently compute various useful quantities related to the cluster. For example, the centroid of the cluster can be computed as $\mathbf{c} = (1/W)\,\mathbf{S}$. The variance of the cluster, defined as $d = (1/W) \sum_{i=1..n} w_i \, ||\mathbf{x}_i - \mathbf{c}||^2$, can be computed efficiently as $d = S_2/W - ||\mathbf{S}||^2/W^2$. With the additional assumption that $||\mathbf{x}_i|| = 1$ for all i, we can also efficiently compute the average cosine similarity between cluster members and the centroid, defined as $a = \sum_{i=1..n} w_i \cos(\mathbf{x}_i, \mathbf{c}) / W$, where $\cos(\mathbf{x}_i, \mathbf{c}) = \mathbf{x}_i^T\mathbf{c} / (||\mathbf{x}_i|| \; ||\mathbf{c}||)$; namely, in this case it is true that $a = ||\mathbf{S}||/W$. The sum of squares of weights, W_2, is useful in the computation of the Bayesian Information Criterion (see the section on cluster splitting below).

The aggregate statistics can also be updated efficiently when a document is added to or removed from the cluster, or when the time T at which the weights are computed changes.

Clustering approach

Initially, the service starts in a "pre-clustering" state, during which it only accumulates incoming clustering without trying to assign them to any clusters. When a certain number of documents of a given language has been accumulated (controlled by a command-line parameter, defaulting to 1000), we use a hierarchical bisecting k-means (i.e. 2-means) algorithm to obtain an initial partition into clusters, as shown in the following pseudocode:

start by placing all documents into one cluster;
while the number of clusters is less than the maximum initial number of clusters do:
    choose the cluster C with the maximum variance from among the current clusters;
    use bisecting k-means to split it into two subclusters C_1 and C_2;
    if neither C_1 nor C_2 is too small, replace C with C_1 and C_2 in our current partition;

At this point, all the listeners are also informed about the initial assignments of documents to clusters. From this point onwards, the service enters its normal mode of operation. Whenever a new document arrives, a check is first performed if a document with the same title (except possibly in terms of whitespace) already exists in the collection; in this case, the new document is not added and a warning is reported instead. Otherwise, we proceed by constructing the new document's feature vector and computing the cosine similarity between this feature vector and the centroids of all the clusters; the new document is assigned into the cluster where this cosine similarity is maximized.

After the addition of a new document, the cluster is considered for splitting. The conditions for this are that it must contain a sufficient number of documents and that sufficiently many additions must have occurred since the last time it was considered for splitting. If these conditions are met, we try to split the cluster into two subclusters using bisecting k-means. We use a variant of the Bayesian Information Criterion [PM00, MG09, Lug12] to decide whether to accept the new split or not. If the cluster is split, all the listeners are notified of the new cluster memberships for all the documents affected by the split.

Background operations

In the background, the service periodically performs maintenance tasks: saving all data to disk, removing old clusters and documents, and merging clusters.

Removal of old documents is performed when the total number of documents for a given language exceeds a user-specified threshold. In this case the oldest few clusters (and all documents belonging to them) are deleted until the number of documents drops below the required threshold. For the purposes of this operation, the age of the cluster is defined to be the age of the most recent document in the cluster.

Cluster merging will be described in more detail below.

Cluster splitting

The Bayesian Information Criterion is a statistical criterion for model selection, based on the formula

$$BIC(model) = \ln Likelihood(data|model) - (\text{number of parameters in the model})/2 \cdot \ln n,$$

where n is the number of data points. Following Pelleg and Moore [PM00], we model the clusters c as spherical Gaussian distributions $N(\mathbf{m}_c, \sigma^2 I)$ with equal variance σ^2 but with different means $\mathbf{m}_c$. In our case the estimation is slightly complicated by the fact that our documents are weighted. In this section, we denote the feature vector of document i by $\mathbf{x}_i$ and the weight of this document by w_i; the dimensionality of our feature space is denoted by d; the number of clusters is k; we further write $W_c = \sum_{i \in c} w_i$ for the total weight of cluster c and $W = \sum_c W_c$ for the total weight of all documents. The mean of each cluster is estimated by its weighted centroid, $\mathbf{m}_c = \sum_{i \in c} w_i \mathbf{x}_i / W_c$. For an unbiased estimate of σ^2, we can use

$$S^2 = (\sum_c \sum_{i \in c} w_i \, ||\mathbf{x}_i - \mathbf{m}_c||^2) / \sum_c (W_c - \sum_{i \in c} w_i^2 / W_c).$$

The sum $\sum_{i \in c} w_i \, ||\mathbf{x}_i - \mathbf{m}_c||^2$ can be expressed as $\sum_{i \in c} w_i \, ||\mathbf{x}_i||^2 / W_c - ||\mathbf{m}_c||^2$, which allows us to compute S^2 as well as all the centroids efficiently from the aggregate statistics that we maintain for each cluster (see the Cluster representation section above).

The log-likelihood component of the BIC can now be expressed as

$$\ln Likelihood(data|model) = \sum_c \sum_{i \in c} w_i \ln (p(\mathbf{x}_i \mid c) P(c)),$$

where $P(c) = W_c / W$ and $p(\mathbf{x}_i \mid c) = (2\pi \sigma^2)^{-d/2} \exp(-||\mathbf{x}_i - \mathbf{m}_c||^2 / 2\sigma^2)$. Plugging these into the log-likelihood formula, and taking into account the definition of S^2 above, gives us

$$\ln Likelihood(data|model) = \sum_c [(-d/2) \ln(2\pi \sigma^2) + W_c \ln W_c - W_c \ln W - ½ (W_c - \sum_{i \in c} w_i^2/W_c)].$$

Thus, the contribution of each cluster c to the log-likelihood component of the BIC can also be computed efficiently from the aggregate statistics that we keep for each cluster.

For the number of parameters in the second component of the BIC, we have $k - 1$ cluster probabilities (down from k due to the constraint that they sum up to 1), $k \cdot d$ centroid coordinates, plus 1 for the variance σ^2. The $\ln n$ factor from the definition of BIC can be replaced by $\ln W$ to take the weighted nature of our documents into account. Thus, the overall BIC can be written as

$$BIC(model) = \sum_c BIC_c ,$$

where BIC_c is the per-cluster BIC:

$$BIC_c = (-d/2) \ln(2\pi \sigma^2) + W_c \ln W_c - W_c \ln W - ½ (W_c - \sum_{i \in c} w_i^2/W_c) - (d + 1) \ln W.$$

When evaluating a possible split of the cluster c into two subclusters c' and c'', we have to simply compute BIC_c, $BIC_{c'}$ and $BIC_{c''}$, and accept the split if $BIC_{c'} + BIC_{c''} > BIC_c$.

Example of a successful split

The original cluster c had 158 documents with a variance of 0.44; $BIC_c = -8.7e6$. The most highly weighted terms in the centroid $\mathbf{m}_c$ were: `BlackBerry`:0.686 `BBM`:0.372 `Q5`:0.195 `Heins`:0.159 `Android`:0.157 `Q10`:0.140 `iOS`:0.133 `users`:0.124 `messaging`:0.124 `$concept$_38150`:0.108 `smartphone`:0.087. (Note that the "`$concept$_38150`" pseudo-term is one of the features based on concepts that can be provided in addition to full text of incoming articles. From the clustering service's point of view, these concepts are opaque and known only by their unique identifier.) The medoid (article nearest to the centroid) was a document titled "Blackberry Messenger Will be Free iOS and Android App". As it turns out, the cluster was actually a mixture of two Blackberry-related topics. The proposed subclusters were:
— Subcluster c' had 79 documents with a variance of 0.42; $BIC_{c'} = -4.1e6$. The most heavily weighted terms in the centroid $\mathbf{m}_{c'}$ were: `BlackBerry`:0.665 `Q5`:0.338 `Q10`:0.257 `Heins`:0.206 `smartphone`:0.128 `Z10`:0.118 `markets`:0.112 `keyboard`:0.109 `BlackBerry's`:0.106 `RIM's`:0.101 `device`:0.089 `RIM`:0.082. The medoid was a document titled "BlackBerry announces budget Q5; Z10 gets BB10.1 update", and there was a tight cluster of similar documents around it as well (with titles such as "BlackBerry Q5 Hands On: A Budget-Friendly BlackBerry for the Masses", "BlackBerry CEO unveils 'slim, sleek' new version", etc.).

— Subcluster c'' contained the remaining 79 documents with a variance of 0.39 and $BIC_{c''} = -3.8e6$. The most heavily weighted terms in the centroid $\mathbf{m}_{c''}$ were `BlackBerry`:0.585 BBM:0.548 `Android`:0.205 `iOS`:0.200 `messaging`:0.167 `users`:0.160 `$concept$_38150`:0.113 `service`:0.105 `Messenger`:0.094 `Heins`:0.093 `app`:0.085. The medoid was the same as in the original cluster c, i.e. "Blackberry Messenger Will be Free iOS and Android App".
Since $BIC_{c'} + BIC_{c''} = -7.9e6$ was greater than the old $BIC_c = -8.7e6$, the split was accepted. The result of the split was that one of the new subclusters, c', contained stories about the launch of the new and cheaper BlackBerry Q5 smartphone, while the remaining cluster c'' now consisted almost entirely of stories about the release of the BlackBerry Messenger app for iOS and Android systems. This has definitely improved the compactness of both clusters. Cluster c' still contains a few documents that are not about its main topic (they are still BlackBerry-related news stories, but not about the Q5 launch); this is mainly due to the fact that there aren't enough such documents to be split off to a separate cluster of their own (the minimum cluster size constraint had been set to 50 documents during that particular test run), though this could happen in the future if more stories on those topics arrive.

Cluster merging

Periodically (currently this is done once per 15 minutes), the NewsCluster service looks for pairs of very similar clusters and considers merging them. First, we compute the cosine similarity between all pairs of cluster centroids. The pairs where this similarity was highest are then considered for merging, in decreasing order of similarity. (Currently, the number of pairs to be considered is limited to 3 times the total number of clusters.)

For each such pair of clusters, we compute various statistical properties of the clusters in their current state as well as of the new cluster that would be the result if these two clusters were merged. The merge is performed if at least one of the following criteria is met:

Cosine similarity criterion: the clusters are merged if the cosine between their centroids is greater than a user-specified threshold.
Lughofer's ellipsoid criterion: following [Lug12], we examine the feature vectors of all the documents belonging to cluster a, and compute $\mathbf{c}^a_i$ as the average i-coordinate of all these vectors and s^a_i as the standard deviation; the same is done for cluster b. We can now think of each cluster as being approximately an ellipsoid whose center is in $\mathbf{c}^a$ (or $\mathbf{c}^b$, respectively) and whose radii are given by $\mathbf{s}^a$ (or $\mathbf{s}^b$, respectively). The merge is performed if the ellipsoids overlap sufficiently, i.e. if

$$||\mathbf{c}^a|| - ||\mathbf{c}^b|| \leq f \sum_i |c^a_i - c^b_i| \; (s^a_i + s^b_i) \,/\, \sum_i |c^a_i - c^b_i|,$$

where f is a user-specified parameter (the higher it is, the more merges will be performed).

Web interface

To insert a new document, a HTTP POST request should be made to the NewsCluster web service, at the URL http://*server*:*port*/add-article. The body of the HTTP request should contain (argument name, argument value) pairs in URL-encoded form, e.g.

id=12345&lang=eng&text=Lorem+ipsum+dolor+sit+amet

The following arguments are required: id (giving a unique identifier of the document; if a document with the same identifier already exists, the new document is ignored); lang (an ISO-639 three-letter code identifying the language of the document); title (optional, giving the title of the document); and text (containing the actual contents of the document itself).

Optionally, a set of ⟨*concept ID*, *weight*⟩ pairs may be provided. If we denote the i-th pair by $\langle c_i, w_i \rangle$, the pairs should be given by two parameters in the request body:

&conceptIds=c_1,c_2,...,c_n&conceptWgts=w_1,w_2,...,w_n

The concept IDs may be arbitrary strings (not containing special characters such as commas etc.) and the weights may be any floating-point numbers.

Return value: the body of the HTTP request contains a JSON value of the form

{"ClusterId":"cluster identifier"}

where the *cluster identifier* is a string that globally uniquely identifies the cluster into which the new document has been placed. Additionally, the NewsCluster service will send the new (article ID, cluster ID) pair to any listener(s) whose URLs have been passed in the command-line parameters.

Other commands supported by the NewsCluster web service are:

http://server:port/save - forces the service to immediately save its state to disk

http://server:port/exit - causes the service to save its state to disk and then terminate

http://server:port/report - returns an HTTP response containing an HTML report on the current state of the clusters. The optional parameter *?centroids=1* will cause cluster centroids to be included. The optional parameter *?lang=language&clusterId=number* will generate a report on the cluster whose internal number is given by the *number* parameter; this report includes the list of documents in the cluster. Note that the reports returned by the report command are not intended to be machine-readable, but to be human-readable for debugging and informational purposes.

2.4 Cross-lingual cluster matching

Clustering service is running separately for each of the article languages. As a result, each cluster only contains articles from a single language. Since we know that articles in different languages can still discuss the same event we needed to create methods for identifying separate clusters that in different languages talk about the same event.

In order to identify clusters which need to be merged we used information from the Cross-Lingual Similarity Service (CLSS) which for each article identifies 10 most similar articles in English, German, Spanish and Chinese language. Intuition behind the developed algorithm is that if many articles from a cluster have most similar articles that belong to a single cluster then the clusters are likely about the same event. The algorithm is shown in **Error! Reference source not found.Error! Reference source not found.** and works as follows.

Algorithm: identifying clusters in different languages that discuss the same event
Input: clustering C containing N=|C| clusters from all languages
cluster c_{test} (containing M articles) that we wish to consider merging with other clusters
Output: list sameEvent that contains clusters discussing the same event as cluster c_{test}
Methods: topSim(a, l) returns most similar articles for article a in language l
sim(a_i, a_j) return similarity between articles a_i and a_j as computed by CLSS service
clust(a) returns id of the cluster to which article a is assigned to
Init: count[c] := 0 for c=0..N
sim[c] := 0 for c=0..N
sameEvent := [c_a]
for each article a_i in c_{test} (i=1..M):
for each language l in L:
for each a_j in topSim(a_i, l):
count[clust(a_j)] += 1
sim[clust(a_j)] += sim(a_i, a_j)
for each c in C:
avgSim := sim[c] / count[c];
if (avgSim > simTresh & count[c] > ratioTresh*10*M)
sameEvent.append(c)

Figure 3 Algorithm for merging clusters describing the same event

Given a cluster of articles c_{test} we wish to find other clusters c that also describe the same event and merge them. We start by initializing two dictionaries – count and sim – that will for each cluster c store relevant information about similarity between clusters c and c_{test}. For each article in the test cluster c_{test} we then identify the most similar articles in all other languages. If test cluster contains M articles then we should for each language get 10*M most similar articles. Each of these most similar articles is assigned to one cluster and our goal is to determine if some clusters occur very frequently among these articles. For each similar article we therefore update the count and sim dictionaries. In the count dictionary we simply increment by 1 the value for the id of the cluster that similar article is assigned to. For the sim dictionary we use the computed similarity between the article in the test cluster and the similar article (similarity value is provided by the CLSS service). The similarity value is again used to increase the existing value for the cluster that similar article is assigned to.

After going through all the articles in the cluster we check how frequently each cluster c occurred among the most similar articles. We use the frequency of occurrence (the count[c] value) as well as the average similarity between the articles (computed from sim[c] and count[c]) to determine if the clusters should be merged. Currently used values for threshold parameters are simTresh = 0.6 and ratioTresh = 0.2. Value of parameter ratioTresh requires that at least 20% of all most similar articles in a single language point to the same cluster in order to merge with it. The current values of parameters were chosen experimentally and will be in the future work a subject of more thorough evaluation.

The result of the described algorithm is a sameEvent list, which contains clusters that discuss the same event as cluster c_{test}. These clusters can be in the following steps merged and represented as a single event.

2.5 Event formation and event information extraction

At this point in the pipeline we have already identified groups of articles that describe the same event as well as potentially found multiple clusters in different languages about the same event. The next step is for us to create an actual event using the clusters and to extract from the articles structured information about the event.

In order to represent an event we have created a data structure that points to one or more clusters, and clusters are the data structures that point to individual articles (see UML diagram on Figure 4 for details). One could argue that semantically the clusters are somewhat redundant – that articles should simply be assigned directly to an event. While this is true, we find that keeping the clusters significantly simplifies the event maintenance. To each event that we create we assign a unique id. It's a number that increases monotonically with each event and for a given event always stays the same.

Events are a very dynamic structures which evolve a lot – at least when events are new and there are new articles coming into the system that write about them. Each time a new article is received it is assigned to a cluster. Due to online approach to clustering, the clusters can change significantly over time – they can merge or split into two clusters and individual articles can be reassigned to a different cluster. Each of these changes needs to be propagated also on the level of events. Event updating is done as follows. In case a single cluster c is split into two clusters (c_a and c_b), the larger cluster stays assigned to the same event, while the smaller cluster is assigned to a new event. The cross-lingual cluster matching has to be recomputed again for clusters c_a and c_b in order to take into account the changes. Alternatively, two clusters can also merge if the clustering method determines that there is sufficient similarity between them. In this case we have two existing clusters that are already assigned to two events. In such instances we take the articles from the smaller cluster and assign them to the larger cluster. What can happen is that the event, where the smaller cluster was, now has no more clusters assigned to it and consequently also no articles. We mark such an event as invalid and remove all the extracted information associated with it. Changes to an event can occur also when individual articles are simply assigned or re-assigned to an existing cluster. Changing the number of articles in a cluster can increase/decrease the support for the cross-lingual cluster matching. To keep the corresponding events up-to-date we re-run the cross-lingual cluster matching procedure each time 5 changes are made to a cluster (either by assigning new article to it or by re-assigning an existing article to a different cluster). Similarly, all

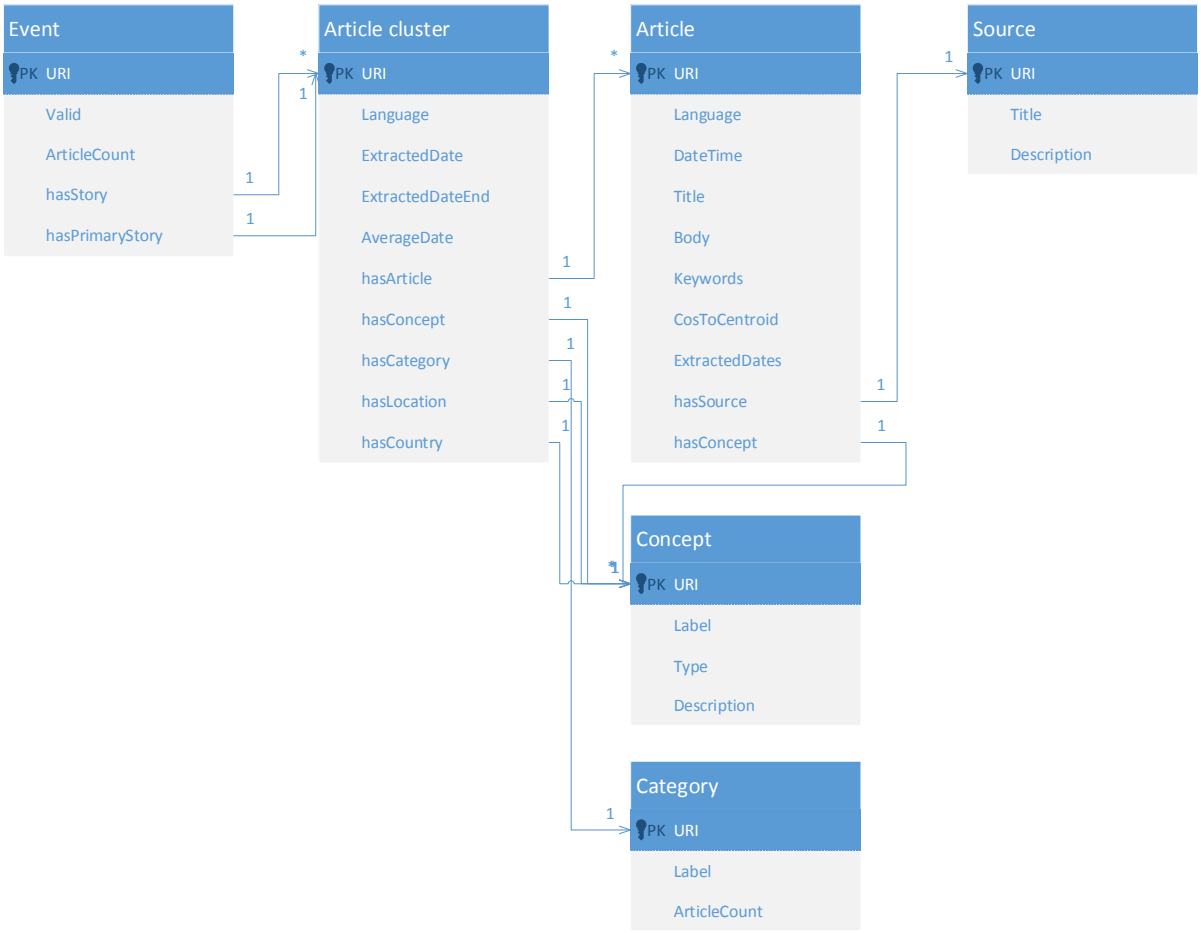

Figure 4 UML diagram of data structures involved in the event registry

event properties (described in the next paragraphs) are also re-computed whenever 5 changes are made on a cluster.

Event title and text snippet

Event title is one of the basic properties that needs to be extracted for each event. Event title should be a short text that summarizes what the event is about. If the event is covered in different languages then the title needs to be determined for all corresponding languages. To set the title(s) of the event we used the clusters of articles assigned to the event. For each of the clusters we identify the centroid article of the cluster and use its title as the title of the event for the particular language. The reason for using the centroid article is that the article is the closest to the center of the cluster and therefore seems to be best fitted to describe the event in that language. The title of the article is used as the title of the event since it is short and should summarize the event content. In the future work we plan to work on finding alternative titles based on analysing commonalities in article titles for all articles in the cluster.

We also wish to identify a text snippet that would in a few sentences describe what the event is about. For this purpose we also rely on analysing individual clusters of articles and identifying the centroid article. From the centroid article we then use the first paragraph as the text snippet for the particular language. Similarly as for the event title we also plan as a part of the future work to analyse different summarization methods to identify a better way for identifying the event text snippet.

Extracting event date

Most events have a particular date that can be associated with the event. This date can be a single day or it can be a time period with a starting and ending date.

In order to determine the event date we use the information about the date references that we extracted from individual articles in the pre-processing phase. Our approach iterates over all articles assigned to an event and collects the identified date references. Before we start counting how frequently each date reference occurs in different articles we also try to take into account the dates with incomplete information. As we mentioned in Section 2.2, some articles provide incomplete date information (such as “September 11” where the year information is missing). In order to still use such information we try to impute the missing values using the complete dates from other articles in the event. If, for example, we find that in these articles the date references for September 11 are mostly associated with year 2001, we also assign this year to the incomplete references. Once data imputation is finished we count the number of times each date is mentioned in the articles and choose the most frequent one. We want to make sure the date is mentioned significantly enough in order to use it as the event date so we set a threshold – the date has to occur at least in 30% of all articles assigned to an event. Although we presented the method as if we were only analysing the single day occurrences, we do in the same way analyse also the detected time periods. If the most mentioned time period is more frequent than the most mentioned single date then we use the time period as the starting and ending date of the event.

Some events don’t contain any date references or the ones mentioned are not frequent enough to be considered as reliable. In such cases we decided to rely on the dates when the articles were published. We could assume that an event is only being reported after it happens so in principle we could use the date of the first article as the date of the event. Our early experiments indicated that setting event date based on a single article can be problematic. Clusters sometimes contain incorrectly assigned articles and taking the earliest article date can set too early date for the event. To be more conservative we decided to simply compute the average date of the articles for the event and use that date as the event date. Using this date is also not ideal and mostly sets the event to a later date than it should – it is however more reliable since it is computed based on several articles. In future we will try to observe the distribution of article dates and try to identify a tipping date at which the number of articles started to significantly increase. Using this tipping date as the event date should be a reliable as well as more accurate estimate of the event date.

Extracting event location

Many events, such as meetings, sport events, natural catastrophes, bombing attacks etc., are associated with geographical location. The location can be a city, area or a whole country. Knowing the location is semantically important part of the event so we want to identify it from the articles.

In order to determine the event location we use the article annotations provided by the text annotation prototype [D3.1.1] and the word-sense disambiguation prototype [D3.2.1]. The provided annotations contain entities as well as relevant keywords and topics detected in the articles and from these we need to identify the mentions of geographic locations.

In order to build a database of possible geographic locations we used the GeoNames database [Geo] which contains over eight million place names. From all the available locations we used those that have a Wikipedia page – since the annotation service uses Wikipedia for disambiguation, these are the only locations that the service can identify in the text. From GeoNames we also used the information about the location labels in different XLike languages, as well as the countries and continents in which the locations are. The country information is, for example, necessary if we later wish to search for events not by a particular location but by the name of the country.

Once we were aware of all possible geographic locations we analysed the event articles and identified the annotations representing the geographic locations. From all geographic references we selected the one that is mentioned most frequently and treat is as a candidate for event location. In order to assure that the location is not mentioned just by coincidence we require that it has to be identified in at least 30% of all

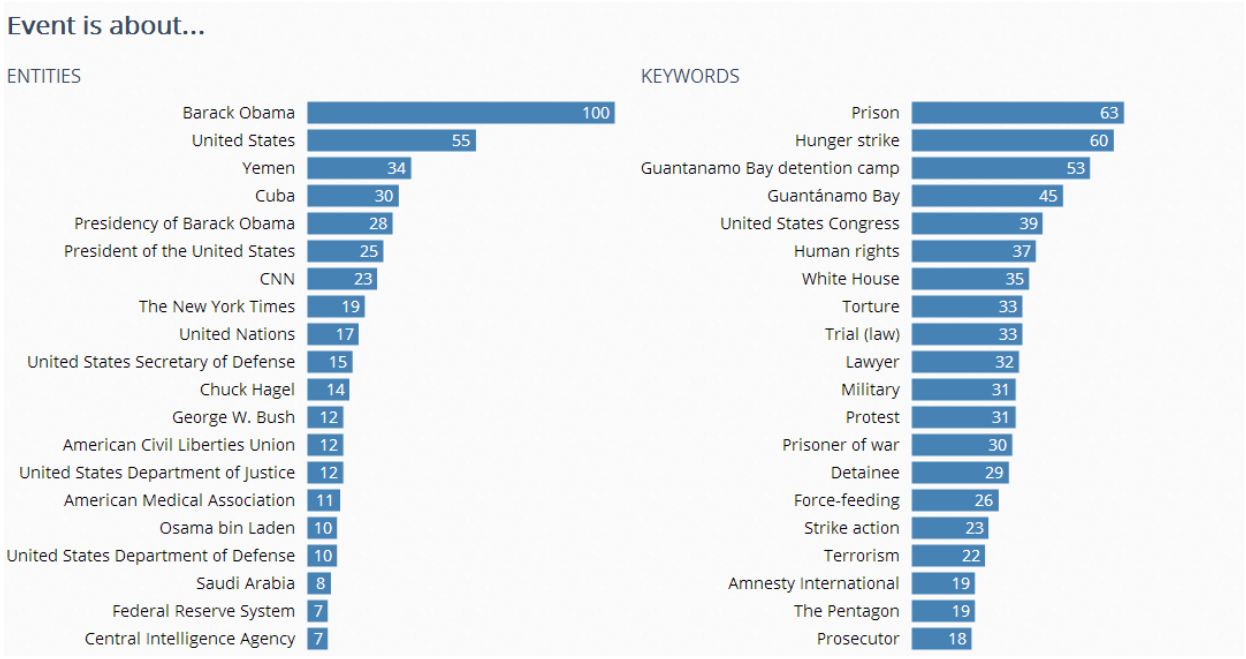

Figure 5 Example of top concepts for event "Hunger strike at Guantanamo enters 100th day".

event articles in order to consider it as the event location. If not location passes this threshold then we conclude that the event is not associated with a particular location.

Extracting event entities and keywords

Each event discusses one or more keywords and mentions a number of entities – people, organizations and locations. These entities and keywords (we will call them concepts) are crucial for understanding what the event is about and we wish to identify them and assign them a score of relevance for the particular event.

For extracting event concepts we again rely on the annotations provided by the text annotation and disambiguation services. We start by identifying the concepts that were annotated in articles that are assigned to the event. Beside the URI, each concept is also assigned a weight that corresponds to the relevance of the concept for the article. The weight is computed based on how frequently the concept appeared in the article. If, for example, Barack Obama is frequently mentioned in the article, then he will be assigned a high weight. The weights are currently integer numbers ranging from 1 to 5. By going over all articles in the event we compute the sum weight for each of the concepts identified in the articles. Concepts that occur frequently in the articles with high weights will get a high total weight, while concepts which occur rarely and with lower weights will obtain a low total weight. Next, we rank the concepts by their total weight and ignore the concepts with total weight lower than M*5*0.1, where M is the number of articles assigned to the event. The threshold condition states that a concept is relevant only if it occurs at least in 10% of articles with a weight of 5, or correspondingly higher percent of the articles with a lower weight. By ignoring the concepts with low weight we are in essence removing the noise, which can occur due to incorrectly assigned articles or spurious annotations. The concepts that are above the threshold are used as the event entities and topics. Along with the concepts themselves we also use the total weights for the concepts since they are a good estimate of what is more/less relevant for the event. The total weights are then normalized to the interval [0, 100] and can be seen when viewing the event info. An example of a possible visualization of top concepts for event "Hunger strike at Guantanamo enters 100th day" is shown in Figure 5.

As it can be seen from the Figure 5 we have separate columns for entities and keywords. Entities consist of people, locations and organizations. Keywords, on the other hand, are words that were annotated, but don't represent entities. Separation between entities and keywords is informative since each category "explains" a different part of the event – entities describe the "who" of the event, while the keywords describe the "what".

In order to separate between entities and keywords we need to know for each concept URI also its type. The types of the concepts are already provided by the word-sense disambiguation service [D3.2.1] but we found the accuracy of the service to be inadequate and insufficient for use. To improve the accuracy we built a parser for Wikipedia's info boxes and determined the concept type based on the relations provided in the info boxes. We determined relations that are characteristic for individual entity types (person, location, organization) and used them to separate entities from keywords. Relations that were, for example, used determine that a concept is a person were birth_date, death_date, spouse or children. Similar relations were identified also for locations and organizations. If a concept was not identified as one of the entity types then it was determined to be a keyword.

Event categorization

Events can also be categorized by their types. Because there can be a huge number of possible event topics it makes sense to organize them into a taxonomy. An example of a small part of such taxonomy could be a root type "Sports", a sub-type "Athletics" with another sub-type "Triple jump". Another example could be a root type "Natural disasters" with a sub-type "Earthquake".

There is no single event type taxonomy that would be *the* taxonomy. Most news publisher uses one but they are considered to be a company secret and are not shared with the outside world. These taxonomies would likely be in most part overlapping although we can expect that different publishers have emphasis on different areas which means that some parts of the taxonomy are expected to be more/less granular.

Since we didn't have access to any taxonomy of events we had to find a publicly accessible taxonomy that would suit our needs. The taxonomy that we chose to use is the DMoz [DMoz] taxonomy. DMoz is an open directory project that has more than 5 million web pages categorized into a taxonomy with over 1 million categories. This number of categories was too large so we decided to only use the categories up to three levels deep. The number of categories we obtained in this way was still over 1000 which seems sufficient for our needs.

An even more important part than the taxonomy itself is for us the fact that nodes in the taxonomy also have a list of web pages that are associated with the particular category. Since the web pages are discussing the topic represented by the category we can learn from them about the characteristic features of each category. We do this by obtaining the content of the web pages, transforming them into the bag-of-words format and weighting the words using the TF-IDF weighting scheme. The pages for the category and its sub-categories can be considered as a cluster in high dimensional space and we can identify the vector representing the center of the cluster. By identifying such vectors for each category we are able to obtain compact representations for the categories. If we then have a new document that we would like to classify into one of the categories we can similarly compute the BOW format of the document, weight the words with TF-IDF weighting and use the cosine similarity to compare the document vector with the vectors of individual categories. The category with the highest cosine similarity can be chosen as the category of the document.

For categorization of events we used the DMoz classifier in the following way. For each cluster in the event we identified 5 (or less, if cluster was smaller) articles that are closest to the center of the cluster. We merged the content of these articles into a single document, computed the BOW format for the document and weighted the words with TF-IDF weighting. We identified the category with the most similar vector and assigned it to the cluster. It can be surprising that the category is actually assigned to the cluster and not the event itself, but there are good reasons for this decision. DMoz classification has to be done for each cluster separately since the clusters are built separately for different languages. Having multiple clusters for an event it is inevitable that there would be disagreements between the computed categories and there is no clear solution how to solve it. Additionally, knowing the categories of individual clusters can also be helpful in our future work for deciding whether merging two cross-lingual clusters into an event was correct or not.

2.6 Identifying related events

World events are not separate islands – events are interconnected, they discuss similar topics or are a part of a chain of connected events. There are different criteria that we can use to identify related events. In principle we can determine event similarity using any of the event properties:

- Time: using a specific time window can identify very diverse events. Since their only commonality is time, the criteria by itself won't identify strongly related events.
- Location: location can be an important feature for identifying geographically related events, especially in combination with time. It can identify events that touch various topics, which are relevant for a particular region. A good example would be finding events occurring in Syria in the last months.
- Entities and keywords (concepts): By comparing the "what" and "who" of the events we can identify events that share similar actors or topics discussed in the events. Using such criteria could identify as similar events, such as "Google releasing Android 4.3" and "Google and LG offering Nexus 4".
- Category: comparing events based on their type is somewhat related to the comparison using entities and keywords. A good example of using the criteria for finding similar events would be looking at an earthquake event and finding other examples of earthquakes.

Most of the described conditions offer a straightforward way for comparing the events. An exception is perhaps the entities and keywords condition since each event is associated with a weighted list of concepts and it is not obvious how we can compare two such lists. What we decided to do is to use the methodology used when comparing documents. Each concept associated with the event is considered to be a document term and its weight represents the term frequency for the document. Having the bag-of-words representation of the events we also want to apply TF-IDF weighting to take into account that some concepts are much more popular than others. After the vectors are weighted we can use any similarity measure (we opted for cosine similarity) to compute concept similarity between events.

2.7 Event registry

Once the event is identified and processed we store all available information in an event registry. Simply put, an event registry is a database of all recognized events. All relevant event parameters are explicitly stored and indexed so that an efficient search can be provided across all the events. The information about the events is updated regularly when the event changes. Since the event extraction is running on an online stream of news articles, the changes of the underlying data structures are very frequent. In order to reduce the work load we only update the event properties after we make 5 changes to any of its clusters (adding or removing an article to the cluster) or when clusters are merged or split. This sometimes causes some events to be slightly outdated but on the other hand keeps the event registry responsive to the API calls.

3 Event registry API calls and user interface

Event registry provides a set of API calls that can be used to query and edit the information about the events. In this section we will describe the main calls, its parameters and the output provided by the service. All the event related calls have the following format:

serviceUrl/event?action=eventAction&list-of-parameters-and-their-values

The eventAction parameter determines the type of the request. The types that we will describe here are getEvents, getEventInfo, getEventArticles, getSimilarEvents and getTrendingEvents. Examples of all API calls and the returned data are listed in Appendix A.

3.1 getEvents API call

The getEvents call is the most important call and is used to query the event registry for events based on several possible search conditions. The relevant parameters and their descriptions are listed in Table 1.

Table 1 Parameters relevant for the getEvents API call

Parameter	Example value(s)	Description
conceptUri	pm_person: http://en.wikipedia.org/wiki/Barack_Obama	[multiple occurrences] One or more concepts that the event is about. If event is not annotated with this concept it will not be included among the results.
timeStart	2013-05-13	[single occurrence] starting date of the time window of interest. Events outside of this window will not be returned as results.
timeEnd	2013-05-20	[single occurrence] ending date of the time window of interest.
lang	eng | deu | spa | zho |slv	[multiple occurrences] one or more languages that should report about the event. If more languages are specified then the event should be reported in at least one of the languages (operator OR is used between languages, not AND)
locationUri	pm_location:http://en.wikipedia.org/wiki/Ann_Arbor,_Michigan	[single occurrence] a location URI that either represents a city or a country. Events that occurred in this city/country will be returned as results.
categoryUri	http://dmoz.com/Business	[multiple occurrences] one or more URIs that represent the event type. Events that belong to the particular type or its sub-type will be included in the results.
minArticlesInEvent	100	[single occurrence] Minimum number of articles that cover the event.
maxArticlesInEvent	500	[single occurrence] Maximum number of articles that cover the event.
publisherUri	http://www.independent.ie/sport/	[multiple occurrences] The URI of the publisher that has published an article about the event. If a publisher hasn't published any article on the event then the event will not be included in the results.

resultType	list | timeAggr | locAggr | conceptAggr | trendingConcepts | docAtlasOnConcepts	[mandatory field, single occurence] The type of the results we wish to obtain. Possible values are: - list. Returns the list of matching events with the basic information about the events. - timeAggr. Returns the distribution of resulting events along time. Used to show the timeline of events. - locAggr. Returns the distribution of resulting events in different geographical locations. Used to show the map of event locations. - conceptAggr. Returns the list of entities that most frequently appear in the events. Also includes a list of entity pairs that most frequently co-occur in the events. - trendingConcepts. Returns the trending of the most popular concepts for the events. - docAtlasOnConcepts. Generates a visualization where events are plotted in 2D space and positioned based on the similarity of their annotations.

3.2 getEventInfo API call

The getEventInfo API call is used to obtain information about a particular event. It expects a single additional parameter which is *eventId*. Given that the provided id is valid, the return value contains the event information, which includes all the extracted information about the event except the actual articles assigned to the event.

3.3 getEventArticles API call

getEventArticles call returns the articles, which are assigned to the event specified in the *eventId* parameter. Table 2 contains the list of possible arguments that can be specified in the request.

Table 2 Parameters for the getEventArticles API call

Parameter	Example value(s)	Description
lang	eng | deu | spa | zho |slv	The language for which we wish to obtain the articles
page	(any number >= 0)	page of the articles
count	(any number > 0)	Number of articles per page to return
sortBy	rel | time	The order in which the returned articles are sorted. Sorting by time orders articles from the newest to the oldest. Sorting by relevance (rel) orders the articles based on how central they are to the event. The value is computed based on the distance of the article from the center its cluster. Ordering by relevance is useful because the articles, which are far from the center, are possibly incorrectly assigned to the cluster and we wish to see them as late as possible.

3.4 getSimilarEvents API call

getSimilarEvents call is used to obtain a list of events that are most similar to the event specified in the *eventId* parameter. Similarity of the events is computed based on the annotated concepts assigned to the events. Two optional parameters are *count*, which sets the number of returned similar events, and *addDetails* (possible values 0 or 1) that determines the amount of details returned for each similar event.

3.5 getTrendingEvents API call

getTrendingEvents call returns the list of currently top trending events. The required parameter is *lang*, which determines the language of the trending events. Additionally we can also specify *count* parameter, that sets the number of top trending events returned, and *addDetails* (possible values are 0 or 1) that determines the amount of details returned for each trending event.

3.6 User interface for event registry

In order to provide an easy access to event registry we've also developed a user interface for the service. The user interface currently provides a search interface that can be used to find the desired events based on all available search criteria. Figure 6 shows a screenshot of events that were found to be related to Barack Obama. The interface allows the user to see the main information about the resulting events as well as two visualizations with a summary of the resulting events – a geographic map of events and the timeline view. Clicking an event opens a new window containing more detailed information about the event as well as the list of actual articles reporting about it.

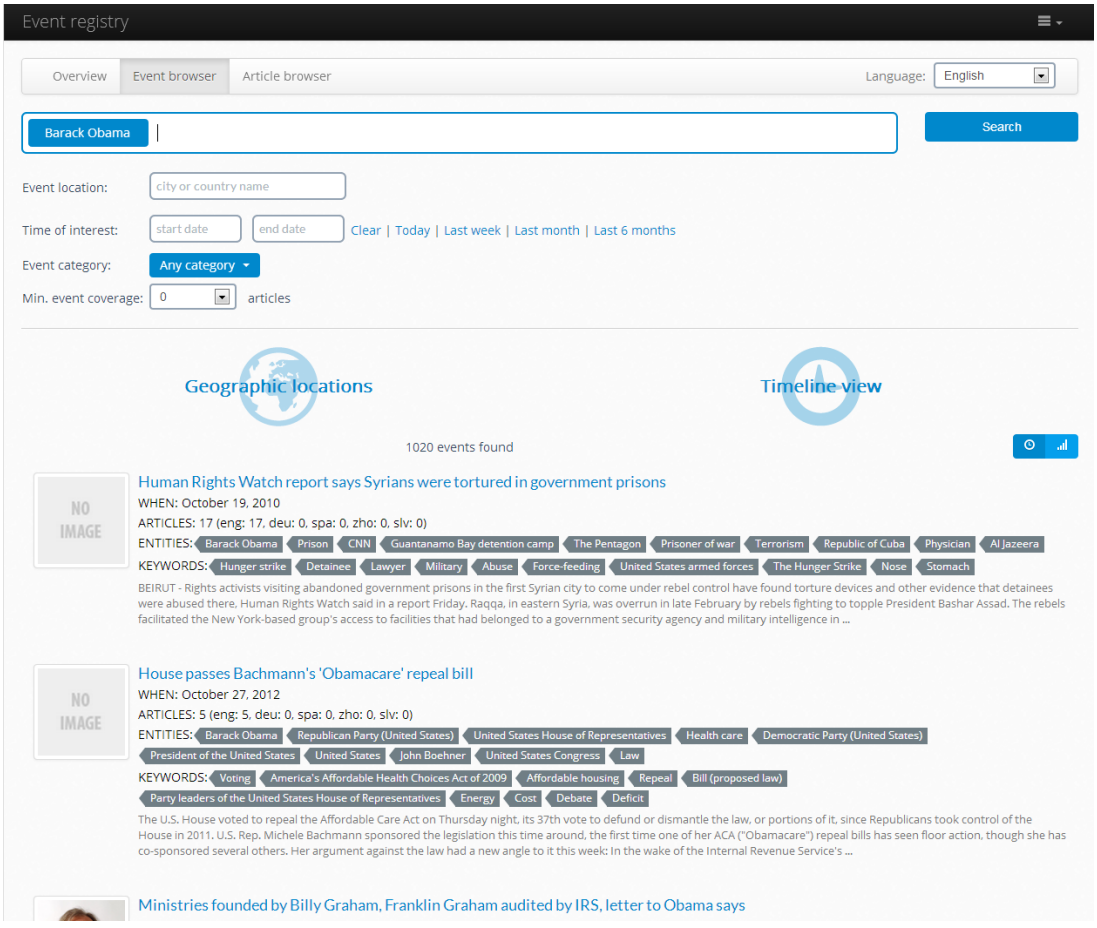

Figure 6 Screenshot of the event browser user interface

4 Future work

Although a lot of effort was already put into the event detection and information extraction, we still have a lot of ideas for new features and improvements. Here we will briefly mention individual parts of the pipeline and their possibilities for improvements.

The existing clustering method already takes into account article text as well as the annotated concepts identified in the article. Semantic role labelling, that is currently being added as a part of the information extracted from the articles, could bring additional relevant features, that could be used to achieve even better clustering of articles.

The process of merging of clusters from different languages currently only takes into account cross-lingual article similarity provided by canonical correlation analysis. When merging the clusters we plan to also include other language independent features, such as detected named entities, category of the cluster, etc. By using multiple sources of information we are less likely to make mistakes when joining the clusters.

The event categorization we currently use relies on DMoz categories. These categories were built to organize web pages and are not a very good match for the task of event categorization. In the future we will try to find a news publisher who would be willing to share their own taxonomy of event types and use it instead of DMoz. Along with the taxonomy we will also need to obtain examples of these event types. These examples will serve as learning examples when we will be building a classifier that will be able to classify events into the new taxonomy.

We also plan to use the event taxonomy for obtaining additional semantic information about the events. For each event type we will try to identify commonalities among events that are characteristic for the event type. For earthquake event type, for example, we can expect that all instances of the event contain the information about its location, the magnitude of the earthquake, its duration and potentially the number of victims. Similarly, for the football match type, the events should contain the names of the teams playing, the location, and the final score. For most event types we hope to be able to identify specific semantic frames that can be learned from instances of the event type. Using these semantic frames we should be able to obtain additional information about the event that is specific to the event type.

The relations between the events are very important for finding related information. Currently we haven't yet put a lot of effort into identifying different kinds of possible relations between events. In the future, we aim to identify at least three different types of relations between events – weakly related events, hierarchical events and events that form a storyline. Weakly related events are events that share some common entities and topics, but are not necessarily close in time. Example of such events would be "Google releasing Android 4.3" and "Google and LG offering Nexus 4". Storylines are formed from events that are relatively close in time and share common entities. Example of a storyline would be events like "Bombing in Syria" and "USA planning an attack on Syria" that could be summarized with a storyline "Unrest in Syria". The third type of event relations are hierarchical events. Such events are, for example, different sports events at the Summer Olympics 2012. In this case, "100 meters run" and "Triple jump" could be represented as a meta-event "Athletic events". By looking at the differences between different types of relations we will try to identify particular characteristics of each type and define methods to automatically identify these relations between different events.

On the side of the user interface we also plan several improvements. An important feature that we want to offer is the ability to edit the event information. It is inevitable that some events or their extracted information will be incorrect and we want to be able to offer the ability to fix the errors. We also plan to implement methods from active learning that will be able to identify events that most likely contain incorrect information in order to make event editing more efficient. Since we plan to offer the event registry to the publishers we also want to allow them to publish in the registry information about the future events. An important part of the future frontend work is also identification of different ways in which the information about the events can be visualized and summarized so that the user doesn't have to go manually through a list of possibly thousands of search results.

Our goal is also to expose the event data as a semantic resource and connect it with other LOD resources.

5 Conclusion

This document presented the methodology we developed in order to identify events from the news articles and extract from the articles the relevant event information.

In order to extract events we use the news articles that are processed by the XLike pipeline. The pipeline includes part-of-speech tagging, named entity detection, anaphora resolution and named entity disambiguation. For the purpose of the event extraction we also implemented detection of date references and detection of article duplicates. Once the articles are processed by the pipeline we use a clustering algorithm to group the articles based on their content similarity and the similarity of detected named entities. The clustering algorithm works online (clusters are updated each time a new article arrives) which means that clusters are frequently changed. Each identified clusters is considered to represent a single event. Because each cluster only contains articles in the same language we have also implemented a method for determining if two clusters in different languages are actually discussing the same event. If such clusters are identified they are represented by a single event. Once events are formed we extract from them the relevant information. This information includes the time of the event, its location and the main actors of the event. We also perform categorization of the event into one of the DMoz categories.

The identified events are saved in the event registry where they are indexed across all available event features. Event registry allows us to perform search queries on the events. The resulting data can be the actual list of matching events or one of the possible data aggregates, which can be used for data visualization. The available API calls and their parameters were described in details. In the Appendix we have provided sample requests and the corresponding return data.

The developed prototype is available at http://eventregistry.ijs.si/ .

References

[AHWY03] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu. A framework for clustering evolving data streams. *Proc. of the 29th VLDB Conference*, 2003.

[AY06] C. C. Aggarwal, P. S. Yu. A framework for clustering massive text and categorical data streams. *Proc. 6th SIAM Int. Conf. on Data Mining* (SDM), 2006.

[AY10] C. C. Aggarwal, P. S. Yu. On clustering massive text and categorical data streams. *Knowledge and Information Systems*, 24(2):171-196 (2010).

[D1.3.1] Early prototype of data infrastructure, XLike deliverable

[D2.1.1] Shallow linguistic processing prototype, XLike deliverable

[D3.1.1] Early text annotation prototype, XLike deliverable

[D3.2.1] Early ontological word-sense disambiguation prototype, XLike deliverable

[D4.1.1] Cross-lingual document linking prototype, XLike deliverable

[D6.2.1] Early prototype, XLike deliverable

[Dav12] M. Davis (ed.) *Unicode Text Segmentation*. Unicode Standard Annex #29, Unicode 6.2.0, 12 September 2012.

[DMoz] "DMoz, open directory project." [Online]. Available: http://www.dmoz.org/.

[FP07] Fung, Gabriel Pui Cheong, et al. "Time-dependent event hierarchy construction." *Proceedings of the 13th ACM SIGKDD conference*. ACM, 2007.

[Geo] "GeoNames." [Online]. Available: http://www.geonames.org/.

[JA98] Allan, James, et al. "On-line new event detection and tracking." *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1998.

[Lug12] E. Lughofer. A dynamic split-and-merge approach for evolving cluster models. *Evolving Systems*, 3:135-151 (2012).

[MG09] M. Muhr, M. Granitzer. Automatic cluster number selection using a split and merge k-means approach. *DEXA Workshops* 2009, pp. 363-7.

[PM00] D. Pelleg, A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. *Proc. ICML* 2000.

Annex A Examples of event registry API calls

GetEvents request for list of events:

http://eventregistry.ijs.si/json/event?action=getEvents&conceptUri=pm_person%3Ahttp%3A%2F%2Fen.wikipedia.org%2Fwiki%2FBarack_Obama&lang=eng&sortBy=rel&page=0&count=25&resultType=list

Return data:


```
{
   "results":[{
        "id":9502,
        "uri":"9502",
        "primaryStory":{
          "uri":"491c258e-688f-41de-8c8e-e3ca71f4a19b-7139",
          "id":9502,
          "language":"eng",
          "extractedDate":"2010-10-19",
          "extractedDateEnd":"",
          "averageDate":"2013-05-15",
          "commonDates":[
            { "Amb":false, "Freq":4, "Date":"-05-09" },
            { "Amb":false, "Freq":2, "Date":"2002-01-" },
            { "Amb":false, "Freq":2, "Date":"2010-12-" }
          ],
          "extractedDateFq":4,
          "articleCount":17
        },
        "stories":[{
            "uri":"491c258e-688f-41de-8c8e-e3ca71f4a19b-7139",
            "id":9502,
            "language":"eng",
            "articleCount":17
          }
        ],
        "medoidArticles":[{
            "id":"246036",
            "uri":"http://www.foxnews.com/world/2013/05/17/human-rights-watch-says-it-has-found-evidence-syrians-were-tortured-in/",
            "title":"Human Rights Watch report says Syrians were tortured in government prisons",
            "body":"<p>BEIRUT - Rights activists visiting abandoned government prisons in the first Syrian city to come under rebel control have found torture devices and other evidence that detainees were abused there, Human Rights Watch said in a report Friday.\u000a Raqqa, in eastern Syria, was overrun in late February by rebels fighting to topple President Bashar Assad.  The rebels facilitated the New York-based group's access to facilities that had belonged to a government security agency and military intelligence in ...</p>",
            "date":"2013-05-17",
            "time":"15:34",
            "lang":"eng",
            "sim":0.6506,
            "source_title":"WWW.FOXNEWS.COM",
            "source_uri":"www.foxnews.com",
            "source_id":"229",
            "isDuplicate":false,
            "story_id":"9502"
```

```
        }],
        "articleCounts":{ "total":17, "eng":17, "deu":0, "spa":0, "zho":0, "slv":0 },
        "concepts":[{
            "uri":"pm_person:http://en.wikipedia.org/wiki/Barack_Obama",
            "id":"19969",
            "type":"pm_person",
            "label":"Barack Obama",
            "label_eng":"Barack Obama",
            "label_deu":"Barack Obama",
            "label_spa":"Barack Obama",
            "label_zho":"Barack Obama",
            "label_slv":"Barack Obama",
            "score":100
          },
          {
            "uri":"pm_wiki:http://en.wikipedia.org/wiki/Hunger_strike",
            "id":"47131",
            "type":"pm_wiki",
            "label":"Hunger strike",
            "label_eng":"Hunger strike",
            "label_deu":"Hungerstreik",
            "label_spa":"Huelga de hambre",
            "label_zho":"Huelga de hambre",
            "label_slv":"Huelga de hambre",
            "score":98
          },
          ...
        ]
    },
    ...
}
```


GetEvents request for time distribution of events:

http://eventregistry.ijs.si/json/event?action=getEvents&conceptUri=pm_person%3Ahttp%3A%2F%2Fen.wikipedia.org%2Fwiki%2FBarack_Obama&lang=eng&sortBy=rel&page=0&count=25&resultType=timeAggr

Return data:


```
{
  "results":[
    {
      "date":"2010-04-20",
      "count":1
    },
    {
      "date":"2010-05-06",
      "count":1
    },
    ....
}
```


GetEvents request for event locations:

http://eventregistry.ijs.si/json/event?action=getEvents&conceptUri=pm_person%3Ahttp%3A%2F%2Fen.wikipedia.org%2Fwiki%2FBarack_Obama&lang=eng&sortBy=rel&page=0&count=25&resultType=locAggr

Return data:


```
{
   "results":[{
         "count":62,
         "city":{
            "wikiUri":"pm_location:http://en.wikipedia.org/wiki/Benghazi",
            "labelEng":"Benghazi",
            "labelSpa":"Bengasi",
            "labelZho":"Benghazi",
            "labelDeu":"Banghazi",
            "labelSlv":"Benghazi",
            "population":650629,
            "lat":32.11667,
            "long":20.06667
         },
         "country":{
            "wikiUri":"pm_location:http://en.wikipedia.org/wiki/Libya",
            "labelEng":"Libya",
            "labelSpa":"Libya",
            "labelZho":"Libya",
            "labelDeu":"Libysch-Arabische Dschamahirija",
            "labelSlv":"Libijska arabska Jamahiriya",
            "lat":28,
            "long":17
         }
      },
      …
}
```


GetEventInfo request:

http://eventregistry.ijs.si/json/event?action=getEventInfo&eventId=9502

Return data:


```
{
   "eventInfo":{
      "id":9502,
      "uri":"9502",
      "primaryStory":{
         "uri":"491c258e-688f-41de-8c8e-e3ca71f4a19b-7139",
         "id":9502,
         "language":"eng",
         "extractedDate":"2010-10-19",
         "extractedDateEnd":"",
         "averageDate":"2013-05-15",
         "commonDates":[
               { "Amb":false, "Freq":4, "Date":"-05-09" },
               { "Amb":false, "Freq":2, "Date":"2002-01-" },
               { "Amb":false, "Freq":2, "Date":"2010-12-" }
```

```
        ],
        "extractedDateFq":4,
        "articleCount":17
    },
    "stories":[{
        "uri":"491c258e-688f-41de-8c8e-e3ca71f4a19b-7139",
        "id":9502,
        "language":"eng",
        "articleCount":17
        }
    ],
    "medoidArticles":[{
        "id":"246036",
        "uri":"http://www.foxnews.com/world/2013/05/17/human-rights-watch-says-it-has-found-
evidence-syrians-were-tortured-in/",
        "title":"Human Rights Watch report says Syrians were tortured in government prisons",
        "body":"<p>BEIRUT - Rights activists visiting abandoned government prisons in the first Syrian city
to come under rebel control have found torture devices and other evidence that detainees were abused
there, Human Rights Watch said in a report Friday.\u000a Raqqa, in eastern Syria, was overrun in late
February by rebels fighting to topple President Bashar Assad.  The rebels facilitated the New York-based
group's access to facilities that had belonged to a government security agency and military intelligence
in ...</p>",
        "date":"2013-05-17",
        "time":"15:34",
        "lang":"eng",
        "sim":0.6506,
        "source_title":"WWW.FOXNEWS.COM",
        "source_uri":"www.foxnews.com",
        "source_id":"229",
        "isDuplicate":false,
        "story_id":"9502"
        }
    ],
    "articleCounts":{ "total":17, "eng":17, "deu":0, "spa":0, "zho":0, "slv":0 },
    "concepts":[{
        "uri":"pm_person:http://en.wikipedia.org/wiki/Barack_Obama",
        "id":"19969",
        "type":"pm_person",
        "label":"Barack Obama",
        "label_eng":"Barack Obama",
        "label_deu":"Barack Obama",
        "label_spa":"Barack Obama",
        "label_zho":"Barack Obama",
        "label_slv":"Barack Obama",
        "score":100
        },
        …
        ]
}
```


GetEventArticles request:
http://eventregistry.ijs.si/json/event?action=getEventArticles&eventId=9502&page=0&count=30

Return data:


```
{
   "articles":[{
         "id":"246036",
         "uri":"http://www.foxnews.com/world/2013/05/17/human-rights-watch-says-it-has-found-evidence-syrians-were-tortured-in/",
         "title":"Human Rights Watch report says Syrians were tortured in government prisons",
         "body":"<p>BEIRUT - Rights activists visiting abandoned government prisons in the first Syrian city to come under rebel control have found torture devices and other evidence that detainees were abused there, Human Rights Watch said in a report Friday.\u000a Raqqa, in eastern Syria, was overrun in late February by rebels fighting to topple President Bashar Assad.  The rebels facilitated the New York-based group's access to facilities that had belonged to a government security agency and military intelligence in ...</p>",
         "date":"2013-05-17",
         "time":"15:34",
         "lang":"eng",
         "sim":0.6506,
         "source_title":"WWW.FOXNEWS.COM",
         "source_uri":"www.foxnews.com",
         "source_id":"229",
         "isDuplicate":false,
         "story_id":"9502"
      },
      ...
}
```


GetTrendingEvents request:

http://eventregistry.ijs.si/json/event?action=getTrendingEvents&count=10&lang=eng

Return data:


```
{
   "trendingEvents":[{
         "articles":[{
               "id":"6240",
               "uri":"http://www.nydailynews.com/sports/more-sports/mcgaughey-prepares-kentucky-derby-winner-orb-belmont-article-1.1339904?localLinksEnabled=false",
               "title":"McGaughey preps Kentucky Derby winner Orb for Preakness Stakes",
               "body":"",
               "date":"2013-05-10",
               "time":"00:48",
               "lang":"eng",
               "sim":0.669,
               "source_title":"WWW.NYDAILYNEWS.COM",
               "source_uri":"www.nydailynews.com",
               "source_id":"17",
               "isDuplicate":false,
               "story_id":"4089"
            },
            ...
         ],
```

```
        "totalCount":355,
        "eventDetails":{
           "id":4089,
           "uri":"4089",
           ...
        }
     },
     {
        "articles": ...
     }
  ]
}
```


