



## D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v.2

Alexander Simov (Ontotext)  
Ian Roberts (USFD)

### Abstract

The first chapter in this document describes the work on adapting the GATE Mimir tool for indexing streams of data. Mimir provides indexing infrastructure for integrated semantic search of collections of annotated documents. The second part of the document presents the updated version of TrendMiner user interface. It includes a description of the identified requirements and an overview of the interface of several tools for streaming media analysis. The final sections describe the specific customizations made for the usecase demonstrators.

FP7-ICT Strategic Targeted Research Project TrendMiner (No. 287863)  
Deliverable D4.2.2 (WP 4)

**Keyword list:** multi-paradigm search, user interface, user interaction, visualisation, JavaScript, sentiment index

Nature: **Demonstration**

Dissemination: **PU**

Contractual date of delivery: **M34**

Actual date of delivery: **31 Aug 2014**

Reviewed By: **José Luis Martínez Fernández (DAEDALUS)**

Web links: **(none)**

## **TrendMiner Consortium**

This document is part of the TrendMiner research project (No. 287863), partially funded by the FP7-ICT Programme.

### **DFKI GmbH**

Language Technology Lab  
Stuhlsatzenhausweg 3  
D-66123 Saarbrücken, Germany  
Contact person: Thierry Declerck  
E-mail: [declerck@dfki.de](mailto:declerck@dfki.de)

### **University of Southampton**

Southampton SO17 1BJ, UK  
Contact person: Mahensan Niranjana  
E-mail: [mn@ecs.soton.ac.uk](mailto:mn@ecs.soton.ac.uk)

### **Internet Memory Research**

45 ter rue de la Rvolution  
F-93100 Montreuil, France  
Contact person: France Lafarges  
E-mail: [contact@internetmemory.org](mailto:contact@internetmemory.org)

### **Eurokleis S.R.L.**

Via Giorgio Baglivi, 3  
Roma RM 0016, Italia  
Contact person: Francesco Bellini  
E-mail: [info@eurokleis.com](mailto:info@eurokleis.com)

### **University of Sheffield**

Department of Computer Science  
Regent Court, 211 Portobello St.  
Sheffield S1 4DP, UK  
Contact person: Kalina Bontcheva  
E-mail: [K.Bontcheva@dcs.shef.ac.uk](mailto:K.Bontcheva@dcs.shef.ac.uk)

### **Ontotext AD**

Polygraphia Office Center fl.4,  
47A Tsarigradsko Shosse,  
Sofia 1504, Bulgaria  
Contact person: Atanas Kiryakov  
E-mail: [naso@sirma.bg](mailto:naso@sirma.bg)

### **Sora Ogris and Hofinger GmbH**

Bennogasse 8/2/16  
A-1080 Wien, Austria  
Contact person: Christoph Hofinger  
E-mail: [ch@sora.at](mailto:ch@sora.at)

### **Hardik Fintrade Pvt Ltd.**

227, Shree Ram Cloth Market,  
Opposite Manilal Mansion,  
Revdi Bazar, Ahmedabad 380002, India  
Contact person: Suresh Aswani  
E-mail: [m.aswani@hardikgroup.com](mailto:m.aswani@hardikgroup.com)

### **DAEDALUS - DATA, DECISIONS AND LANGUAGE, S. A.**

C/ López de Hoyos 15, 3º, 28006 Madrid,  
Spain

**Contact person: José Luis Martínez  
Fernández**

Email: [jmartinez@daedalus.es](mailto:jmartinez@daedalus.es)

### **Institute of Computer Science Polish Academy of Sciences**

5 Jana Kazimierza Str., Warsaw, Poland  
Contact person: Maciej Ogrodniczuk  
E-mail: [Maciej.Ogrodniczuk@ipipan.waw.pl](mailto:Maciej.Ogrodniczuk@ipipan.waw.pl)

### **Universidad Carlos III de Madrid**

Av. Universidad, 30, 28911, Madrid, Spain  
Contact person: Paloma Martínez Fernández  
E-Mail: [pmf@inf.uc3m.es](mailto:pmf@inf.uc3m.es)

### **Research Institute for Linguistics of the Hungarian Academy of Sciences**

Benczúr u. 33., H-1068 Budapest, Hungary  
Contact person: Tamás Váradí  
Email: [varadi.tamas@nytud.mta.hu](mailto:varadi.tamas@nytud.mta.hu)

D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments  
in streaming media - v2

## **Executive Summary**

This deliverable describes the user interaction with the TrendMiner web-based portal.

The first chapter in this document describes the work on adapting the GATE Mimir tool for indexing streams of data. Mimir provides indexing infrastructure for integrated semantic search of collections of annotated documents. The second part of the document presents the updated version of TrendMiner user interface. It includes a description of the identified requirements and an overview of the interface of several tools for streaming media analysis. The final sections describe the specific customizations made for the use case demonstrators.

## Contents

<b>Executive Summary .....</b>	<b>4</b>
<b>Contents.....</b>	<b>5</b>
<b>List of Acronyms .....</b>	<b>6</b>
<b>List of Figures.....</b>	<b>7</b>
<b>Relevance to TrendMiner .....</b>	<b>8</b>
Relevance to the project objectives .....	8
Relevance to other work packages .....	8
<b>1 Multi-paradigm search software .....</b>	<b>9</b>
GATE Mimir .....	9
Indexing Annotated Documents with Mimir .....	11
Ranking of Search Results .....	13
Adapting for Streaming Data .....	14
Experiments and Performance Evaluation.....	17
Conclusions and Recommendations .....	18
<b>2 Cross-lingual web UI for trends/sentiments in streaming media.....</b>	<b>20</b>
Requirements for the user interface.....	20
Functional requirements.....	20
Non-functional requirements .....	22
<b>3 TrendMiner UI Architecture .....</b>	<b>23</b>
Track configuration .....	23
Results visualisation .....	24
Track filtering.....	24
<b>4 Web UI Improvements .....</b>	<b>26</b>
More interactive components.....	26
Query expansion & data normalisation .....	26
Data export.....	26
Clustering Browsing and Management .....	27
Clustering Browser .....	27
Clustering Management .....	27
<b>5 Usecases Extensions.....</b>	<b>29</b>
WP6 financial usecase extensions (EK) .....	29
WP7 political usecase extensions (SORA) .....	30

### **List of Acronyms**

<b>API</b>	Application Programming Interface
<b>AJAX</b>	Asynchronous JavaScript and XML
<b>HTML</b>	HyperText Markup Language
<b>JSON</b>	JavaScript Object Notation
<b>REST</b>	Representational State Transfer
<b>UI</b>	User Interface

## List of Figures

Figure 1. Mimir life cycle .....	10
Figure 2. Overall Mimir architecture .....	11
Figure 3. Inverted and direct index representation.....	11
Figure 4. High level view of the data and execution flow when creating and searching a .....	12
Figure 5. The three stages in the life-cycle of a Mimir index, prior to version 5:0. Different.....	15
Figure 6. A sync-to-disk operation saves the recent document data to a new on-disk	16
Figure 7. A compact operation merges all the index batches currently in the index cluster .....	16
Figure 8. The standard Mimir architecture for federated indexes. The federated index implementation handles both load distribution for indexing, and the execution of federated queries. ....	19
Figure 9. The suggested architecture for federated indexes in TrendMiner. A custom load balancer takes over the responsibility for distributed indexing, while the execution of federated queries is left to the federated index. ....	19
Figure 10. UI Workflow diagram.....	23
Figure 11. Track configuration .....	24
Figure 12. Track filtering.....	25
Figure 13. Clustering Browser .....	27
Figure 14. New clustering dialog .....	28
Figure 15. Delete clustering .....	29
Figure 16. Stock market and twitter data.....	29
Figure 17. Fine-grained time intervals exploration.....	31

## **Relevance to TrendMiner**

### **Relevance to the project objectives**

The TrendMiner project aims to create tools for the analysis of vast amounts of multilingual streaming data in real time. However, in order to make those tools usable for real-world users, it is essential to hide all complicated functionality behind a user interaction process which is as straightforward and intuitive as possible. The main goal of this deliverable is to present the current version of this frontend layer.

### **Relevance to other work packages**

The primary goal of the user interface is to allow the users to make sense of the information extracted from various streaming data sources. Thus, the design choices are highly dependent on the outputs of the tools developed within *WP2 Multilingual Ontology-Based Information Extraction and Knowledge Modeling* and *WP3 Machine Learning models for Mining Trends from Streaming Media*.

Moreover, it is essential to develop a user experience which directly reflects the users' needs and expectations in order to render it intuitive and easy to use. To this end, the user interface design is related to the outputs of the requirements gathered in work packages *WP6 Multilingual Trend Mining and Summarisation for Financial Decision Support* and *WP7 Multilingual Public Spheres: Political Trends and Summaries*.

It is also important to understand what kind of interaction the potential users will expect from the system. This is an essential task because of the users' tendency to be dissatisfied with user interface designs with which they are unfamiliar. In order to discover the users' expectations, we use the results from the market watch activities performed as a part of *WP8 Dissemination and Exploitation*.



## 1 Multi-paradigm search software

This chapter describes the work that was done to adapt the GATE Mimir framework for integrated semantic search to work with streaming data. In the first section we provide some background description of Mimir. Next we present the work done in order to add support for streaming data. We conclude the chapter with some experimental results and recommendations.

### GATE Mimir

**Mimir**<sup>1</sup> is an integrated semantic search framework, which offers indexing and search over full text, document structure, document metadata, linguistic annotations, and any linked, external semantic knowledge bases. It supports hybrid queries that arbitrarily mix full-text, structural, linguistic and semantic constraints. A key distinguishing feature are the containment operators, that allow flexible creation and nesting of full-text, structural, and semantic constraints, as well as the support for interactive knowledge discovery.

Mimir has been designed as a generic and extensible, open source framework<sup>2</sup>. It can also be used as an on-demand, highly scalable semantic search server, running on the GATECloud [TRCB13] platform.

The high-level concept behind Mimir is illustrated in Figure 1. First a document collection is processed with NLP algorithms, such as those provided by GATE [CMBT02]. Typically the semantic annotations also refer to Linked Open Data resources, accessed via a triple store, such as OWLIM [Kir06] or Sesame [BKVH02]. The semantically annotated documents are then indexed in Mimir, together with their full-text content, document metadata, and document structure markup. At search time, the triple store is used as a source of implicit knowledge, to help answer the hybrid searches that combine full-text, structural, and semantic constraints. The latter are formulated using a SPARQL query, executed against the triple store.

---

<sup>1</sup> Old Norse “The rememberer, the wise one”

<sup>2</sup> Download from <http://gate.ac.uk/mimir/>.

D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

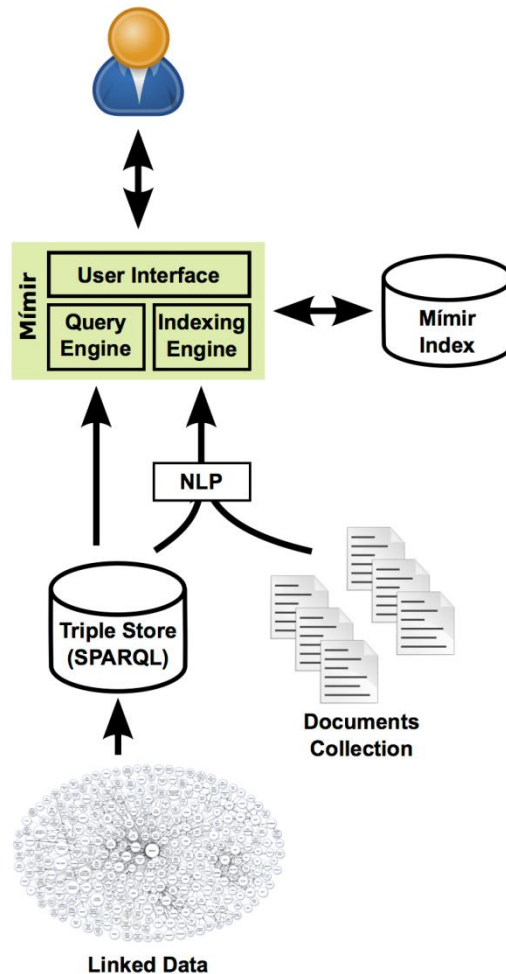


Figure 1. Mimir life cycle

Mimir's architecture is shown in Figure 2. It is implemented as a web application that runs server-side and can optionally be distributed across multiple machines. It includes both information seeking and information discovery user interfaces, as well as a REST-style API for programmatic access. Each Mimir instance manages one or more indexes stored locally. Additionally, it can also use the REST APIs to access any index served by remote Mimir instances and make it available locally. Any set of local and remote indexes can be grouped together in a *federated index*.

When used in combination with remote indexes, federation enables Mimir to scale to millions of documents, through very large indexes, distributed across multiple servers. This enables:

- Faster indexing: less content in each Mimir index;
- Faster searches: search space is broken into slices that are searched in parallel;
- Joining search results is trivial: union of result sets;
- Scalability through adding more nodes and creating a federated index;
- Search speed stays almost constant while data increases: each individual repository has the same amount of data; there are just more repositories, federated together.

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

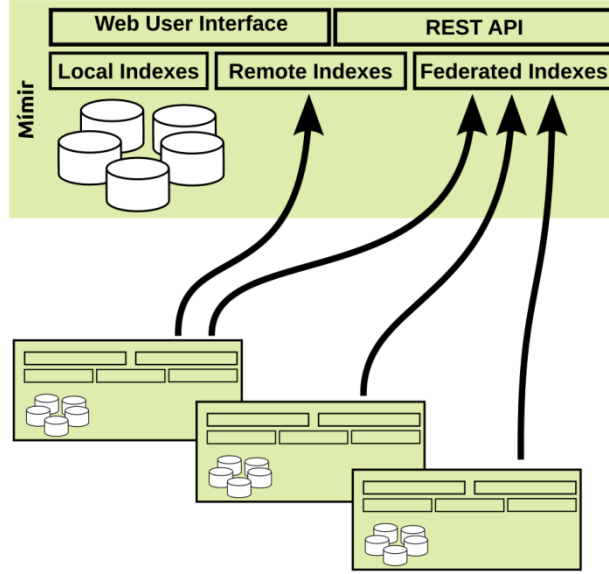


Figure 2. Overall Mimir architecture

$docID_1^1(pos, pos, \dots pos);$ $termID^1 \rightarrow docID_2^1(pos, pos, \dots pos);$ $\dots$
$docID_1^2(pos, pos, \dots pos);$ $termID^2 \rightarrow docID_2^2(pos, pos, \dots pos);$ $\dots$
$\dots$
$docID^1 \rightarrow termID_1^1(count);$ $termID_2^1(count); \dots$
$docID^2 \rightarrow termID_1^2(count);$ $termID_2^2(count); \dots$
$\dots$

Figure 3. Inverted and direct index representation

#### Indexing Annotated Documents with Mimir

Mimir uses a compound index approach, where different types of content are stored in separate but aligned inverted and direct indexes.

Inverted indexes are used to find which of the documents contain occurrences of search terms, where terms may be words, annotations, or entity mentions. These indexes are used to support information-seeking tasks where each query is executed to produce a result set comprising documents.

The inverted indexes include position information, which is represented in terms of token offsets. Ignoring implementation details, an inverted index is a list of term IDs and associated posting lists (see top half of Figure 3). The posting list for each term is a sequence of postings, each including a document ID and a position. Term and

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

document IDs are long integers and consistent between the corresponding direct and inverted indexes.

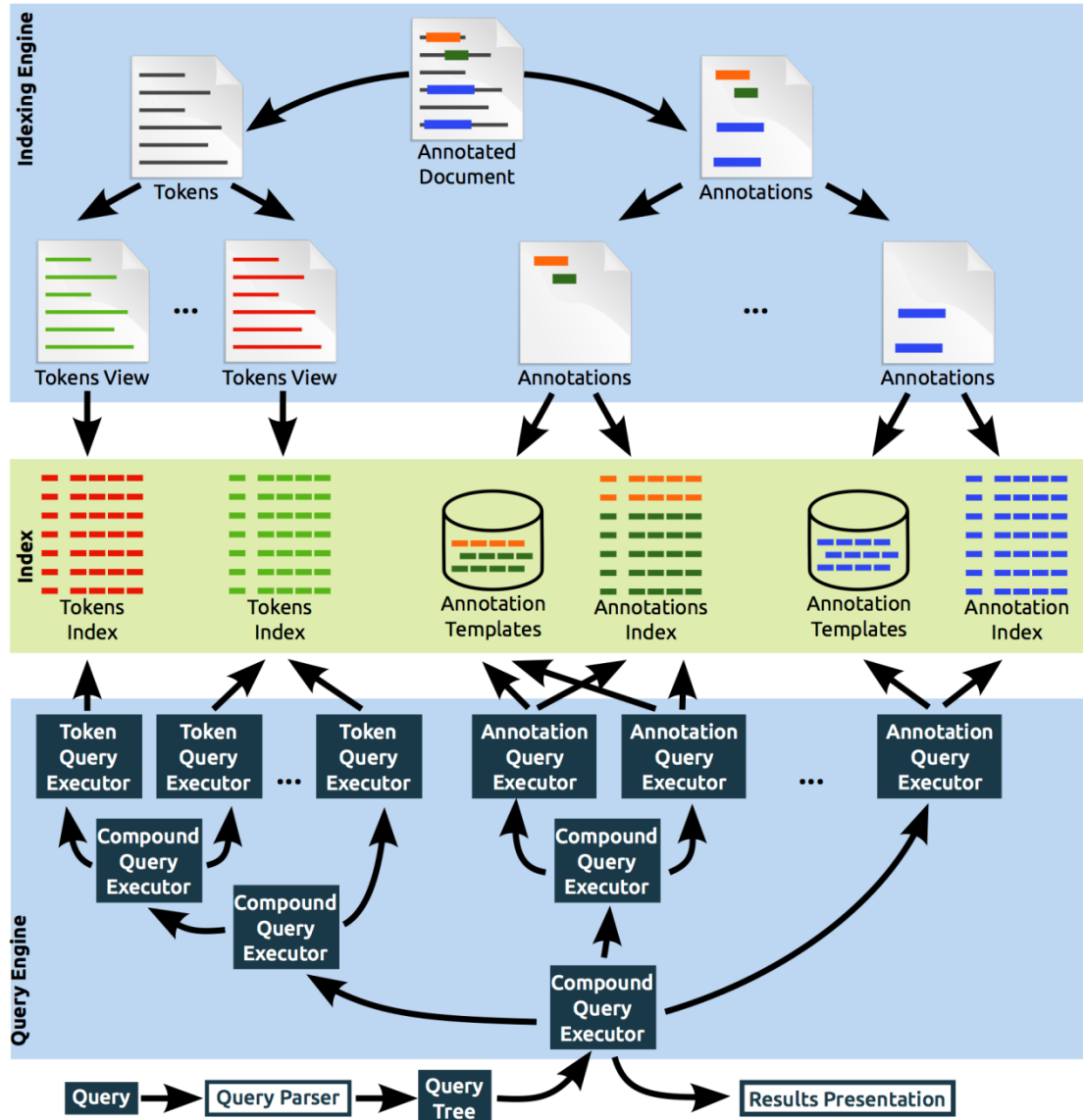


Figure 4. High level view of the data and execution flow when creating and searching a Mimir index

The upper part of Figure 4 illustrates the indexing process, while the bottom refers to the execution of queries. The central band shows the types of data stored into the compound index.

Term IDs are ordered by the lexicographic order of the term strings; document IDs represent the indexing order of the documents; and all posting lists are sorted by document or term ID.

Mimir also provides *direct indexes*, which perform the opposite mapping: from documents to terms. Given a set of documents (typically retrieved via an earlier search) direct indexes are used to find which terms, of any type, occur in those documents, and with what frequency. This functionality is used to support information discovery tasks, which are exploratory in nature and users often require

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

visualisations of the most frequent terms, co-occurrence matrices, and other interfaces for interactive document retrieval and analysis.

The use of direct indexes is optional and can be configured when a new index is being built. Direct indexes are smaller than their inverted counterparts because they are not required to include position information, as that is already stored in the corresponding inverted index. The two types of index are illustrated in Figure 3.

The on-disk indexes are implemented using the MG4J library<sup>3</sup> [BV05]; if required, similar indexing structures can be implemented using alternative IR engines, such as Lucene[HG04].

Conceptually, a document that has been processed with NLP tools comprises textual content and annotations (some encode structural markup, whereas other represent automatically created syntactic and semantic information). Additionally, document metadata may also be available (e.g. mime type, creation date). Mimir uses the document representation format defined by the GATE framework [CMBT02] as an interchange format, since it can represent all these types of information.

##### *Ranking of Search Results*

Mimir was designed originally for information discovery searches, needed for example by intellectual property search experts and environmental science researchers. The primary focus in this case is on enabling users to formulate very precise queries with elaborate semantic constraints. Once a matching document set is narrowed down, such users would typically investigate the entire result set. This is due to the information discovery nature of their search, which is different from seeking to retrieve a specific, best-matching document [Pir09]. In such applications, ranking can be disabled.

In information seeking applications, however, ranking of search results is indeed necessary. Therefore, Mimir provides implementations for a set of popular ranking algorithms, and new ones can be implemented on-demand through a plug-in mechanism. For each Mimir index, a developer can choose between:

- **No scoring:** All documents are ranked equally; results are returned in the order they are found in the index. This is the default setting.
- **Count scoring:** Matching documents are ranked according to the number of query matches contained.
- **BM25:** Documents are ranked using the Okapi BM25 algorithm [RWJ+95]. For text elements, the BM25 algorithm is applied as usual. For annotations, it is constructed from an OR between all the matching annotations. For example, if a document collection contains annotations *{Person gender={male|female}}*, and a query is presented for *{Person gender=male}*, then all occurrences of *<Person gender="male">* annotations in the document collection are treated as the same term. When queries combine both text and annotations, ranking scores are combined. For example, for the query “the name of *{Person gender=male}*” “the name of” is scored against the text index, while *{Person gender=male}* is scored against the annotation index. The BM25 scores for each part are calculated independently, and then combined as a weighted sum.

---

<sup>3</sup> <http://mg4j.di.unimi.it/>

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

- **TF.IDF:** Documents are ranked using the TF.IDF algorithm [Jon72], in a manner analogous to BM25 ranking.
- **Hit length scoring:** Documents are ranked according to the length of the matching snippets within. While the other algorithms are provided by wrapping the MG4J scorer implementations, this scoring algorithm is a stand-alone implementation, that uses the Mimir API directly. It is provided as a simple example of a scorer implementation that developers can use as a starting point for their own scoring algorithms. Its actual suitability for scoring search results may be limited.

In order to help users with *relevance judgements*, Mimir highlights matching textual snippets in each returned document. Each snippet provides the textual context around each search hit, separated by “...” if there are more than one hit per document.

### Adapting for Streaming Data

The ability of Mimir to provide multiple paradigm access to document collections was of great interest to TrendMiner, as it could power innovative ways of finding and presenting content. However, Mimir had one important shortcoming – it had been designed to work with static document collections, or at least with collections that changed slowly. This was unsuitable for use in TrendMiner and this section describes the work done to address this issue, which has ultimately led to the production of version 5.0 of Mimir.

Prior to version 5.0, Mimir indexes were read-only. The typical life cycle of a Mimir index included several stages, described next (and also illustrated in Figure 5):

1. Send all the documents inside a collection to the Mimir server for indexing. During this process, the index being created is in *indexing* mode, and is ready to accept new documents for indexing, but cannot be used to answer any queries.
2. Close the index. This is a relatively lengthy process, during which all the index batches created at the previous step are joined together into an unitary index. During this stage, the index can neither accept new documents for indexing, nor answer any queries.
3. Search the index. Once the index closing process has completed, the index switches to *searching* mode, and can be used to answer queries. At this point no further documents can be indexed.

If an indexed document collection needed to be updated with new documents, the only way to achieve that was to create a new index with the new documents, and then join the old and the new index into a federated index which can produce results containing both old and new documents. This was a cumbersome process, due to the fact that newlyindexed documents were not searchable until the new index was closed and added to the federation. If a collection received frequent updates, then the federated index would end up containing a very large number of sub-indexes.

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

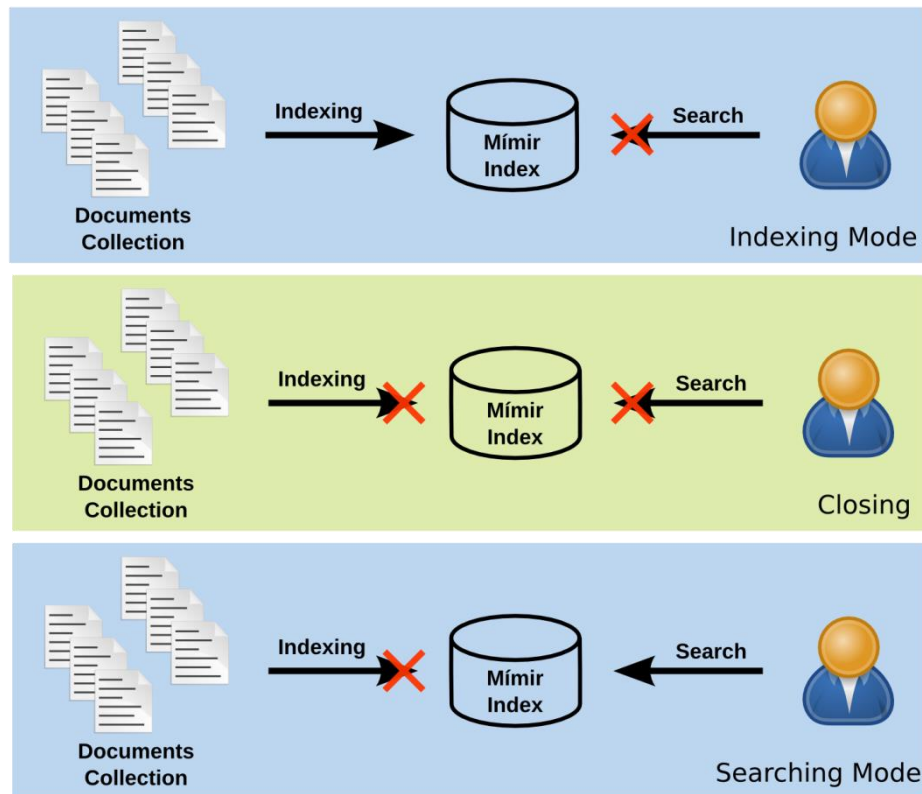


Figure 5. The three stages in the life-cycle of a Mimir index, prior to version 5:0. Different types of operations are possible at different stages, and no stage allows concomitant searching and indexing.

To improve the suitability of Mimir for large amounts of streaming data, we implemented live indexes that can accept new documents for indexing at the same time as they serve queries based on the documents already indexed. Mimir relies on MG4J for the implementation of on-disk indexes, and MG4J indexes are designed to be read-only, in order to maximise performance. Given these constraints, the approach we used was to index incoming documents in batches, and make each batch searchable as soon as it has been produced.

At any point, the Mimir index is:

- serving queries based on all the currently existing batches, presented as a unitary index by means of an MG4J index cluster implementation;
- indexing new documents into a new batch.

This set-up is illustrated in the top half of Figure 6, where the index batch currently being constructed is shown in a dashed line.

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

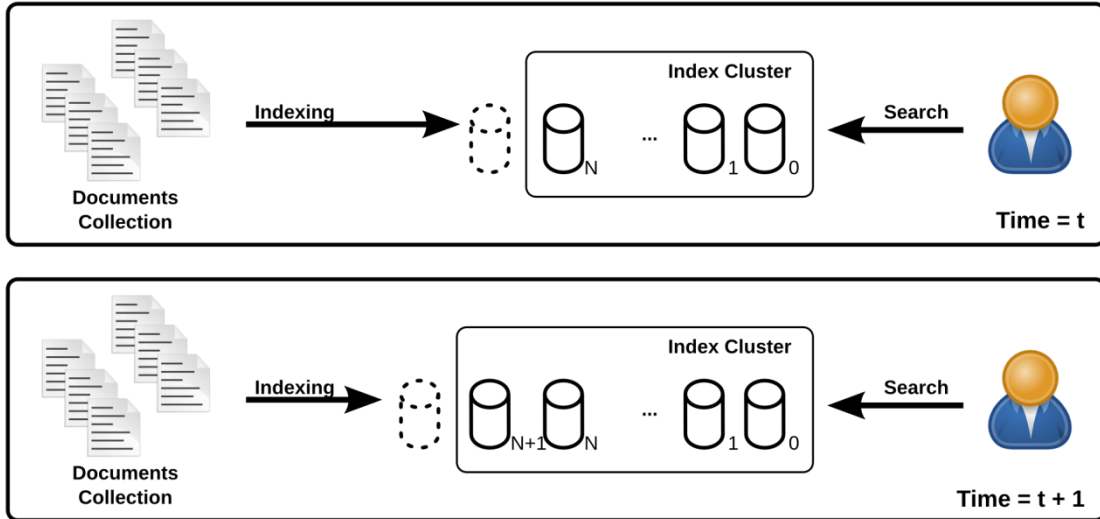


Figure 6. A sync-to-disk operation saves the recent document data to a new on-disk index batch, which is then added to the index cluster.

The maintenance of a running Mimir index relies on the following two operations.

**Sync-to-disk** is the operation that generates new index batches. For efficiency reasons all data relating to new documents submitted for indexing is accumulated in RAM. At regular intervals, or when the data reaches a pre-defined threshold, the in-RAM data is written to a new index batch. The newly created index batch is added to the index cluster that serves queries, which makes the newly indexed documents searchable. Finally, the in-RAM data structure for storing new documents is cleared, making it ready to accept new documents. This operation is presented in Figure 6. Indexes have a setting for the maximum time interval between batches. This is effectively the maximum amount of time that can pass between a document having been submitted for indexing and the moment when the document becomes searchable.

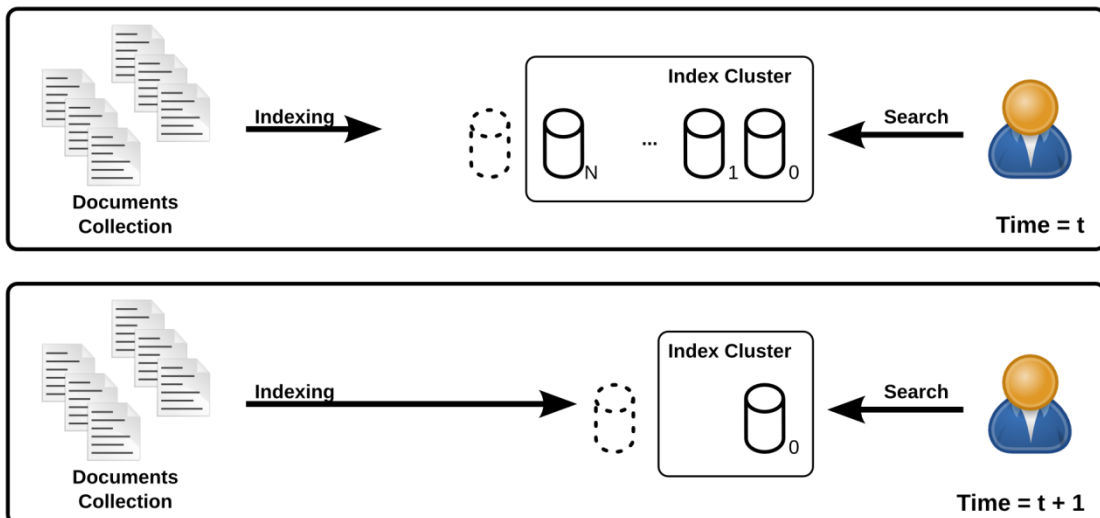


Figure 7. A compact operation merges all the index batches currently in the index cluster into a single index batch

A long-lived Mimir index will generate many index batches, depending on the number of documents being indexed, and on the setting for the time interval between batches.



#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

This can lead to problems as a very large number of files will be created, all of which need to be kept open. The **Compact** operation is used to mitigate this. Compaction, presented in Figure 7, consists of merging all the current index batches into a single batch. This operation is performed in the background whenever the number of batches grows too large (the default limit is 20 batches).

The *sync-to-disk* and *compact* operations are performed automatically when required, using background processing threads. At the same time, the Mimir index is being populated, potentially from multiple parallel foreground threads. This is needed so that the indexing is not slowed down by the need to perform maintenance operations, and has required very careful design of the interactions between the multiple threads, in order to ensure index consistency.

### Experiments and Performance Evaluation

In order to evaluate the indexing performance of the new Mimir live index, we performed some experiments that we present next.

The dataset used was a collection of 35, 000 tweets, pre-annotated with various entity mentions. This set of documents was then repeatedly sent for indexing, in an infinite loop.

We used the GCP tool<sup>4</sup> for sending documents for indexing. This had the advantage that it allowed the use of multiple parallel threads, and that it already included a connector capable of sending annotated documents to a Mimir server.

We measured the maximum indexing throughput to be around 800, 000 tweets per hour, using a single indexing node. The hardware environment consisted of a server using Intel Xeon (E5-2670) CPUs, running at 2.6 GHz. The operating system used was Ubuntu Linux 64 bit; the Mimir web application was hosted by Tomcat 7.0.34, executed using Oracle Java 1.7.0 25. During our experiment, Mimir utilised between 1 and 2 CPU cores.

In our experiments we tried using 10, 20, and 30 processing threads. We found that the increasing the number of threads over 10 had no (positive) effect on the throughput. A larger number of processing threads tended to saturate the input capacity of the Tomcat container, leading to refused connections.

The configuration for the Mimir index allowed for up to 5 indexing threads to be used for indexing. Given that the indexing process used at most 2 CPU cores, this seemed to indicate that the bottleneck lay somewhere else. We suspected that the problem was caused by the format of the data, which consists of a very large number of very small files. The GCP tool we used in our experiment opens a new connection for each document which, while normally not a problem, may be unsuitable for tweets. The Mimir server has the capacity of receiving multiple documents for each open connection, so it should be possible to batch the documents before sending them for indexing. This method could be used to reduce the number of connections being opened. It is also possible to use a persistent long-lived connection to submit a stream of documents for indexing.

In order to confirm/infirm our suspicions, we added a feature to the GCP Mimir connector allowing it to batch updates to the server at regular intervals, instead of

---

<sup>4</sup> <http://gate.ac.uk/gcp>

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

automatically opening a new connection for every new document, regardless of its size. We set up a second experiment, with the same data, but with the connection rate limited to one new connection per second. This had a dramatic effect on the indexing speed, and we were able to reach just under **3 million tweets per hour**<sup>5</sup>. This was achieved by using 20 parallel threads for the GATE pipeline that prepares data for indexing, all sending their output to the same single Mimir node.

The new set-up also drastically reduced the workload on the side of the Mimir server, which now had to manage far fewer connections. As a result of this, the Mimir server process generally used around 0.8 – 0.9 CPU cores, with very rare spikes reaching close to 1.6 cores. This suggests that the speed of the network interface rather than the CPU capacity is more likely to become a limiting factor.

#### *Conclusions and Recommendations*

In order to support higher throughput, we recommend removing all unnecessary annotations from the input documents prior to sending them for indexing. Mimir will ignore all annotations that are not explicitly configured for indexing, so their presence would not cause an error, however a large number of spurious annotations can greatly increase the amount of data transported over the network.

The best way to improve indexing throughput with Mimir is to use *federated indexes*. These are indexes that are distributed across multiple computers and which can be presented as a single unitary index (see Figure 2 for an illustration). A federated index allows indexing of a document stream in parallel, on multiple instances of Mimir. This allows the indexing process to be scaled-up as required. A recent blog post from Twitter<sup>6</sup> shows that the number of tweets per minute related to the Sochi 2014 Winter Olympics has reached 72, 630. This is equivalent to just under 4,4 million tweets per hour. Given the single-node indexing rate of around 3, 000, 000 tweets per hour, this data would require a cluster of 2 Mimir indexing nodes to ingest in real-time, whilst leaving plenty of spare capacity.

If federated indexing is to be used, we would also suggest the following slight modification of the standard architecture. By default, a federated index can be used to both distribute the indexing load between multiple nodes, and to serve federated queries that address the joint index (see Figure 8).

---

<sup>5</sup> 2, 956, 945, to be precise

<sup>6</sup> <https://blog.twitter.com/2014/farewell-sochi2014>

D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

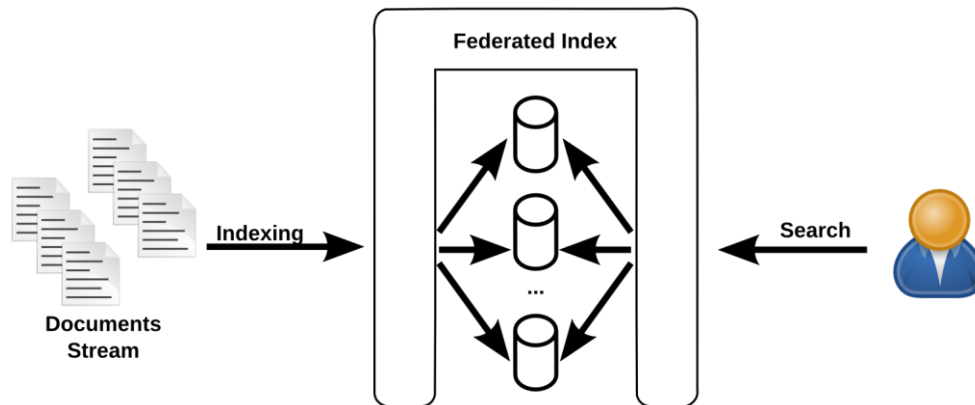


Figure 8. The standard Mimir architecture for federated indexes. The federated index implementation handles both load distribution for indexing, and the execution of federated queries.

In the interest of efficiency, we would suggest implementing a separate load balancer that can directly distribute the input stream to the multiple indexing nodes, without having to connect to the central node that manages the federated index. This would also allow the load balancer to use either batched document submission, or long-lived connections, as indicated above. This architecture is illustrated in Figure 9.

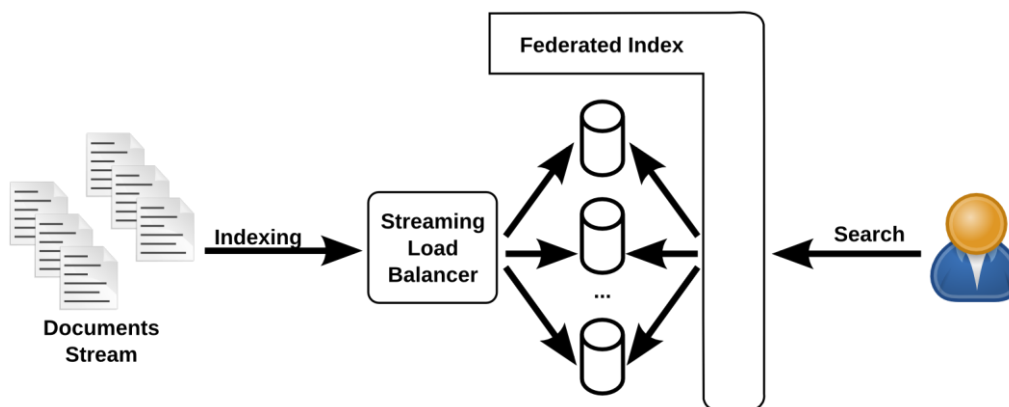


Figure 9. The suggested architecture for federated indexes in TrendMiner. A custom load balancer takes over the responsibility for distributed indexing, while the execution of federated queries is left to the federated index.

## **2 Cross-lingual web UI for trends/sentiments in streaming media**

This chapter presents the updated version of the TrendMiner UI prototype (v2), initially delivered at M24. The contents of the following sections build on top of deliverable D4.2.1 and some of the descriptions are intentionally not repeated here.

The structure of the chapter content starts with a reminder of the requirements, followed by a high level architecture overview. The substantial new content is set out in the subsequent sections covering new features and functionality as well as custom extensions for the usecases demonstrators.

### **Requirements for the user interface**

#### *Functional requirements*

This section describes what classes of functionality the interface should provide to the user and what kinds of interaction should be possible.

#### *1.1. Information to visualise*

##### *1.1.1. Sentiment over time*

The interface should visualise in a clear way the changes in the sentiment index calculated over a group of documents which match the user's search criteria. It should be obvious to the user what the trend is at each moment and which the interesting moments are.

##### *1.1.2. Activity over time*

The user should be able to understand how the streaming sources' activity changes over a user-defined period of time. The system should be able to show how the different topics matching the user's filtering criteria evolve over time and when an interesting event happens e.g. there is a major change in the stream activity or a change in the relative activity of the separate topics.

##### *1.1.3. Emerging trends detection*

The system should show information about which are the emerging topics in the stream.

##### *1.1.4. Possibility to drill down to the raw data*

The user should be able to see the raw documents matching their search criteria.

##### *1.1.5. Geo-spatial activity*

The system should visualise the geographic dimension of the activity of the streaming sources i.e. the geographic distribution of the documents matching a user query.

##### *1.1.6. Economic views*

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

The system should allow the user to compare the information extracted from the streaming sources with various economic metrics e.g. stock prices, unemployment rates etc.

##### *1.1.7. Forecasts*

The system should have a way to visualise the results of the prediction models and provide the user with suggestions about how a topic will continue to evolve.

##### *1.2. Interactive search and filter of the results*

The system should allow the user to interactively modify the set of documents which are analysed by restricting various aspects of a document. The filtering should happen responsively so that the user can move back and forth between exploring the restricted set of documents and modifying the search criteria. The features of the documents on which the user can filter include:

##### *1.2.1. Time interval*

The user should have the possibility to analyse documents only from a specific period which can vary from several hours to years.

##### *1.2.2. Topics & mentions*

The user should be able to restrict the monitoring to cover only documents which are detected to be relevant to a specific topic or mention specific entities.

##### *1.2.3. Regions*

The user should be able to restrict the analysed dataset to a limited geographic location. This requirement is needed in order to reflect the geographic dimension of most topics.

##### *1.2.4. Data source type*

The user should be able to select the data source to be analysed e.g. to monitor only tweets or only blogs etc.

##### *1.2.5. Language*

The user should be able to analyse only documents written in a certain language of interest.

##### *1.3. Parallel search*

The system should allow for the parallel exploration and comparison of at least two different document sets.

##### *1.4. Possibility to persist search templates*

It is important to give the user the possibility to persist his/her search criteria in order to replicate the results in a later moment.

##### *1.5. Possibility to export raw data*

It is essential to provide the users with the possibility to extract the raw data in a format which could be used by other industry proven tools. That way it will be possible for the user to perform analyses which are not included in the

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

TrendMiner platform. This will guarantee that the user can take full advantage of the raw data collected in the platform.

##### *Non-functional requirements*

This section describes the requirements for the UI which are not directly connected to its functionality but are important factors for the quality of the user experience and the overall value of the system.

##### *2.1. Web-based UX*

The system should be entirely Web-based. It should not require the installation of additional software or plug-ins on the user's computer. In order to fulfill the high expectations towards the user interface of a modern system, only standardised technologies like HTML5, JavaScript, AJAX etc. should be used. The system should support the latest versions of the most popular Internet browsers.

##### *2.2. Present the information in real time*

The system should respond in a timely manner, without significant delays between filter configuration and analytics dashboard rendering. This will allow the user to iteratively filter the data, review the results, and then adjust the filters again and so on.

##### *2.3. Simple and intuitive visualisations*

We should use simple and self-explanatory components to visualise and filter the information. It should be clear to the user what information is visualised just by seeing the component, so that the need for reading any sort of documentation or user guide is minimised. The users should be familiar with the type of components used for visualisation.

##### *2.4. Multilingual UI*

The user interface should have versions in the supported languages.

### 3 TrendMiner UI Architecture

This section provides a short overview of the TrendMiner UI architecture described in detail in the previous version of this document. For brevity we highlight only the major concepts and decision points to provide smooth introduction to the updates and new features available.

The following diagram (Figure 10) presents a high level overview of the components and activities involved in typical user interaction with the UI.

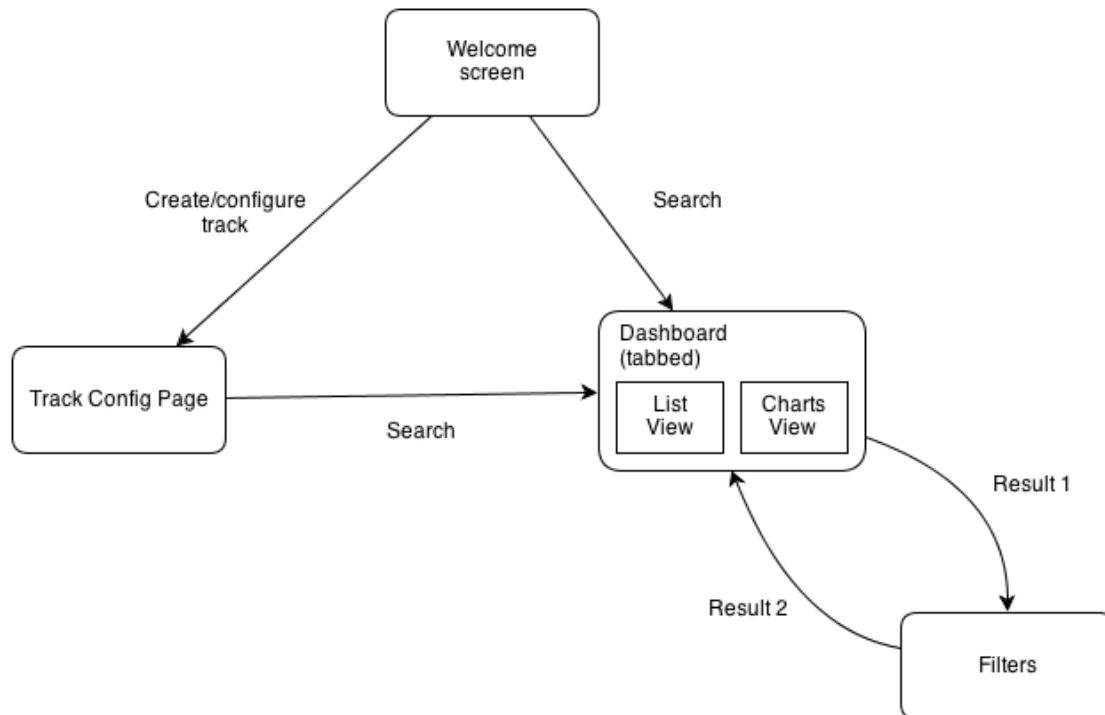


Figure 10. UI Workflow diagram

#### *Track configuration*

The top level of document filtering is the track configuration page (Figure 11). It serves as a starting point in the system. Its interface allows the user to restrict the vast amount of documents available in the system to only a relevant subset, which the analyst wants to investigate. In the terminology of TrendMiner such a subset is called a *track*.

There are several search dimensions which might be specified in a *track* configuration including:

- Topics and mentions
- Region of origin
- Data Sources
- Language of the resources
- Time period

### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

The screenshot shows a web interface for configuring a track. It includes several sections: 'Track Name' with a text input field containing 'Ebola in the Social media'; 'Topics' with a selection area showing 'Epidemic', 'Ebola', and 'Liberia' tags, and a 'Match All' checkbox; 'Regions' with a text input field; 'Data Sources' with checkboxes for 'Blogs', 'Social networks' (checked), and 'News channel'; 'Language' with a dropdown menu showing 'any language', 'English', and 'Francaise'; 'Track Period' with 'From' and 'To' date pickers both set to '03/09/2014'; 'Enable track interval' checkbox (checked); and 'Track Interval' with a horizontal slider ranging from '20m' to '6M', currently set at '24h'.

Figure 11. Track configuration

#### *Results visualisation*

Once configured, the track configuration is persisted on the server and the results are loaded from the data layer so that they can be analyzed by the user. The user interface provides two types of data visualization: text content and aggregated analytical information. The latter is implemented by a rich variety of visual charts and diagrams.

#### *Track filtering*

To facilitate the user in exploring and efficiently analyzing the results the interface provides a flexible way to perform a second level of filtering on the track results. This process is supported by a set of interactive components offering fast and light-weight approach for further search refinement, scoping down the search space based on user observations.

The filtering process comprises three phases starting from the initial track results and reaching the desired set of results. Figure 12 depicts the dependencies between the different filtering components implementing these phases. The user can interact with any of them independently and in any order.

- The **timeline filtering** component allows the users to see the level of track activity over time. The timeline component provides a way to focus precisely on those interesting periods.
- The **content filtering** component is a set of interactive widgets allowing the user to state additional restrictions based on mentions of certain types of entities or topics, co-occurring entities and their popularity, sentiment values, etc. The set of UI components deployed include: a tag cloud, a faceted search component, and auto-completing text fields.



## D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

- Finally the user has the possibility to see the refined results independently, either as text content (**Content view**) or as aggregated analytical information (**Analytical view**)

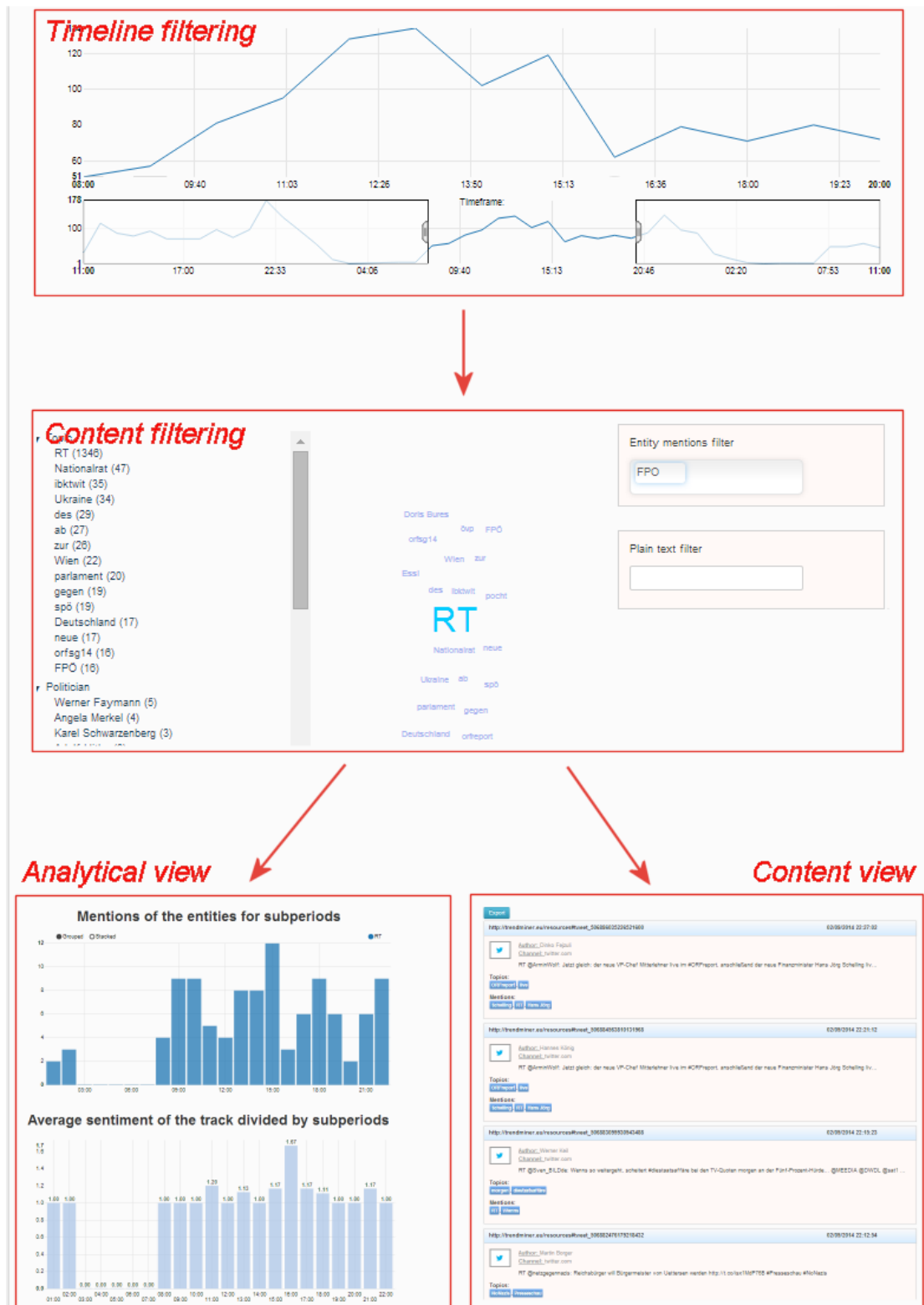


Figure 12. Track filtering

## 4 Web UI Improvements

This section contains updates and new features which were not available by the time the first prototype of the TrendMiner UI was delivered (M24). The majority of the changes were made based on the feedback collected after the first release. Others result from the availability of new processing tools in the TrendMiner platform.

### More interactive components

In the initial version, communication between some of the filtering components was not completely smooth and the user did not receive adequate feedback from the system when a filter changed its value or setting. Now all filters exchange notifications on any user interaction and the widgets are updated on the fly, following their dependency policy (recall Figure 12). Thus when the timeline filter is changed, the dependent tag cloud and faceted search components are updated based on the restrictions set by the newly selected time window. For the user this might be a decision point as to whether or not the selected time window is interesting for further exploration.

### Query expansion & data normalisation

To improve the multilinguality aspect, the new UI backend service must be able to deal with different kinds of data normalization to handle properly the use of diacritic symbols in some of the languages. Special attention was given to the umlauts in German where they could be presented in at least three forms in the text: as they appear with diacritic signs; without diacritic signs with added 'e' symbol or only by the main letter (for example: ÖVP, OeVP or OVP).

Solving this problem involves two tasks:

- query expansion - this process works with the user's input to the system during the track configuration. It involves detection of the use of umlauts in any written form and generation of all equivalent variants to be searched for in the data layer.
- results consolidation (normalisation) - this includes the collection of equivalent terms having different representations in the result set and representing them properly in the aggregated analytical views.

### Data export

This feature allows exporting of the current result set of resources, with all filters applied, into a CSV file for further reuse in other tools. Such tools can contribute additional value on the data or just provide a better visualization.

The content of the exported results includes the raw data as well as any metadata generated by the different tools in the TrendMiner platform. This functionality is accessible via the **Export** button in the *Lists* result view.

## D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

### Clustering Browsing and Management

This component provides a visual means for exploration and management of clustering sets based on the implementation of spectral clustering algorithm described in deliverable *D3.2.1 Clustering models for discovery of regional and demographic variation*.

#### Clustering Browser

This component (Figure 13) provides access to any clustering results available in the data layer. A clustering result is represented by a name and a set of clusters. Each cluster is a group of terms with their importance score, plus an optional label for the cluster.

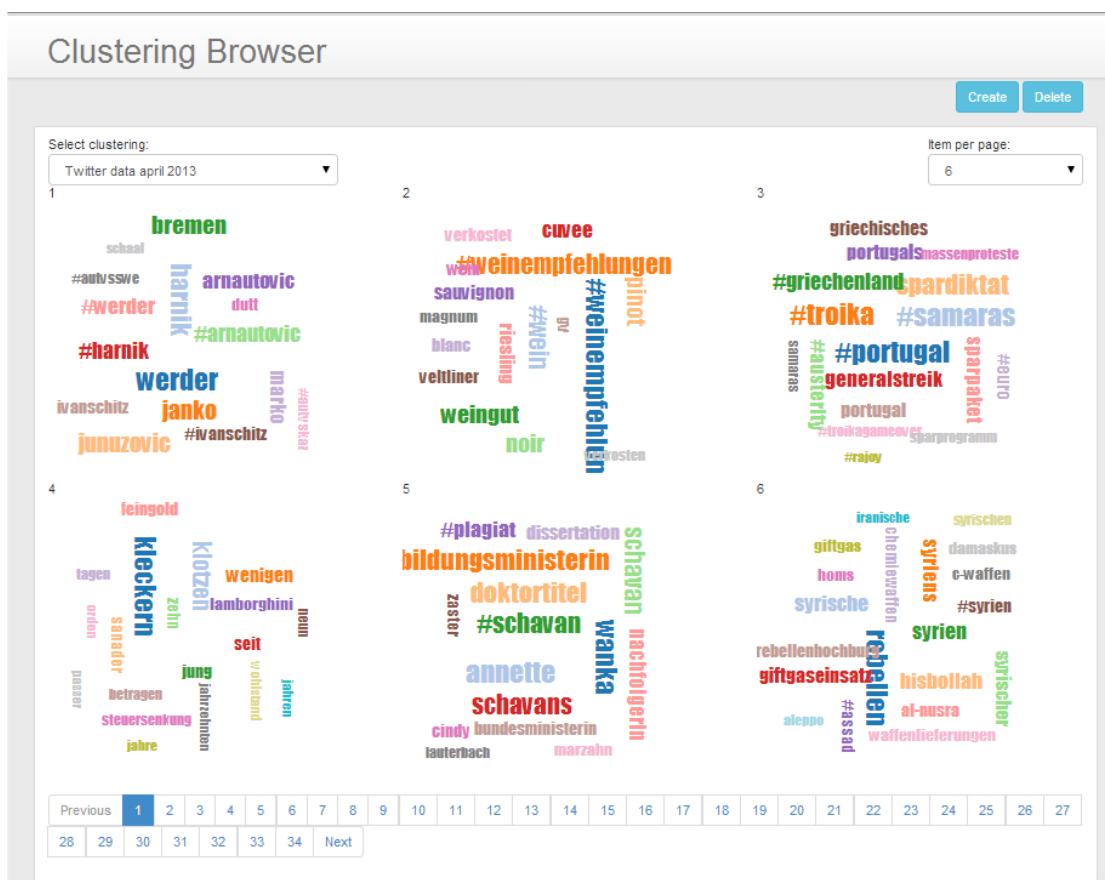


Figure 13. Clustering Browser

The user interface supports different methods to lay out the clusters as well as paging in order to avoid Web browser overloading.

#### Clustering Management

This feature provides the user the ability to create new or remove existing clusterings in the data layer.

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

On clustering creation (**Create**), the user is expected to provide the following parameters (Figure 14):

- Unique name for the result clustering
- Sources of interest
- Time period

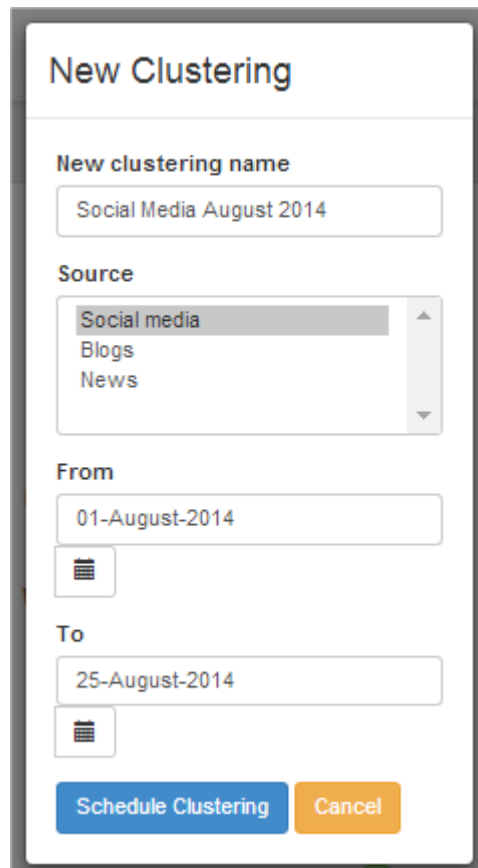
The image shows a 'New Clustering' dialog box. It has a title bar 'New Clustering'. Below it, there's a section 'New clustering name' with a text input field containing 'Social Media August 2014'. The next section is 'Source' with a dropdown menu showing 'Social media', 'Blogs', and 'News'. Below that is the 'From' section with a date input field containing '01-August-2014' and a calendar icon. The 'To' section has a date input field containing '25-August-2014' and a calendar icon. At the bottom, there are two buttons: 'Schedule Clustering' (blue) and 'Cancel' (orange).

Figure 14. New clustering dialog

The creation of a clustering is a computationally heavy operation which might take hours to complete. Therefore the user interaction is implemented in an asynchronous manner - a non blocking request is sent to the clustering service and as soon as the clustering is done it will appear as a choice in the **Select** clustering field of the browser.

The removal of existing clustering results is accesible via the **Delete** button on the main page. The deletion process is supported by a simple selection and confirmation dialog (Figure 15). The removal procedure does not involve heavy computation and it is performed synchronously.

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

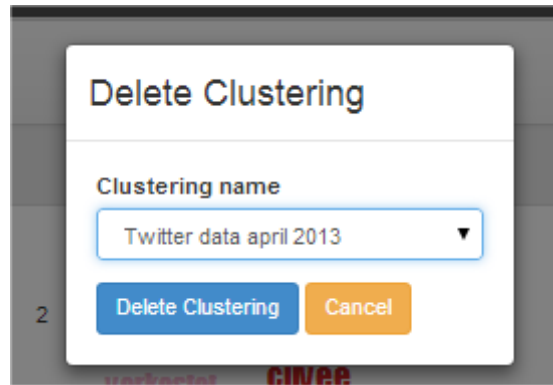


Figure 15. Delete clustering

## 5 Usecases Extensions

This section describes the customisations of the UI to serve specific needs of the usecases which are not applicable to, or not required by, the rest.

### WP6 financial usecase extensions (EK)

For the purpose of the financial usecase it is of great importance to be able to align stock market information with social media timeline information. In this way it is easy to analyze the correlation between prices and hot topics and trends over time.

We implemented this functionality as an additional diagram on top of the main timeline filter (Figure 16), allowing the user to focus on specific period by moving or changing the time window selector.

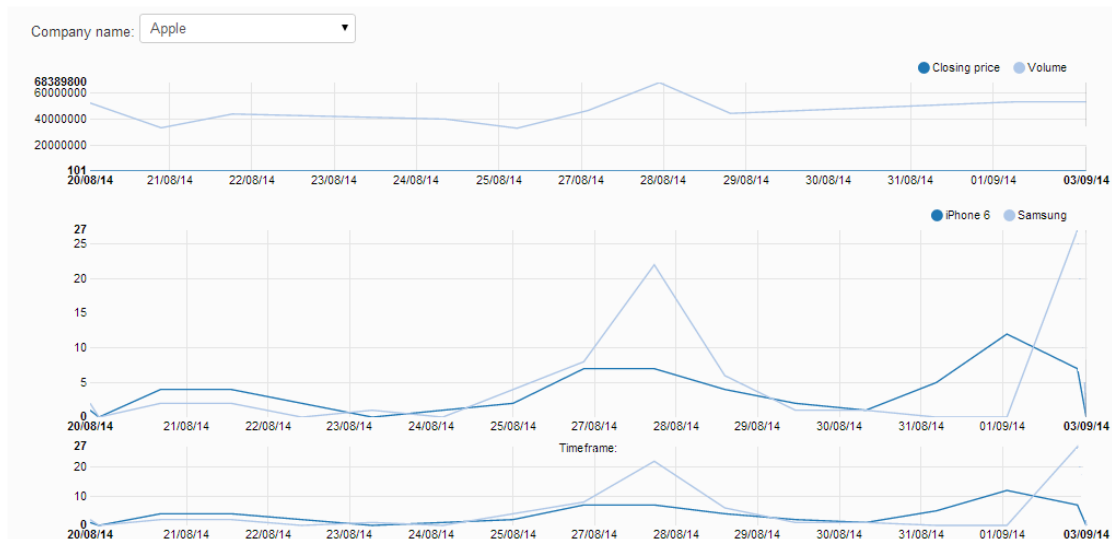


Figure 16. Stock market and twitter data

The current implementation focuses on two aspects of the data: *Closing price* and *Volume*, diagram lines that can be switched on and off in the UI (top right corner of the diagram). On top there is a company selector field listing all available company records in the data store.

### WP7 political usecase extensions (SORA)

A major point of concern for the political usecase is the timely delivery of new data to the user, live updates and ability for fine-grained data exploration. To approach these requirements we had to rework not only the UI frontend and supporting services but also the content delivery (out of scope for this document).

The improvements of the UI started with the *track configuration* phase where we support more options for time interval specification (see Figure 11). Now we support fixed time intervals (from - to dates) as well as relative interval with minimal value of 20 minutes. This enables the user to focus on the very recent publications.

Related to the timely data delivery is the ability to request 'fresh data' in a easy way without switching pages and reconfiguring tracks. To achive this we provided an **Update** button on each page where results are represented including the filters page. In this way the user interface is reloaded on request with the very latest updates available in the system.

To support fine-grained data exploration we implemented adaptive data representation procedures which dynamically change the data representation granularity in the UI depending on the observed period. Now some of the views (where the time aspect is present) are able to show information up to the minute. Figure 17 demonstrates results on an hourly basis.

D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

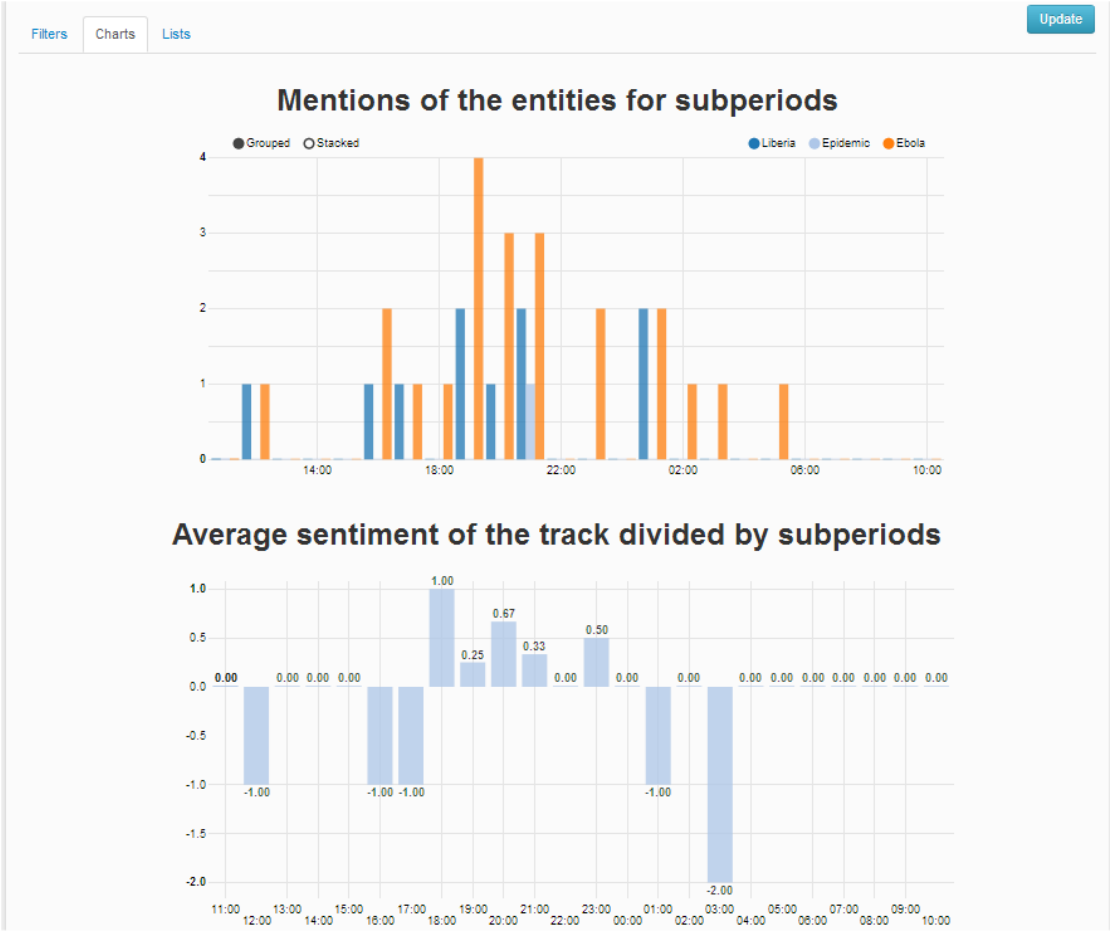


Figure 17. Fine-grained time intervals exploration

#### D4.2.1 Multi-paradigm search software; Cross-lingual web UI for trends/sentiments in streaming media - v2

##### Bibliography

[BKVH02] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *The Semantic WebISWC 2002*. Springer, 2002.

[BV05] Paolo Boldi and Sebastiano Vigna. MG4J at TREC 2005. In Ellen M. Voorhees and Lori P. Buckland, editors, *Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005)*, 15–18 November 2005, volume 500 of *Special Publications*, pages 266–271. NIST, 2005. <http://mg4j.dsi.unimi.it/>.

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. Gate: an architecture for development of robust hlt applications. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 7–12 July 2002, ACL '02, pages 168–175, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[HG04] Erik Hatcher and Otis Gospodnetic. Lucene in Action (In Action series). Manning Publications Co., Greenwich, CT, USA, 2004.

[Jon72] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[Kir06] Atanas Kiryakov. OWLIM: balancing between scalable repository and lightweight reasoner. In *Proceedings of the 15th International World Wide Web Conference (WWW2006)*, 23–26 May 2010, Edinburgh, Scotland, 2006.

[Pir09] Peter Pirolli. Powers of 10: Modeling complex information-seeking systems at multiple scales. *IEEE Computer*, 42(3):33–40, 2009.

[RWJ+95] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, and Mike Gatford. Okapi at trec-3. *NIST SPECIAL PUBLICATION SP*, pages 109–109, 1995.

[TRCB13] Valentin Tablan, Ian Roberts, Hamish Cunningham, and Kalina Bontcheva. GATECloud.net: a Platform for Large-Scale, Open-Source Text Processing on the Cloud. *Philosophical Transactions of the Royal Society A: Mathematical, Physical & Engineering Sciences*, 371(1983):20120071, 2013.