# System Design, draft

Deliverable D2.1

Version FINAL

**Authors:** Zuhaitz Beloki[1], German Rigau[1], Aitor Soroa[1], Antske Fokkens[2], Piek Vossen[2], Marco Rospocher[3], Francesco Corcoglioniti[3], Roldano Cattoni[3], Thomas Ploeger[4], Willem Robert van Hage[4]

**Affiliation:** (1) EHU, (2) VUA, (3) FBK, (4) SYN

Building structured event indexes of large volumes of financial and economic data for decision making

ICT 316404

| | |
|---|---|
| **Grant Agreement No.** | 316404 |
| **Project Acronym** | NEWSREADER |
| **Project Full Title** | Building structured event indexes of large volumes of financial and economic data for decision making. |
| **Funding Scheme** | FP7-ICT-2011-8 |
| **Project Website** | http://www.newsreader-project.eu/ |
| **Project Coordinator** | Prof. dr. Piek T.J.M. Vossen<br>VU University Amsterdam<br>Tel. + 31 (0) 20 5986466<br>Fax. + 31 (0) 20 5986500<br>Email: piek.vossen@vu.nl |
| **Document Number** | Deliverable D2.1 |
| **Status & Version** | FINAL |
| **Contractual Date of Delivery** | December 2013 |
| **Actual Date of Delivery** | January 10, 2014 |
| **Type** | Report |
| **Security (distribution level)** | Public |
| **Number of Pages** | 115 |
| **WP Contributing to the Deliverable** | WP2 |
| **WP Responsible** | EHU |
| **EC Project Officer** | Susan Fraser |

**Authors:** Zuhaitz Beloki[1], German Rigau[1], Aitor Soroa[1], Antske Fokkens[2], Piek Vossen[2], Marco Rospocher[3], Francesco Corcoglioniti[3], Roldano Cattoni[3], Thomas Ploeger[4], Willem Robert van Hage[4]

**Keywords:** system design, big data, scaling NLP

**Abstract:** This deliverable describes the first version of the System Design framework developed in NewsReader to process large and continuous streams of English, Dutch, Spanish and Italian news articles. The deliverable describes the system architecture to automatically process massive streams of daily news in 4 different languages to reconstruct longer term story lines of events. The system relies on a central knowledge repository, called the KnowledgeStore, which is also described. Finally, we present the first version of the Decision Support System (DSS), which is meant to provide insight into the sequences of events that led up to a current situation so that a user can extrapolate to what might happen in the future.

## Table of Revisions

| Version | Date | Description and reason | By | Affected sections |
|---|---|---|---|---|
| 0.1 | November 2013 | Structure of the deliverable set | Aitor Soroa | All |
| 0.2 | December 2013 | First draft of the document | All Authors | All |
| 0.3 | December 2013 | Revised document | All Authors | All |
| 0.4 | January 2014 | Major revision | All Authors | All |

## Executive Summary

This deliverable describes the first cycle of tasks T02.1 “Overall architecture”, T02.2 “Data Structures and representation formats”, T02.3 “Software modules and APIs” and T02.4 “Scaling requirements” (25PM of effort). It shows the first prototype of the NewReader system for processing, integration and visualization of complex event structures extracted from a large set of documents. The deliverable describes a first approach for scalable NLP and presents a proposal for an architecture for streaming processing of huge textual data.

# Contents

# List of Tables

# 1 Introduction and scope

This deliverable describes the first version of the **System Design** framework developed in NewsReader to process large and continuous streams of English, Dutch, Spanish and Italian news articles. The goal of the NewsReader project[1] is to automatically process massive streams of daily news in 4 different languages to reconstruct longer term story lines of events. For this purpose, the project will extract events mentioned in the the news articles, the place and date of their occurrence and who is involved. Based on the extracted knowledge, NewsReader will reconstruct a coherent story in which new events are related to past events. The research activities conducted within the project strongly rely on the automatic detection of events, which are considered as the core information unit underlying news and therefore any decision making process that depends on news articles.

The NewsReader architecture for NLP processing integrates a large number of tools which relate to various research areas such as Information Retrieval and Extraction, Natural Language Processing, Text Mining or Natural Language Understanding. Within the project we distinguish two main types of linguistic processing: inter-document linguistic processing and cross document linguistic processing. Inter-document linguistic processing involves the linguistic processing of a single text document, and comprises steps such as tokenization, lemmatization, part-of-speech tagging, parsing, word sense disambiguation, Named Entity and Semantic Role Recognition for all the languages in NewsReader. Besides, Named entities are linked as much as possible to external sources such as Wikipedia and DBpedia. Details of the NLP modules used in the project are described in project deliverable D4.2 “Event Detection, version 1”[2]. We have defined a common annotation scheme, the NLP Annotation Format (NAF) which serves as a shared model that allows these NLP tools to cooperate. NAF is multi-layered, stand-off annotation scheme based on XML which allows the representation of a great variety of linguistic information in a common way. Specific input and output wrappers have been also developed or adapted to work with the new formats and APIs.

Cross document processing involves steps such as document clustering or linking textual expressions, i.e. mentions, from several documents which refer to the same entity or event, creating together chains of coreferring mentions. Events and their relations needs to be represented using formal semantic structures in an extremely compact and compressed form, eliminating duplication and repetition, detecting event identity, completing incomplete descriptions, and finally chaining and relating events into plots (temporal, local and causal chains). Abstraction over different mentions of the same events, participants, places and times results in a single representation of an instance with links to the places where it is mentioned in the news. We have designed an annotation format called Grounded Annotation Framework (GAF, [Fokkens *et al.*, 2013]), which formally distinguishes between mentions in NAF and instances in the Simple Event Model (SEM, [van Hage *et al.*, 2011]).

The NewsReader project will follow a streaming computing paradigm, where documents

[1] FP7-ICT-316404 ”Building structured event indexes of large volumes of financial and economic data for decision making”, www.newsreader-project.eu/

[2] `http://www.newsreader-project.eu/files/2012/12/NewsReader-316404-D4.2.pdf`

will arrive at any moment and have to be processed continuously. The project foresees the processing of huge amounts of textual data at any rate. In this deliverable we describe the scaling solution used in the first year of the project. We also report the results of analyzing circa 65.000 documents using the NewsReader system where the NLP modules are deployed on several machines and distributed among different places. We also describe the proposal for the second version of the architecture which fully implements a streaming computing architecture.

The result of the day-by-day processing of large volumes of news and information is stored in the KnowledgeStore central data repository, which plays a crucial role in the NewsReader architecture. We present the implementation of the first version of the KnowledgeStore, a framework that contributes to bridge the unstructured and structured worlds, enabling to jointly store, manage, retrieve, and semantically query its contents.

Finally, as a result of the NLP processing, the project will generate highly dense and dynamic structural data. The ultimate goal of the project is to assist professional decision makers to make well informed decisions, based on the knowledge NewsReader can offer to them. We present the first version of the Decision Support System (DSS), which is meant to provide insight into the sequences of events that led up to a current situation so that a user can extrapolate to what might happen in the future.

This deliverable is structured as follows. Section 2 describes the main requirements for all the components of the NewsReader architecture, including an study of state-of-the-art technologies on Big Data processing. Section 3 presents the main architecture of NewsReader, including the annotation formats used within the project. Section 4 shows the NewsReader pipeline implemented in the first year of the project, showing the results of the linguistic processing performed so far. The first version of the KnowledgeStore is described in Section 5, and Section 6 describes the Decision Support System. Finally, 7 draws some conclusions and future work.

# 2 System requirements

In this section, we will describe the system requirements of the NewsReader project. These requirements constitute the general desiderata that need to be fulfilled for the project, and will thus guide the design of the general architecture and main technological choices.

As stated in the introductory section, NewsReader aims to identify the event mentions across documents in four different languages: English, Dutch, Spanish and Italian. In addition, NewsReader will extract information about the event's participants, temporal constraints and locations. Factuality[3] of event mentions will also be extracted, as well as the authority of the source. Therefore, provenance information about all the knowledge extracted within the project has to be recorded.

Based on the extracted knowledge, NewsReader will reconstruct a coherent story in which new events are related to past events. Events and their relations will be represented using formal semantic structures in an extremely compact and compressed form, eliminating duplication and repetition, detecting event identity, completing incomplete descriptions, and finally chaining and relating events into plots (temporal, local and causal chains).

NewsReader will use the economic and financial domain for evaluation purposes. We foresee up to five-hundred-thousand news items and websites within this domain each day which can be potentially relevant for professionals working in this sector. The project has to process these data streams on a day-to-day basis using natural-language-processing (NLP) techniques to extract the economic-financial events from the text. We will process large volumes of news and information and store the outcome in a knowledge base, the KnowledgeStore, in which each event is unique, connected to time and place and connected to many other events.

Based on the above, we can summarize the main requirements as follows:

- NewsReader will show how large volumes of textual data can be processed within the given time-constraints and how the output can be stored efficiently in a database that captures the essential information about who, what, when and where. More specifically this involves:
  - Processing data in four languages, as well as storing event data (i.e. the outcome of event detection, processing and reasoning) using the KnowledgeStore technology.
  - Inter-connecting a large number of NLP modules for state-of-the-art event extraction modules so that they can be integrated into a common NLP processing architecture.
  - Handling long-term diachronic cumulation of events, integrating the new with the old.

[3]Factuality information is useful for recognizing whether the events mentioned in the text actually happened, did not happen, or there is some uncertainty about the event occurring or not.

  - Handling the daily stream of new incoming data, which involves the updating of the KnowledgeStore.
  - Demonstrate the scalability of the KnowledgeStore solution to eventually handle billions of documents in different languages.

- In order to be useful in the decision-making application scenario of the project, the content of the KnowledgeStore has to be accessible in an effective way. To achieve this goal, we will develop innovative visualization techniques for events, their internal structure and their relations to other events, that will graphically and adequately display the content of the KnowledgeStore. More specifically this will involve:
  - Querying and reasoning over huge and dense semantic graphs of data to obtain correct, complete and relevant results.
  - Combining the output of querying the KnowledgeStore with queries over structured data.
  - Visualization of complex event structures, which immediately shows the important implications and supports interaction and manipulation to access more/other data when needed.
  - Interaction of users with the visual representations to support their search for evidence in decision processes.

## 2.1 Big data requirements

Nowadays there is a continuous increase of computational power needs due to an overwhelming flow of textual data. This calls for a paradigm shift in computing architecture and large scale data processing. As mentioned above, NewsReader foresees a huge flow of news items per day which have to be processed in a reasonable time frame (one or few hours). The project faces thus an important challenge regarding the scalability of linguistic processing of texts.

The challenges NewsReader faces fall into a new class of the so called “Big Data” tasks. These tasks require large scale and intensive processing and must be able to scale efficiently to very big volumes of data [Yu and Chen, 2013; McCreadie *et al.*, 2013; Sakr *et al.*, 2013]. *MapReduce* [Dean and Ghemawat, 2008a] is a popular programming model framework designed to perform large scale computations. It is able to scale to thousand of nodes in a fault-tolerant manner. However, *MapReduce* follows a *batch* processing model, where computations start and end within a given time frame. Batch processing is not well suited to working with the form of underlying data the NewsReader project is faced with, due to their lack of responsiveness [Brito *et al.*, 2011].

*Streaming computing* [Alon *et al.*, 1996; Neumeyer *et al.*, 2010] represents an alternative programming model for dealing with a continuous flow of data (streams) which require very high levels of data throughput and a low level of response latency. This programming model assumes that data is presented to the algorithm as one or more input streams

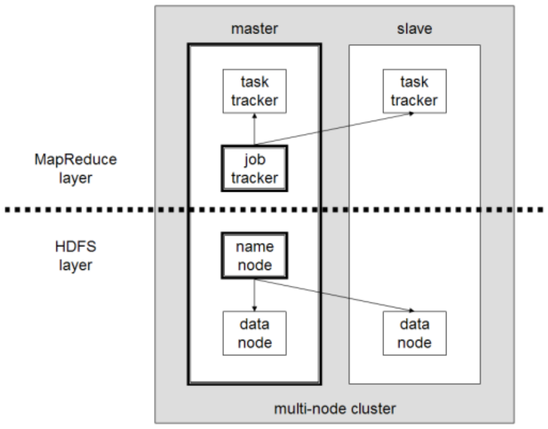


Figure 1: A Hadoop cluster.

that are processed in order, and only once. The NewsReader architecture fits under this programming model, as we expect NewsReader to continually receive news which have to be processed and integrated into the KnowledgeStore.

In the following sections, we will briefly analyze some of the most widely used frameworks for massive data processing. We will start describing the *Hadoop* framework for batch processing. *Hadoop* is not the architecture of choice for NewsReader but we describe it nonetheless because of its relevance in the big data ecosystem. We then focus on two of the most widely used frameworks for streaming computing, namely *Storm* and *Yahoo S4*. We finally describe —and justify— the main choice taken in the project regarding which framework to use.

## Hadoop

Hadoop is a framework for storage and large-scale processing of data sets on clusters of commodity hardware. It is an alternative, open-source, implementation of the *MapReduce* algorithm. A specific file system called Hadoop Distributed File System (HDFS) is also part of the project. HDFS is derived from the Google File System (GFS).

Hadoop programs are executed in clusters of computers that are inter-connected by switches. Computers that are connected by the same switch are located in the same rack.Two different type of nodes are found in a hadoop system: one single master node and multiple worker nodes. As Figure 1[4] shows, each node contains modules needed both by the *MapReduce* algorithm and the HDFS. The master node consists of four components. The JobTracker and NameNode control MapReduce and HDFS related processes, respectively. The DataNode and TaskTracker are responsible for storing data and executing MapReduce functions, respectively. Each worker node works as both a DataNode and a TaskTracker.

Different file systems have been used along with Hadoop, but HDFS is the most common system, since it was developed specifically for Hadoop.

[4]Taken from `http://bit.ly/19gNBd9`

## Hadoop MapReduce

*MapReduce* was designed and implemented by Google. Hadoop *MapReduce* is an open-source implementation which has been widely used during the last few years. This algorithm arose from the need of the Google company to run straightforward programs with very large input data sets. This led to designing a solution which runs their programs distributed across a large cluster of machines. The biggest part of the efforts focused on issues such as parallelization, distribution of data, synchronization between nodes, load balancing and fault tolerance. Therefore, a new library was designed, which would hide from the user all the logic about aforementioned issues, letting programmers concentrate their efforts on their application logic. One of the most important characteristic of *MapReduce* is that it fits well with batch processing systems, whereas it leads to serious problems for realtime streaming processing systems.

The user of the aforementioned library needs to implement two types of functions: *map* functions and *reduce* functions. A map function takes a key/value pair and produces a set of intermediate key/value pairs. Reduce functions receive all the intermediate pairs grouped by the key value, and produce new pairs as output. The final output is available in several output files, one for each reducer task. The algorithm is illustrated in Figure 2.[5]

For instance, consider the problem of counting the occurrences of each word in a document. Map functions would take each line of the document as the value and the offset of the beginning of the line as the key. Then, it would emit a new intermediate pair for each word in the line, consisting of the word itself and the number of occurrences in that line. Each reducer would take all the occurrences of a word, and would emit the sum of them.

The library is fault tolerant; it knows how to react when a worker node or even the master node fails. The master pings the workers periodically and, if a worker does not respond, it is marked as failed. When a worker fails, all the map tasks completed by the worker have to be re-executed, since their output is stored in the local disk and is therefore inaccessible. All tasks in progress are also reset. The master node writes checkpoints of its internal status periodically. When the master node fails, it will be restarted from the last checkpoint.

The file system stores several copies (3 by default) of each file-block across the cluster. To reduce bandwidth usage, the master node attempts to assign each map task to a worker containing one of the copies of the corresponding input data.

The user chooses the number of map and reduce tasks that will be created. It is therefore important to take into account what the granularity will be. As a general rule, it is better to have many more map and reduce tasks than machines in the cluster. This way it is easier to take advantage of the load balancing, and it is easier to migrate tasks when a node fails.

[5]Taken from [Dean and Ghemawat, 2008b]

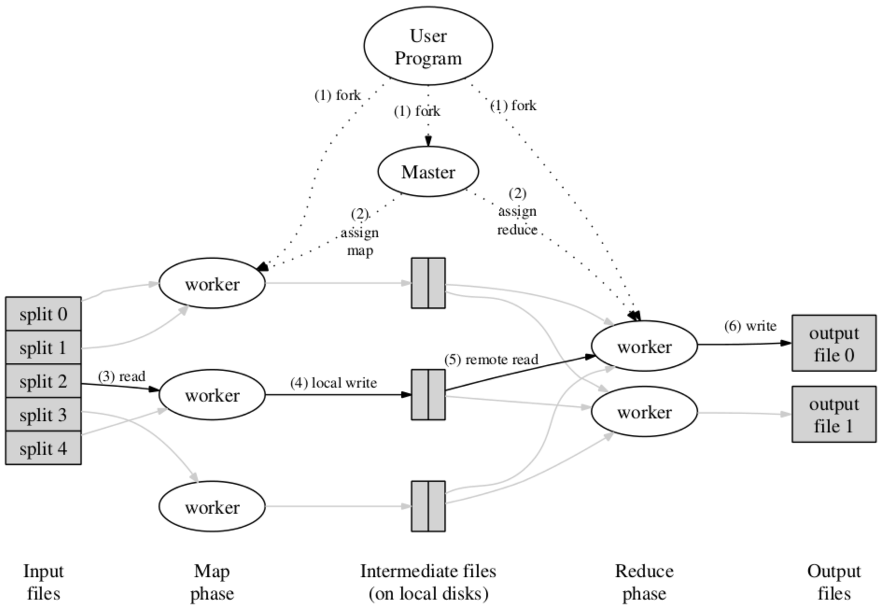


Figure 2: MapReduce execution overview.

## The Hadoop Distributed File System

When Google first implemented MapReduce, the Google File System (GFS) was used. Hadoop, instead, uses the Hadoop Distributed File System (HDFS), which is inspired by GFS. The HDFS, besides offering persistence, improves availability and data durability and helps returning to the desired status when a node or a rack fails, resulting in overall better performance. While the interface is similar to the UNIX file system, standards were sacrificed in favor of performance for applications.

The HDFS consists of two types of nodes and a client to access the file system. The NameNode is the main node, similar to the master node in MapReduce. This node stores the hierarchy of the directories and files in memory. Each file is split in blocks (typically 128 MB), and each block is replicated three times through the cluster. Replicated blocks are stored in different DataNodes. Besides, the file system tree, the NameNode keeps the mapping between file blocks and the DataNodes containing the file blocks in memory. Therefore, when the clients needs to read a file, it first has to ask the NameNode for the location of the DataNodes containing the blocks of the file.

The DataNodes are the nodes where all the data blocks are stored. To store a block, two physical files are needed, one for data and another one for metadata. NameNodes have to be subscribed to the NameNode. It is made by a handshake, a process where the NameNode is prepared to settle in the file system. The communication between nodes is made by sending heartbeats. The NameNode responds to the heartbeats sending instructions to the DataNodes. If the NameNode doesn't receive any heartbeat from a DataNode in a specific lapse of time, the latter is marked as failed.

The applications access the file system using the HDFS client. It supports operations for creating, reading and deleting files as well as creating and deleting directories. The client asks the NameNode for the locations of the DataNodes for each read/write operation.

When a new block is written, the HDFS stores three replicas of the same block. The replicas are stored in different machines in different racks. This improves availability and recoverability when a given node or even an entire rack fails. A balancer tool is offered for the situations when the stored data is irregularly distributed across the cluster, for instance because new machines have been added after initializing the cluster.

### S4: Distributed Stream Computing Platform

S4 is an open source, general-purpose, distributed, scalable and partially fault-tolerant platform for developing distributed programs for processing continuous streams of data. This engine was inspired by the *MapReduce* algorithm, but oriented for streaming processing. An attempt to adapt Hadoop for an application where large realtime streams of data were received failed, and it was concluded that a library that would work for both batch and stream processing was not viable. S4 offers the flexibility to deploy new algorithms as needed in research environments, while scalability and high availability requested by production environments are taken into account.

The main units in the design of this system are the Processing Elements (PEs). The PEs encapsulate the functionality of each logical piece of processing. The only way of communication between PEs is by sending messages, making the system derive from a combination of MapReduce and the Actors model. A high level of encapsulation and transparency is achieved by this model, resulting in a high level of simplicity.

Processing Elements are defined by the following four features: its functionality, the type of events it consumes, the keyed attribute in those events and the value of the keyed attribute. Each PE consumes all the events that fulfill the mentioned features. Special PEs are available, with no key defined, which consume all the events of the corresponding type. There are several ready-to-use PEs available with different functionalities (sort, join, filter...). Creating custom PEs is easy and simple. Since stream computing processes do never end (unless the user kills them), the PEs are created with a given amount of time to live. After the specified period of time is expired, the PE is eligible for removal.

The Processing Nodes (PNs) are the logical containers of the PEs. As shown in Figure 3,[6] these nodes make use of the communication layer to listen to events and dispatch new events. The communication layer is an abstraction layer that manages the cluster and enables the PNs to communicate between them being unaware of physical nodes. ZooKeeper is used for communication inside the cluster.

S4 lacks a cluster balancing system, making the system unbalance over time.

[6] Taken from [Neumeyer *et al.*, 2010]

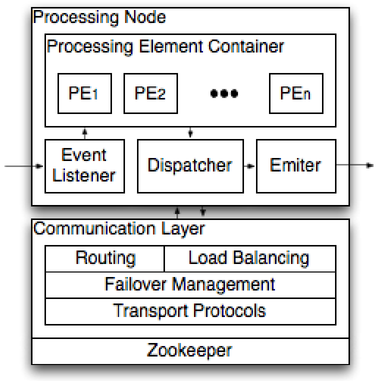


Figure 3: Internal structure of a processing node in S4.

### Storm

Storm was created to satisfy the needs of a distributed and scalable realtime computation framework. Previous existing similar frameworks were batch computing oriented, and were not suitable for stream oriented computing. The design goals followed by Storm are the follows:

- Make the design friendly and easy to understand.
- Provide a simple Programming Interface for processing data streams.
- Design a scalable cluster with high availability using commodity hardware.
- Minimize latency supporting local memory reads and avoiding disk I/O bottlenecks.

Systems implemented with Storm are easily scalable by adding new commodity hardware to the cluster. There is no need to change the algorithms. In the words of its creators "Storm's small set of primitives satisfy a stunning number of use cases."[7]. Whilst being similar to S4, one of the biggest differences between them is that Storm guarantees that no data will be lost. Compared to Hadoop, Storm is easier and simpler to use. Other Storm features are failure tolerance and the possibility of programming modules written in any programming language available.

The main abstraction structure of Storm is the topology. The topology represents the logical graph of the application. Each node of the graph is a processing component for a given task, while the edges are the paths each data-tuple makes. Input data comes from one or more data streams represented each of them as a sequence of tuples. Two types of processing components can be found in a Storm topology: spouts and bolts. Spouts are commonly the first component taking part in a topology. A spout creates the stream, an unbounded sequence of tuples, and sends them to the next component in the topology.

[7]`http://bit.ly/18TEteL`, accessed December 12, 2013.

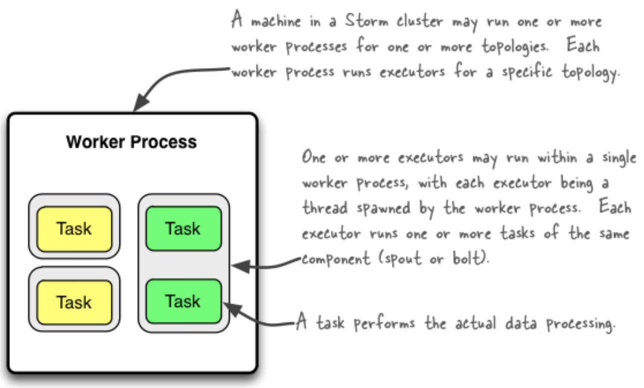

A machine in a Storm cluster may run one or more worker processes for one or more topologies. Each worker process runs executors for a specific topology.
Worker Process
Task
Task
Task
Task
One or more executors may run within a single worker process, with each executor being a thread spawned by the worker process. Each executor runs one or more tasks of the same component (spout or bolt).
A task performs the actual data processing.

Figure 4: A Storm worker process containing several executors and tasks.

The other components are the bolts. Bolts are the most common processing components in topologies. They take input tuples sent by spouts or other bolts, and emit new output tuples to the next bolt.

When it comes to the issue of cluster management, Storm uses a centralized model like Hadoop. There is a master node, called Nimbus, and multiple worker nodes, known as Supervisors. The Nimbus is responsible for creating Supervisor instances through the cluster and assigning a task or a set of tasks to each of them. It is also its job to monitor the cluster for failures. Supervisors manage all the input and output events of a worker node and starts/stops task processes as necessary. Storm, like S4, also uses ZooKeeper to manage communication inside the cluster.

Three different types of entities are distinguished in Storm: worker processes, executors and tasks. Worker processes are the logical containers for components. Each worker is physically a single JVM and contains part of the topology. An executor is a thread spawned by the corresponding worker process. It may run one or more tasks for a spout or bolt in the topology. A task is an instance of a spout or a bolt. Each spout or bolt can have several copies across the cluster. By default one single task is executed per executor, though it is a user-configurable value. Figure 4[8] shows the relation between worker processes, executors and tasks.

A Storm topology processes data as it comes, as it is a realtime computing framework. Therefore, there is no file system nor any kind of persistence system offered along with the framework. If persistence is needed by an application, external NoSQL databases like Cassandra or Mongo DB are common solutions.

Within the NewsReader project we decided to use the Storm framework for implementing the scalable architecture. The reasons which lead us to take this decision are the following:

- Storm follows a streaming programming paradigm instead of a *batch* type processing. As NewsReader will process documents continuously on a daily basis, this require-

[8]Taken from `http://bit.ly/19gO139`

ment is a must.

- Storm integrates easily with “NoSQL” type of databases, such as HBase and Cassandra. Because one of the main components of the KnowledgeStore is precisely the HBase component, we foresee an easy integration between the Storm pipeline and the KnowledgeStore.
- Storm supports a large number of programming languages, unlike Hadoop or S4, which only support packages programmed in Java. Therefore, Storm offers great flexibility to integrate a great variety of NLP components.
- Storm is a mature framework with a growing and vibrating user-base. For instance, while the S4 mailing list[9] contains about 500 messages in total, the Storm mailing[10] list has more than 5.000.

Section 4 describes the use of the Storm framework in the NewsReader architecture for the processing in the first year, as well as the design of a fully parallel Storm pipeline designed for the second year of the project.

## 2.2 Linguistic processing requirements

NewsReader needs deep linguistic processing to achieve its main goals. Event extraction is a complex task which involves various areas of Natural Language Processing and Text Mining. NewsReader aims to make a leap and integrate current state-of-the-art NLP processing tools into a common platform. Besides, the NewsReader project has to process large volumes of textual data within tight time constrains.

Within the NewsReader project, we distinguish two types of linguistic processing, namely, inter-document and cross-document processing. Inter document processing involves the linguistic processing of a single document, and comprises modules which range from tokenization to event coreference or factuality analysis. Cross document processing involves dealing with multiple documents and comprises tasks such as clustering and cross-document event coreference.

From a system design point of view, the linguistic processing requirements lead to the following questions which have to be addressed:

- Design a linguistic annotation format to allow for interoperability among the NLP processors. The format should be able to integrate a wide range of linguistic information (named entities, event coreference, semantic roles, opinions, etc.) and use RDF compatible representations whenever possible to facilitate communication with the KnowledgeStore. Besides, the format should be a suitable format to work on a large scale, distributed and parallel data processing NLP environment which can process thousand of documents every day. Finally, it should be easy to integrate other kind

[9]`http://mail-archives.apache.org/mod_mbox/incubator-s4-user/`
[10]`https://groups.google.com/d/forum/storm-user`

of information in the format, so that the tools used in NewsReader can be integrated directly in other pipelines for other projects.

- According to project deliverable D4.2 "Event Detection, version 1",[11] the NLP modules integrated into the NewsReader system should fulfill the following requirements:
  - **Modular**: Unlike other NLP toolkits, which often are built in a monolithic architecture, the modules should follow a data centric architecture so that modules can be picked and changed (even from other NLP toolkits). The modules should behave like Unix pipes, taking standard input, do some annotation, and produce standard output which in turn is the input for the next module.
  - **Efficient and Accurate**: The modules should be as efficient as possible, both in processing time as well as in the quality of the annotations they produce.
  - **Multilingual**: When possible, NLP modules should be parametrizable as to allow for linguistic processing on more than one language.
- Analyze the optimal implementations and configurations of our modules: grouped as virtual machines, interacting with the KnowledgeStore, using client-server architectures for modules that use large background models, etc.
- Decide the optimal design for processing documents and sentences from documents in parallel, exploiting the non-dependencies of certain modules.

## 2.3 KnowledgeStore requirements

As mentioned in the previous sections, NewsReader needs to efficiently store all the output produced by the linguistic processing tools in such a way that all the essential information about events and related information (who, what, when and where) can be easily and effectively accessed. As NewsReader aims to deal with billions of documents in different languages, efficient storage and effective retrieval are indeed key factors for its success.

The place were all the content is stored and can be accessed is the KnowledgeStore. A brief introduction to the KnowledgeStore is provided in Section 5, while a more detailed description is available in Deliverables D6.1 and D6.2.1.

From a system design point of view, the KnowledgeStore impose the following requirements on the NewsReader system infrastructure:

- (scalability) Given the number of documents and semantic content expected to be handled by the KnowledgeStore, a distributed infrastructure is required. In particular, given the typology of resources to be handled, and the kind of access to them, an IT ecosystem composed of
  - a Hadoop cluster

[11] `http://www.newsreader-project.eu/files/2012/12/NewsReader-316404-D4.2.pdf`

  - a HBase cluster,
  - a triple store server (Virtuoso), and
  - a KnowledgeStore front-end server that orchestrate the storing of data among the first three components,

  is foreseen[12].

- (accessibility) The KnowledgeStore is fundamentally a storage server, therefore the interaction with the other modules of the system should occur according to a client-server paradigm. In particular, a network infrastructure is needed so that the various modules can access the HTTP ReST API offered by the KnowledgeStore.
- (querying and reasoning) To achieve its goals, NewsReader needs to check the consistency, completeness and factuality of the events and events related information extracted from the documents. This requires the KnowledgeStore to support queries and reasoning over a huge and dense semantic content—stored according to Semantic Web best practices—which in turn requires the availability in the IT infrastructure of a triple store server (e.g. Virtuoso)—a state of the art storage server enabling querying and reasoning over Semantic Web content—supporting the aforementioned tasks.
- (performance) As the NewsReader system is expected to (i) process a vast quantity of daily news, and (ii) support online access through the decision support tool suite, the KnowledgeStore has stringent requirements on the system infrastructure to be set up. Indeed, to achieve the expected performances, robust hardware resources are needed. In particular, the machines in the IT infrastructure has to be equipped with fast disks (in particular to support the fast loading and retrieval of large quantity of data), large memory (especially to support efficient querying of the triple store server), and fast network communication (to support bottleneck-less communication among the different clustered machines).

[12]More details are provided in Section 5

# 3 General Architecture

This section provides an overview of the general architecture in NewsReader. We describe the relative order of processes and how different components of the architecture communicate with each other. Representation formats play a major role in the communication between components. We have developed two new representation formats and extended an existing representation model to fulfill the requirements for NewsReader. We will describe both the overall setup and the representation formats employed by NewsReader.

In Section 2, we described the system requirements for NewsReader. Our architecture must have the following properties in order to fulfill these requirements:

1. **Modularity**. Extracting events and establishing relations between them is a complex task. Several NLP tasks need to executed to come to our final result and we aim to do this for four languages. It should be easy to integrate these modules in the overall architecture. It should also be easy to exchange modules, either for language indepedendent components to communicate with language specific components of different languages or to improve the overall outcome by experimenting with more than one state-of-the-art module for a specific task.

2. **Exchange**. It should not only be easy to exchange information between NLP modules, but also between NLP modules and other resources. This applies in particular to the KnowledgeStore. We want to cumulate information about events over time. In order to do so, certain NLP modules, such as modules looking at event identification or named entity disambiguation, need to be able to communicate with the KnowledgeStore. This allows us to use previously extracted information to improve processing of new information.

3. **Scalability**. The idea behind NewsReader is that it is a news recorder which interprets and stores all the news. The system should be designed in such a way that it is easy to scale it up to the amount of data that must be processed in such a scenario (LexisNexis estimates around 2 million documents per working day). The same applies to the querying, reasoning and visualization modules that interact with the KnowledgeStore. These modules should be able to handle vast amounts of data.

4. **Portability**. It should be easy to run the complete architecture at different locations. This is both needed to divide the processing load over different partners within NewsReader as to allow third parties to experiment with our setup.

This section is structured as follows. Section 3.1 presents an overview of the architecture of the entire process from the input data to the KnowledgeStore. It describes the relative order of steps, possibilities of parallization and communication between the KnowledgeStore and NLP modules. In Section 3.2, we introduce the NLP Annotation Format (NAF), the representation format we use during NLP processing. In order to represent events, their participants, locations, times and relations between them, we extended the Simple Event

Model (SEM). This extended SEM (SEM+) is described in Section 3.3. SEM+ instances and relations are related to analyses represented in NAF using the Grounded Annotation Framework [Fokkens *et al.*, 2013] (GAF). Section 3.4 provides an overall description of this framework. In addition to information obtained by processing text, information from structured data can be added to the KnowledgeStore.

## 3.1 A modular architecture

This section describes the overall architecture used in NewsReader. First, we will address how NLP modules are packed in virtual machines (henceforth VM(s)). This decision is based on (among others) interactions between modules, the basic units their processing applies to and their dependencies (apart from those on other modules in our pipeline). We will present these properties for NLP modules that are either used or likely to be used in NewsReader after addressing VMs. First, a full overview of the modules that are currently considered for NewsReader and planned setups are explained. This is followed by a description of interactions between modules and the Knowledge Store. Finally, we describe the setup that is currently used for processing.

### 3.1.1 Virtual Machines

Individual NLP modules may have different dependencies and installation requirements. Besides, within the project we want to be able to replicate the result of NLP modules, that is, one NLP module applied to a particular input text has to produce the same output regardless the software framework (machine, operating system, etc.) where it is installed. We thus pack the NLP modules into virtual machines (VMs) in order to handle this and hence facilitate portability of the modules. Using VMs is a common practice among cloud computing solutions having to deal with big quantities of data, and thus they give us a proper foundation to build the processing architecture upon.

### 3.1.2 A flexible architecture

NLP modules are grouped on the basis of compatible settings and configuration and sequential dependencies for processing texts. Much can be gained in efficiency by applying parallel processing where possible. This helps us to avoid bottlenecks around modules that need more processing time than others. In NewsReader, we employ modules that work on a sentence level, a document level and cross-document or cluster level. The level of operation determines to what extend parallel processing can be applied.

Another factor that influences how we can group NLP modules is the dependency between modules in the system. We will use the term *NLP dependencies* to refer to dependencies between modules in our system. When talking about NLP dependencies, we mean that a given module needs the output of another module as its input. This should be distinguished from the usual case of dependency or coupling in software engineering where a program module relies on other modules themselves and not on whether these

| **module** | **unit** | **input** | **output** |
|---|---|---|---|
| IXA tokenizer<br>Stanford tokenizer<br>TokenPro | document<br>document<br>document | raw text<br>raw text<br>raw text | sentences, tokens<br>sentences, tokens<br>sentences, tokens, normalized tokens, char position of tokens |
| IXA PoS tagger<br>Stanford PoS tagger<br>TextPro Pos tagger | sentence<br>sentence<br>sentence | tokens<br>tokens<br>tokens | lemmas, PoS-tags<br>lemmas, PoS-tags<br>lemmas, PoS-tags |
| VUA WM tagger | sentence | tokens, lemmas, PoS | multiwords |
| IXA Parser<br>Mate Parser<br>Stanford Parser<br>ChunkPro Parser | sentence<br>sentence<br>sentence<br>sentence | raw text/lemmas, PoS<br>raw text/lemmas, PoS<br>raw text/lemmas, PoS<br>raw text/tokens, PoS | constituents<br>dependencies<br>dependencies, PS-trees,<br>chunks |
| TimePro | sentence | tokens, PoS, chunks | Timex3 |
| IXA NER<br>EntityPro NER | sentence<br>sentence | lemmas, PoS<br>tokens, lemmas, PoS | named entities<br>named entities |
| UKB-WSD<br>SVM-WSD | sentence<br>sentence | lemmas, PoS<br>tokens, terms (= lemma + PoS) | synsets<br>tokens & terms with word sense information |
| Spotlight NED | sentence | named entities (NEs) | disambiguated NEs |
| Mate SRL<br>Event classification | sentence<br>sentence | lemmas, PoS, dependencies<br>terms, dependencies, time and location entities | semantic roles<br>predicates, roles |
| Graph-based coref<br><br>Entity coreference<br>Event coreference<br>Toponym resolution | document<br><br>document<br>document<br>?? | lemmas, PoS, NE, constituents<br>entities, lemmas<br>semantic roles, lemmas<br>tokens, NE | entity coreferences<br><br>entity coreferences<br>event coreferences<br>toponym coordinates |
| VUA Factuality | sentence | tokens, PoS | factuality |
| VUA Discourse | document | NITF (raw input) | document structure |
| VUA opinion miner | sentence | tokens, terms, factuality, entities, constituents, dependencies | opinions |
| Aggregation | cluster | lemmas, Timex3<br>semantic roles | coreferences, event-participant relations |

Table 1: NewsReader modules for English and their properties

modules have been run on the data prior to running the module at hand. Coupling can influence the possibility of placing modules in the same VM, because two modules may have incompatible dependencies, in which case they cannot be placed in the same VM. Since this is not a core factor in our research, we will not further elaborate on this issue here.

The aforementioned NLP dependencies are of a more flexible nature than cases of coupling. Typically, a NLP module needs certain information about the data in order to work properly. This means that this information should be identified before the module applies and presented in a format that the module can read. In other words, when we talk about a NLP dependency of a module, the module does not really depend on another module, but rather on receiving specific input. In our scenario, where our input is raw data newspaper text, this generally means that some module that provides this required input must apply before the NLP module applies. NLP dependencies thus indicate the relative order in which specific modules should apply.

The level of operation and NLP dependencies thus have an impact on the flexibility of our system architecture. Flexibility provides possibilities of optimizing NLP processing. We thus aim for maximal flexibility in our setup. Table 1 provides basic information about modules either included in or intended for the NewsReader architecture. The column *Module* provides the name of the module, *unit* stands for level of operation, *input* indicates the information the module requires and *output* represents the information the module provides. A more elaborate description of individual modules can be found in Deliverable 4.2.1 [Agerri *et al.*, 2013]. The modules described in Table 1 all work for English. Alternative modules for other languages will have the same *unit* and *input* requirements in most cases or else they will be very similar.

As mentioned above, the output of the tools is represented in NAF. This is a layered, extensible format where each tool incrementally adds its output while maintaining all information that was present in its input. If, for instance, a module needs named entities, lemmas and PoS tags, we can relate its position in the architecture to the named entity recognizers. The named entity recognizers also require lemmas and PoS tags, so their output will always include this information.

Figure 5 represents a basic architecture taking the input requirements of individual modules provided in Table 1 into account. The arrows in the figure represent NLP dependencies, which can either apply to an individual module or to all modules included in a group (represented by blocks in dashed lines). If more than one black arrow points to a specific module or group, the output of all preceding modules is required. The green arrow from raw text to the parser indicates a short cut: most parsers we work with can either take raw text or tokenized text with PoS as their input. The discourse module is not included in this picture, because there are no dependencies between this module and the other modules of the system. It takes raw text as an input and none of the NLP modules make use of its output. It can therefore be applied at any stage in the process.

Modules that are placed in the same group can be executed in parallel. The named entity recognizers, parsers, factuality module and word sense disambiguators all take tokens, lemmas and PoS as their input. The named entity recognizer provides the necessary input for resolving entities and determining coreference between them. The parsers provide input for identifying time expressions (chunks) and semantic role labelling (dependencies), which enables event coreference. Because the aforementioned modules requiring named entities do not interact with those requiring syntactic information, these processes can also be parallelized. Finally, event classification and the graph-based entity coreference module combine information from the named entity recognizers and the parsers. The module that identifies opinions also takes the output of the factuality module into account. Aggregation cannot be parallelized with other processes, since it applies to a cluster of documents.

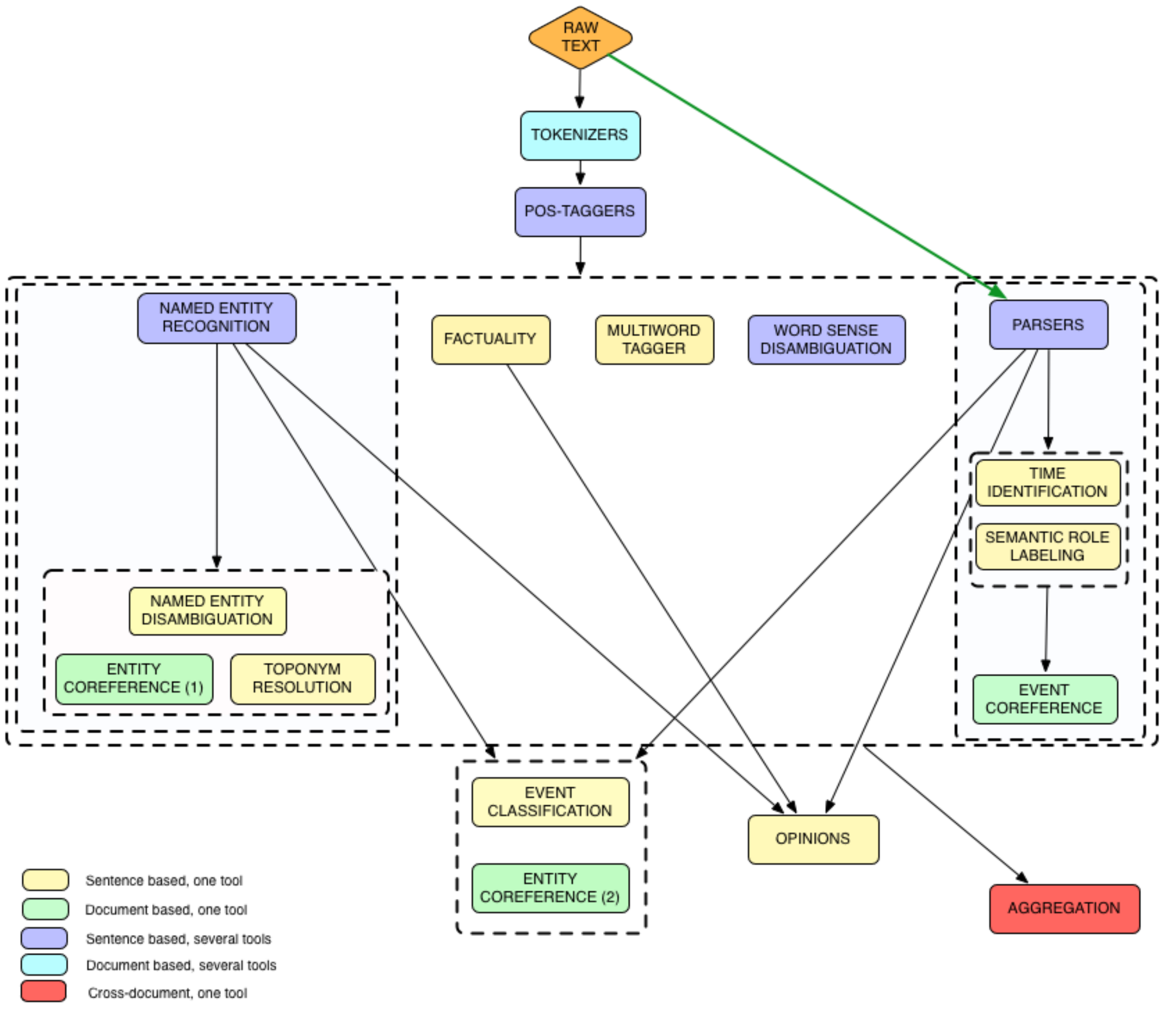


Figure 5: Application order of modules (documents as smallest units)

The architecture presented in Figure 5 includes modules that can run on sentences, complete documents and document clusters. If we follow this architecture, we assume that modules that can be run on a sentence level also take complete documents as their input. Modules that need complete documents as their input cannot be placed in a group where parallel processing is applied on a sentence level. Figure 6 represents an alternative architecture which separates modules that take documents as their basic input unit from those that can work on a sentence level. This architecture is less flexible in the permitted order of application of individual modules, but can nevertheless lead to more efficient results. The efficiency of parsers is highly dependent on the length and complexity of the input sentence. Optimizing processing on a sentence level can therefore lead to a significant gain in efficiency. This may outweight what is lost by the additional restrictions separating sentence level modules from document level modules.

The question of which architecture leads to the overall best results needs to be answered

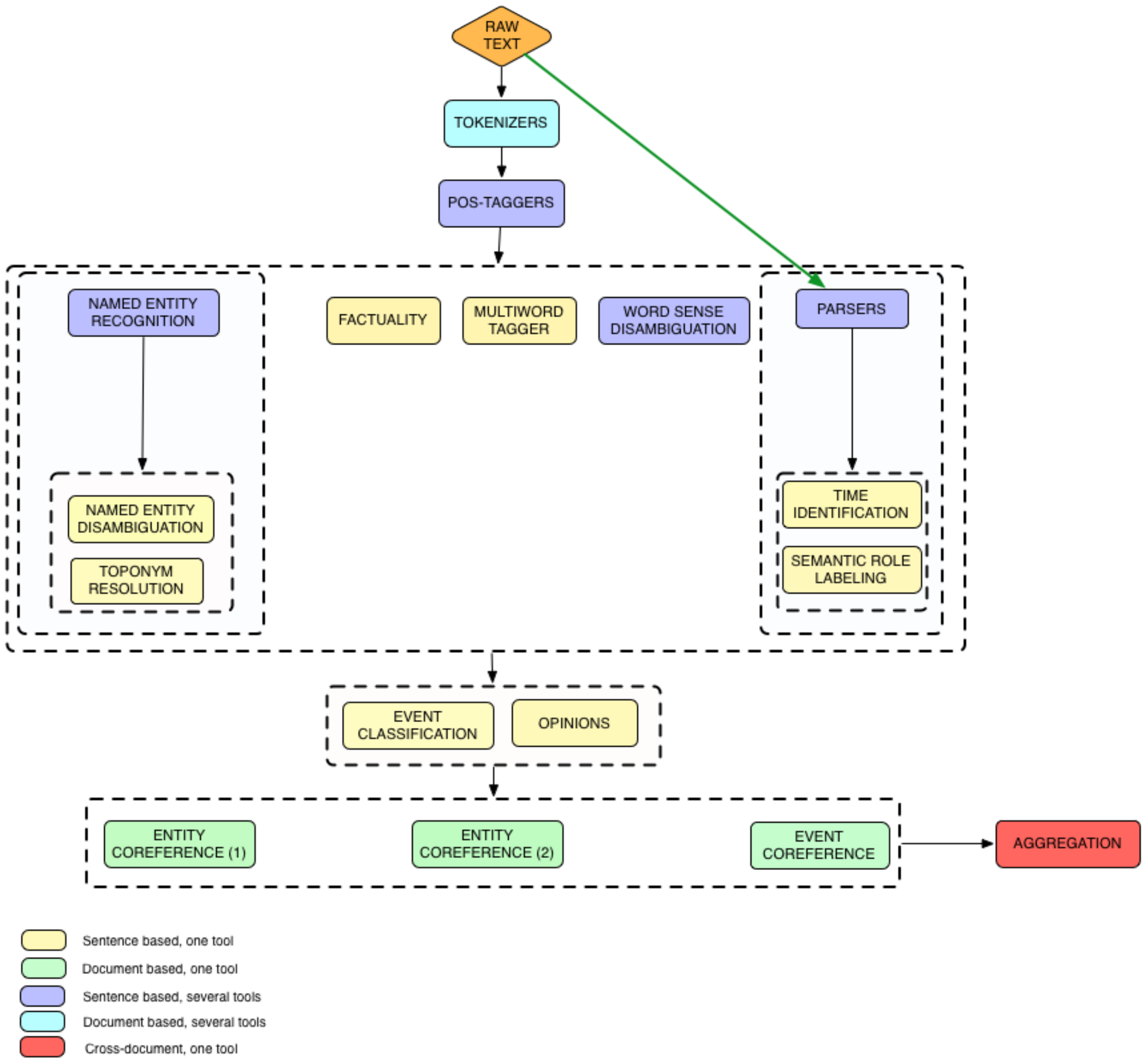


Figure 6: Application order of modules (sentences as smallest units)

empirically. It is dependent on a number of factors including the choice of modules for specific tasks. The choice of modules and whether performance improves when the output of more than one module is combined are research questions that require extensive empirical investigation themselves. In NewsReader, we will consider the two architectures outlined above as possible alternatives.

### 3.1.3 Interaction with the Knowledge Store

Figure 7 provides a simplified sketch of our overall architecture. The KnowledgeStore is placed at the center: it is the place where incoming documents and results of NLP

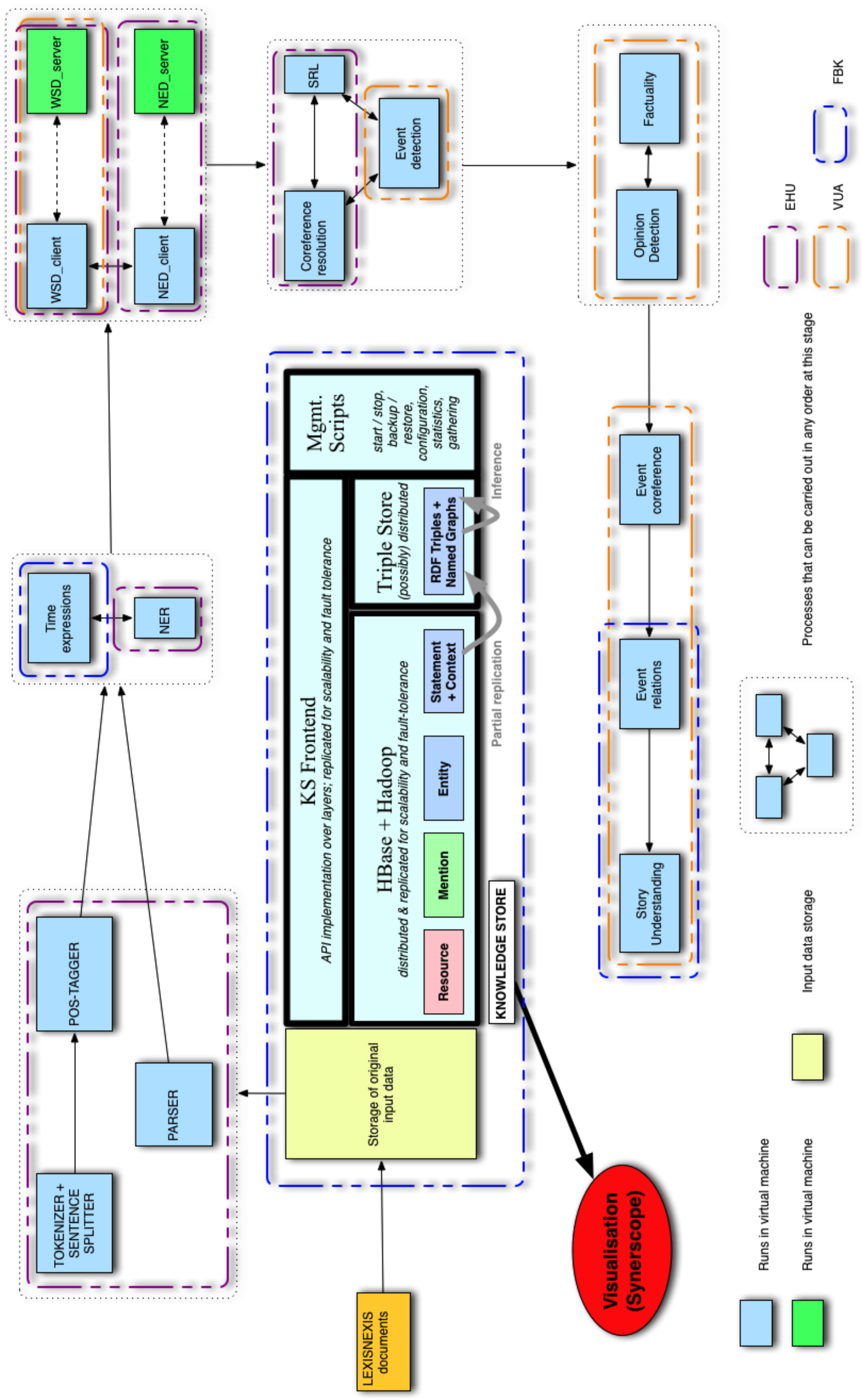


Figure 7: Simplified representation of NewsReader's system architecture

processing are stored. At several stages, there may be interaction between modules and

| **Modules** | **KS components** |
|---|---|
| Tokenization and sentence detection<br>PoS tagger<br>Parser | Resource |
| ?Time expressions<br>Named Entity Recognition | Mention, Entity |
| ?WSD client<br>NED client | Mention, Entity |
| Coreference resolution (entities)<br>Semantic Role Labeling<br>Event detection | Mention, Entity |
| Sentiment analysis<br>Factuality | Statement + Context |
| ?Event coreference<br>Event relations<br>Story understanding | Entity |

Table 2: Overview of interaction between the KnowledgeStore and NLP modules

| **task** | **module** | **task** | **module** |
|---|---|---|---|
| Tokenizing | IXA tokenizer | NE disambiguation | Spotlight NED |
| PoS tagging | IXA PoS tagger | Dependencies | Mate dependency parser |
| Multiword tagger | VUA MW tagger | Semantic roles | Mate semantic role labeler |
| NE recoginition | IXA NER | TimeX | TimePro |
| Opinion mining | VUA Opinion miner | Event coreference | VUA event corefence (lb) |
| WSD | VUA WSD | Factuality determiner | VUA factuality |

Table 3: Overview of tasks and modules included in IXA pipeline

the Knowledge Store, but any transaction can be postponed by stream-in and stream-out processes between modules directly.

Each annotation in the KnowledgeStore will have a pointer to the NAF annotation it was generated from (provenance marking). NAF annotations will most likely also be stored in the Knowledge Store, where we estimate an additional 100 GB is needed for the 1M news and annotations outlined above. Table 2 indicates the interaction between modules and individual components of the KnowledgeStore. Modules are grouped according to the stage in which they take place. The Knowledge Store is described in more detail in Section 5.

#### 3.1.4 Architecture of the baseline system

As mentioned above, the previous sections outlined the full architecture planned for NewsReader. The basic system we are using for processing at the moment is a pipeline that does not exploit the possibilities of parallization yet. NewsReader started processing the news in November 2013. This subsection describes the architecture of the pipeline that was used for processing the first 66,000 documents. The basic system is simpler than the setup described above in two aspects. First, for each task one module is selected. Second, the system does not explore possibilities of optimizing the process through parallelization. In most cases where alternative modules were avalaible, the IXA module is used in this first setup. We therefore use the term *IXA pipeline* to refer to the basic system.

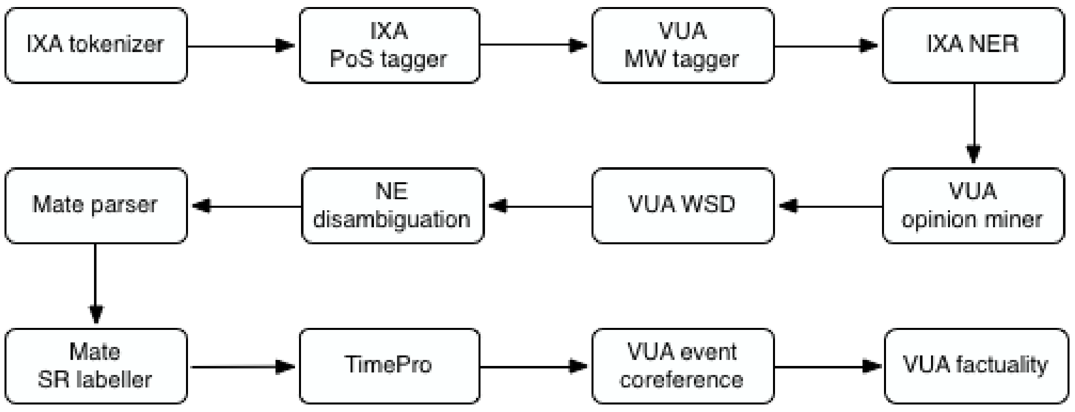


Figure 8: The IXA pipeline

The tasks that are supported and their modules are listed in Table 3. Figure 8 represents the IXA pipeline with all modules in order of application. Both Table 3 and Figure 8 only represent the modules for document processing. The output of the pipeline is used as input for the VUA aggregation module, which identifies cross document event and entity coreference. It should be noted that baseline systems are used for some tasks. Notably, the opinion miner that is currently used does not take factuality into account yet, which is why it can be applied before the factuality module. Furthermore, the current event coreference module is a simple baseline system which uses lemmas to establish coreference. In this case, the position in the pipeline need not change when a more sophisticated module is used, because all information needed by the more advanced implementation is already available to the current module in the pipeline.

The IXA pipeline as presented above will serve as a baseline for further experiments. Currently, more advanced modules are being intregrated including the Graph-based coreference module and a newly developed module for identifying and normalizing temporal expressions. After testing several alternative pipelines, we will experiment with combining alternative modules that perform the same task and optimizing processing through parallelization as described above.

| LAF requirements | NIF requirements |
|---|---|
| Expressive adequacy | Compatibility with RDF |
| Media independence | Coverage |
| Semantic adequacy | Structural interoperability |
| Incrementality | Conceptual interoperability |
| Uniformity | Granularity |
| Openness | Provenance and Confidence |
| Extensibility | Simplicity |
| Human readability | Scalability |
| Processability | |
| Consistency | |

Table 4: Requirements defined for LAF and NIF.

## 3.2 NAF

NewsReader makes use of a wide variety of NLP tools that are used in a complex architecture. We aim at improving our results by combining existing tools with tools developed especially as part of the NewsReader project. All these tools need to be able to communicate to each other. We therefore developed the NLP Annotation Format. The motivation behind this format, its desiderata and how it fulfills these are presented in this section.

### 3.2.1 Motivation

As the number of available NLP tools increases and they are used in more and more complex architectures, awareness of the importance of standardization rises ([Ide *et al.*, 2003], [Bosma *et al.*, 2009], [Hellmann *et al.*, 2013], among others). One of the main challenges lies in the fact that linguistic annotations as well as the output of NLP tools can be based on different theories or insights which each may have their own strengths.

Standardization efforts must therefore bring these variations together without compromising the richness of the individual output of different tools. In this section, we will explain how this is carried out by combining strengths of a format based on the Linguistic Annotation Format [Ide *et al.*, 2003, LAF] with those of the NLP Interchange Format [Hellmann *et al.*, 2013, NIF]. In order to support our goals, NAF carries the role of RDF even further than currently done in NIF. Table 4 lists the requirements defined for LAF and NIF. In fact, NAF fulfills all of these requirements. For reasons of space, we will limit ourselves to indicating properties of NAF related to these requirements in **bold font**. The main idea behind NAF is that it combines advantages of LAF-related formats and NIF. As such, it provides a framework that fulfills requirements imposed by NewsReader and other projects involving complex NLP architectures and Linked Data. It can therefore be adopted in various other NLP projects.

NAF is based on Knowledge Annotation Format [Bosma *et al.*, 2009, KAF]. This format follows the main principles of LAF as outlined in [Ide *et al.*, 2003]. Like LAF, KAF aims at **maximum flexibility**, **processing efficiency** and **reusability**. It is a layered, **extensible** format where each tool **incrementally** adds its output while maintaining all information that was present in its input. KAF has shown to be suitable for a complex pipeline combining tools developed at different sites in the KYOTO project. For more recent projects, however, some additional properties are required. First, as explained above, within the NewsReader project, we aim to extract storylines and store them as RDF triples in the KnowledgeStore. The information in the KnowledgeStore will also be used to support linguistic processing (e.g. for disambiguation). Second, we have more than one tool for several of the steps in the pipeline. Ideally the output of these tools should be combined. Third, we delay decisions as much as possible. Instead of only providing the output that received the highest score, we will include several possible outcomes per tool with their confidence scores.

These three desiderata can be addressed by using **RDF conforming** representations as shown by NIF. NIF is a RDF compliant format for linguistic information that is designed to accommodate the constantly increasing wide variety of NLP tools [Hellmann *et al.*, 2013] and can include information on provenance and confidence. Both provenance and confidence score indications are essential when combining the output of different tools. A drawback of NIF is that it does not seem to be a practical format for internal use of NLP tools. This assumption is confirmed by [Hellmann *et al.*, 2013]'s own user evaluation of the format.

NAF combines the strengths of KAF and NIF by using **Uniform Resource Identifiers** as much as possible in a representation that in other aspects follows the LAF recommendations. It is suitable to be used in NLP tools and offers the means to combine outcome of alternative tools while indicating **provenance and confidence scores**, as also offered by NIF. We even take the idea of using RDF conform representations a step further than NIF and also encourage to use URIs to refer to linguistic properties and values. The next section will elaborate on the main advantages of this extensive use of RDF compliant representations.

### 3.2.2 RDF in linguistic representations

RDF is a useful data model for NAF due to several reasons. This section will list the main reasons and explain how they support the desiderata outlined in the previous section.

First, RDF is by nature a graph model, which makes declarative specification of dependency patterns easy, for instance in SPARQL. Triple stores are typically optimized for queries that require multiple joins. That makes evaluation of dependency graph queries, which are typically long branched chains, efficient. This facilitates the communication between the KS and linguistic processing tools.

Second, RDF uses URIs for identification and URIs are not limited to the scope of a document, but have a global validity. This makes it easy to represent coreference relations across documents as done in the Grounded Annotation Framework [Fokkens *et al.*, 2013, GAF]. In GAF, formal representations of instances can be linked to one or more mentions of this instance in text, hence indicating which mentions corefer to this instance. A similar approach is taken to model the relations between instances: e.g., we can indicate that a semantic role label between the mention of a participant and the mention of an event is the mention of the relation between the event and this participant.

Third, RDF forms the basis on which RDFS and OWL ontology reasoning is possible. This allows for some very useful operations, such as subclass, subproperty and property chain reasoning. This last property forms the main motivation to use URIs more extensively than is done in NIF. If we can represent linguistic properties as ontologies, we can define how output of different tools relate to each other. If, for instance, there are differences in **granularity** between output of different tools, it can be used to generalize over linguistic information (e.g., NNS $\subseteq$ NN $\subseteq$ NP). It is also possible to define equivalence or near equivalence. If, for instance, users want to use URIs that refer to ISOcat definitions that are still under construction, they can define their own tool-specific ontology of properties and values that can be linked to the ISOcat standards later on.

It should be noted that NAF strongly encourages the use of URIs, but does not enforce it. New layers of information can thus be integrated in NAF representations easily by using strings to represent properties and values. Ontologies that support the full reasoning and comparison advantages of RDF can be defined at a larger stage, when the need to compare or combine the information rises.

### 3.2.3 NAF: NLP Annotation Format

NAF is a layered annotation format, based on XML. If a process adds information which cannot be held by existing layers, it just adds a layer of annotation. Any previous layers remain intact and can still be used by other processes. Layers may be connected by means of references from one layer to items in another (lower level) layer.

A full description of the NAF format is given in the NAF manual. The remainder of this section relates NAF to ISO standards, and gives an overview of the NAF layers of annotation.

#### Annotation layers

In the NewsReader project, we use NAF to automatically annotate text documents. In this section, we show annotated examples from different NAF layers for a single sentence:

> *Followers of Muqtada al-Sadr clashed with British troops in the city of Amarah in battles late Monday that killed 15 Iraqis and wounded eight, said a coalition spokesman in the city, Wun Hornbyckle.*

NAF provides the following layers to represent the output of common NLP tasks:

**The header** contains metadata information about the input document, such as its public ID, the URI, creation time, etc. The header also records information about all the LP modules which were used to produce the NAF document.

```
<nafHeader>
 <fileDesc creationtime="2004-04-06"/>
 <public publicId="3938040573f3b401a3f9c66974fb4c4b"
        uri="http://usatoday.com/news/2004-04-06-iraqi-shiites_x.htm"/>
 <linguisticProcessors layer="text">
  <lp name="ixa-pipe-tok-en"
      timestamp="2013-06-26 14:15:18"
      version="1.0"/>
 </linguisticProcessors>
 <linguisticProcessors layer="terms">
  <lp name="ixa-pipe-pos-en"
      timestamp="2013-06-26 14:15:18"
      version="1.0"/>
 </linguisticProcessors>
 ...
</nafHeader>
```

Figure 9: Example: a fragment of the NAF header. The NAF header includes metadata about the source document, such as its creation time or URI. The header also includes information about all the LP modules which underwent the input text to produce the final NAF document.

**The raw layer** contains the input document verbatim. Because the input text may contain many characters which are invalid in XML, the raw layer is enclosed within a `CDATA` element.

**The text layer** contains the tokens of the document. Optionally, sentence, paragraph and page boundaries are indicated. This layer – the *text* element in NAF – is the result of sentence splitting and tokenization. Figure 10 shows how the example sentence is annotated in the text layer.

**The terms layer** contains words and multi-words. It also includes meta-information such as part-of-speech, references to other resources such as wordnet senses, whether or not it is a named entity, compound elements (in case of a compound), etc. Since (multi-)words consist of tokens, they refer to tokens in the *text layer*. Figure 11 shows two examples of (multi-)words in the terms layer.

**The chunks layer** contains chunks of words, such as noun phrases, prepositional phrases, etc. Since chunks consist of words, they refer to words in the *terms layer*. Each chunk has a *head*, which is also an item in the terms layer. Figure 12 shows two examples of chunks in the chunks layer.

**The dependency layer** contains dependency relations between words. Since words participate in dependency relations, they refer to words in the *terms layer*. Figure 12 shows examples of dependency relations between words in the example sentence.

```
<raw><![CDATA[Followers of Muqtada al-Sadr clashed .... Wun Hornbyckle.]]>
</raw>
<text>
  <wf id="w1" sent="1" offset="0" length="9">Followers</wf>
  <wf id="w2" sent="1" offset="10" length="2">of</wf>
  <wf id="w3" sent="1" offset="13" length="7">Muqtada</wf>
  <wf id="w4" sent="1" offset="21" length="7">al-Sadr</wf>
  <wf id="w5" sent="1" offset="29" length="7">clashed</wf>
  ...
 </wf>
</text>
```

Figure 10: Example: a fragment of the raw and text layers. The raw layer includes the input text verbatim. In the text layer, each token (enclosed in a *wf* element) has an identifier, an offset, a page number, a sentence number and a paragraph number.

**The entity layer** contains entity mentions. Entities mentions have an entity type (person, organization, etc) and are linked to and instance from an external resources such as Wikipedia or dbpedia. Figure 13 shows examples of entities found in the example sentence.

**The coreference layer** contains clusters of term spans which refer to the same entity. Figure 13 shows the coreference clusters of the example sentence.

**The semantic role layer** . Semantic Role Labeling (SRL) is a shallow semantic analysis which detects semantic arguments associated with predicates. Figure 14 shows the SRL layer for the example sentence.

**The time expression layer** identifies time expressions mentioned on the text. The time expressions are annotated using a format which mimics the TimeML standard [Pustejovsky *et al.*, 2010]. Figure 15 shows the temporal expressions extracted from the example sentence.

**The factuality layer** encodes the veracity or *factuality* of events as mentioned in the text. This information is useful for recognizing whether the events mentioned in the text actually happened (factual events), did not happen (contrafactual events), or there is some uncertainty about the event occurring or not. Figure 15 shows the factuality values for the example sentence.

The above layers form a chain of dependencies. The base layer of every NAF file is the text layer. All other layers are optional and are founded on the text layer, which makes it compliant with LAF. NAF files with few layers are useful for further processing, or for applications which need only superficial annotation. Although the chunks layer and the dependency layer can be added independently of each other, they are connected by a shared dependency on the terms layer, which ensures that they are both composed of the same elements.

```
<terms>
  <term id="t1" type="open" lemma="follower" pos="N" morphofeat="NNS">
    <span>
      <target id="w1"/>
    </span>
    <externalReferences>
      <externalRef resource="wn30g" reference="eng-30-10099375-n"
                   confidence="0.525004"/>
      <externalRef resource="wn30g" reference="eng-30-10100124-n"/>
                   confidence="0.474996"
    </externalReferences>
  </term>
  <!--of-->
  <term id="t2" type="close" lemma="of" pos="P" morphofeat="IN">
    <span><target id="w2"/></span>
  </term>
  <!--Muqtada-->
  <term id="t3" type="close" lemma="Muqtada" pos="R" morphofeat="NNP">
    <span><target id="w3"/></span>
  </term>
  <!--al-Sadr-->
  <term id="t4" type="close" lemma="al-Sadr" pos="R" morphofeat="NNP">
    <span><target id="w4"/></span>
  </term>
  <!--clashed-->
  <term id="t5" type="open" lemma="clash" pos="V" morphofeat="VBD">
    <span>
      <target id="w5"/>
    </span>
  <externalReferences>
    <externalRef resource="wn30g" reference="eng-30-02667698-v"
                 confidence="0.338607"/>
    <externalRef resource="wn30g" reference="eng-30-01561143-v"
                 confidence="0.331206"/>
    <externalRef resource="wn30g" reference="eng-30-00805228-v"
                 confidence="0.330187"/>
  </externalReferences>
</term>
 ...
</terms>
```

Figure 11: Example: a terms layer fragment. The *span* element contains references to the tokens in the *text layer* which comprise the (multi-)word. The (optional) *externalReferences* element contains references to wordnet senses and their corresponding confidence values.

```
<!-- Chunk layer -->
<chunks>
 <!-- Followers of Muqtada al-Sadr -->
 <chunk id="c1" head="t4" phrase="NP">
  <span>
   <target id="t1"/>
   <target id="t2"/>
   <target id="t3"/>
   <target id="t4"/>
  </span>
 </chunk>
 <...>
</chunks>

<!-- Dependency layer -->
 <deps>
   <!--nsubj(clashed-5, Followers-1)-->
    <dep from="t5" to="t1" rfunc="nsubj"/>
    <!--prep_with(clashed-5, troops-8)-->
    <dep from="t1" to="t5" rfunc="root"/>
    <!--amod(troops-8, British-7)-->
    <dep from="t8" to="t7" rfunc="amod"/>
 <...>
</deps>
```

Figure 12: Example: chunks and dependency layer fragment. The *span* element in the chunk layer contains references to items in the *terms layer* which comprise the chunk. Regarding the dependency layers, the first *dep* element indicates that *Followers* (the `from` attribute) is the subject (the `rfunc` attribute) of the *clash* verb (the `to` attribute). Both the `from` and the `to` attribute refer to the *terms layer*.

```
<!-- Entity layer -->
<entities>
 <entity id="e1" type="misc">
  <references>
    <!--British-->
    <span><target id="t7"/></span>
  </references>
  <externalRef resource="wikipedia"
             reference="http://en.wikipedia.org/wiki/United_Kingdom"
             confidence="0.541706"/>
 </entity>
 <entity id="e2" type="location">
   <references>
    <!--Amarah-->
    <span><target id="t13"/></span>
  </references>
  <externalRef resource="wikipedia"
             reference="http://en.wikipedia.org/wiki/Amarah"
             confidence="0.541706"/>
 </entity>
 ...
</entities>

<!-- Coreference layer -->
<coreferences>
  <coref id="co1">
    <!-- the city -->
    <span><target id="t31"/><target id="t32"/></span>
    <!-- Amarah -->
    <span><target id="t34"/><target id="t13"/></span>
  </coref>
</coreferences>
```

Figure 13: Example: entity and coreference layers. Named entity mentions are identified and related to external resources such as Wikipedia.

```
<!-- Entity layer -->
<srl>
 <predicate id="pr1">
  <!--clashed-->
  <externalReferences>
   <externalRef reference="clash.01" resource="PropBank"/>
   <externalRef reference="Hostile_encounter" resource="FrameNet"/>
   <externalRef reference="battle-36.4" resource="VerbNet"/>
   <externalRef reference="battle-36.4-1" resource="VerbNet"/>
  </externalReferences>
  <span><target id="t5"/></span>
  <role id="rl1" semRole="A0">
   <!--Followers of Muqtada al-Sadr-->
   <externalReferences>
    <externalRef reference="battle-36.4#Agent" resource="VerbNet"/>
   </externalReferences>
   <span><target head="yes" id="t1"/><target id="t2"/><target id="t3"/>
         <target id="t4"/></span>
  </role>
  <role id="rl2" semRole="A1">
   <!--with British troops-->
   <externalReferences>
    <externalRef reference="battle-36.4#Co-Agent" resource="VerbNet"/>
   </externalReferences>
   <span><target head="yes" id="t6"/><target id="t7"/><target id="t8"/>
   </span>
 </role>
 <role id="rl3" semRole="AM-LOC">
   <!--in the city of Amarah-->
   <span><target head="yes" id="t9"/><target id="t.mw10"/><target id="t12"/>
         <target id="t13"/></span>
  </role>
 </predicate>
 ...
</srl>
```

Figure 14: Example of semantic rol labeling layer. The example shows the information to one predicate, *clash*. The predicate is linked to external event models, like *clash.01* in PropBank or the *Hostile_encounter* frame of FrameNet. The example also shows how the roles of the predicate are filled in the text.

```
<!-- time layer -->
<timexs>
 <!-- 1970 -->
 <timex3 id="timex1" type="DATE"
    value="1970">
  <span><target id="c7"/></span>
 </timex3>
 <!-- 2003 -->
 <timex3 id="timex2" type="DATE"
    value="2003">
  <span><target id="c9"/></span>
 </timex3>
 <!-- between 1970 and 2003 -->
 <timex3 id="timex3" type="DURATION">
    value="P33Y" beginPoint="timex1"
    endPoint="timex2"
    temporalFunction="true"/>
</timexs>

<!-- factuality layer -->
<factualitylayer>
 <factvalue id="w19" prediction="CT+" confidence="0.9211625450434778"/>
 <factvalue id="w5" prediction="CT+" confidence="0.9404844085799826"/>
 <factvalue id="w26" prediction="CT+" confidence="0.8379289630516332"/>
 <factvalue id="w23" prediction="CT+" confidence="0.9717756091958029"/>
</factualitylayer>
```

Figure 15: Example to time and factuality layers.

## 3.3 SEM

SEM was designed to represent events in the broad sense of the word, derived from various sources (from the web, sensory data, historical documents, etc.). These data can be incomplete (*e.g.* missing values) and partial (*e.g.* missing entire facets), and they follow different design decisions. SEM has to be very flexible to cope with these issues. As SEM is meant to represent data from uncontrollable sources, the notions of temporary validity (during what temporal interval an event or a statement holds) and authority (according to whom an event or statement holds) become important. It is also important to allow all classes and properties in the model to be optional and duplicable, and to be flexible towards different ways of modeling time, place, type, and point of view,. In the rest of this section we first describe how these requirements are implemented in SEM and motivate the modeling decisions taken for the implementation. Then we discuss the classes and properties that make up SEM and how to model views and temporary validity with the concept of contexts.

### 3.3.1 Modeling Decisions

The primary consideration for designing SEM is that it should on the one hand be forgiving for the inherent messiness of the (Semantic) Web, while on the other hand still allowing a user to derive useful facts. On the Web, vocabulary owners can choose different options to classify the same domain, because different situations merit different distinctions. It can be hard to decide in advance which way will prove to be the most useful, especially because the Web allows reuse across domains, in applications that were not predicted beforehand. The more constraining a model is, the harder it is to reuse. To profit the most from what the (Semantic) Web has to offer it pays off to model with relatively weak semantics. To compensate for the lack of formal inference you can make with a weak model, you have to rely more on graph patterns (*e.g.* with SPARQL) to do reasoning.

The greatest implication of our decision to tailor an event model for data on the web is that we **can not** commit to a specific definition of an event. Events, according to SEM, encompass everything that happens, even fictional events. Whether there is a specific place or time or whether these are known is optional. It does not matter whether there are specific actors involved. Neither does it matter whether there is consensus about the characteristics of the event. For example, King Arthur's quest, the landing of UFO in Roswell or the elections of G.W. Bush, are valid events in SEM.

An important corollary of this loose definition of event and multitude of possible sources is that handling different viewpoints is crucial. In particular three aspects of viewpoints: (1) Roles, (2) time bounded validity of facts (*e.g.* time dependent type or role), (3) attribution of the authoritative source of a statement.

In order to query events at a relevant level of abstraction for any given application we need a good typing system. We would like to be able to reuse any vocabulary on the Web to pick our types from, regardless of how the concepts in these vocabularies are modeled.

The concrete implications of the web context for the RDF model of SEM are the following.

- We allow types to be both individuals or classes. This way we can borrow type identifiers from any vocabulary. It should not matter whether the type has been modeled as an individual or a class by the foreign vocabulary. (*cf.* OWL 2 punning)
- We use as few disjointness statements as possible, even where they would seem obvious. For example, SEM does not enforce places to be disjoint with actors. This allows reuse of vocabularies that do not make the distinction between a geographical region and its governing body.
- We only use rdfs:domain and rdfs:range to non-restricting classes (*i.e.* that can not be proved to be disjoint to any other SEM classes and hence do not restrict the domain or range of the property). This way we do not inherit any constraints from these classes through property semantics.
- We map to other event models with the SKOS vocabulary,[13] instead of using OWL constructs, to avoid overcommitment. In principle these mappings can be treated as documentation of the meaning of the SEM constructs and not as parts of their formal definition. SKOS mapping properties do not transfer OWL consequences. If stronger mappings are necessary it is possible to decide to momentarily replace the appropriate mappings relations with stronger versions, like owl:sameAs or rdfs:subClassOf.
- Every class and property is optional and can be duplicated, *i.e.* we do not model cardinality restrictions. Specifically we do not enforce the use of sem:types (not rdf:types, which *are* necessary).
- We do not declare properties functional, even if that seems appropriate. This avoids conflicts when aggregating data from different sources. For example, you might gather various birth dates for a single person. Even though the person was only born once–and thus inconsistency is appropriate–we do not want this to break our system. When reasoning over the web, debugging someone else's data is not always possible.
- We pay a special attention to graph patterns for efficient reasoning, to compensate for the limited formal constraints of SEM.

These implications are in line with the view of the Web outlined in chapter 1 of Allemang and Hendler [Allemang and Hendler, 2009].

### 3.3.2 SEM Specification

In this section we describe how these modeling decisions are implemented in SEM. First we discuss the core classes and the properties that make up SEM; then how to model points of view, possibly with temporary validity as named graphs that contain a number of (partial)

[13] `http://www.w3.org/2004/02/skos/`

event descriptions; and finally how to model time and space with OWL Time. We give a simple and more elaborate example of how an event from the historical domain can be modeled in SEM in Figure 19. These examples represent information from the following sentence,[14] which represents a typical sentence from the historical domain:

> *The Dutch launched the first* police action *in the Dutch East Indies in 1947; the Dutch presented themselves as liberators but were seen as occupiers by the Indonesian people.*

This example is interesting for a number of reasons: (1) it contains conflicting views on the role of the actor: were the Dutch liberators or occupiers? (2) it makes explicit according to which authority the roles hold (the Dutch / Indonesian people); (3) it presents a challenge for modeling the type of the place involved: the Dutch East Indies were at that time an independent Republic according to the Indonesians, but were a "controlled region" according to the Dutch. The next subsections describe the classes, properties and constraints of SEM.

**Classes** SEM's classes are divided in three groups: Core classes, Types, and Constraints. This is illustrated in Figure 16. There are four core classes: sem:Event (what happens), sem:Actor (who or what participated), sem:Place (where), sem:Time (when). Each core class has an associated sem:Type class, which contains resources that indicate the type of a core individual. Individuals and their types are usually borrowed from other vocabularies. For example, the sem:Place "Indonesia" (tgn:1000116) from Figure 19 and its sem:PlaceType "republic" (tgn:82171) are borrowed from the Getty Institute's Thesaurus of Geographical Names (TGN).[15]

The sem:Type classes exist to aggregate the various implementations of type systems in any vocabulary. Some vocabularies do not have properties that exactly correspond to the sem:type property, even though a type can be derived from the value of other properties. This can be done by using Alan Rector's Value Sets and Value Partition patterns.[16] These design patterns are illustrated in figure 17. Having explicit sem:Type classes provides a placeholder to define these patterns. If you want to make the class of all harbors using GeoNames' geo:featureCode property you could do this in the following two ways. You could define geo:featureCode to be a subproperty of sem:placeType. This makes geo:H.HBR a class, containing all geo:Features that are a harbor. If you do not want to turn the individual geo:H.HBR into a class you can follow the value sets pattern and define the set of harbors to be a subclass of sem:Place and an owl:Restriction on the geo:featureCode property with owl:hasValue geo:H.HBR. This approach keeps geo:H.HBR an individual.

[14]The original text is in Dutch. This sentence is extracted from the Netherlands Institute for Sound and Vision's catalogue description of the TV episode of Andere Tijden broadcasted on the 2004-10-26. The serie Andere Tijden consists of documentaries on historical topics.

[15]`http://www.getty.edu/research/conducting_research/vocabularies/tgn/`

[16]`http://www.w3.org/TR/swbp-specified-values/`

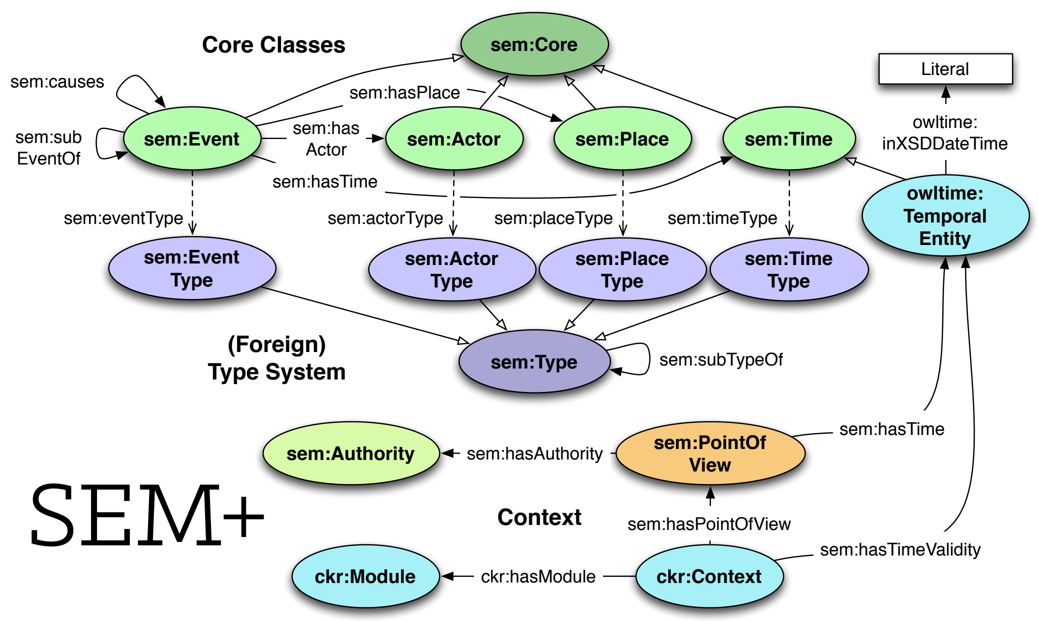


Figure 16: The classes of the Simple Event Model. Arrows with open arrow heads symbolize rdfs:subClassOf properties between the classes. Regular arrows visualize rdfs:domain and rdfs:range restrictions on properties between instances of the classes.

**Context** The version of SEM used in the NewsReader, SEM+, contains only one method to constrain the context of a statement: ckr:Context. This can be used to specify either the point of view of a collection of triples or the temporal validity. Roles, as defined in previous versions of SEM can be declared using subproperties of sem:eventProperty or sem:type. For example, Actors that have the role "occupier" in a certain event can be declared by introducing a subproperty of sem:hasActor, nwr:occupier, that relates the event to the actor. The temporal validity of this fact can be defined by putting the triple in a ckr:Module in a ckr:Context that limits the validity using sem:hasTimeValidity. The attribution of points of view, or the source of certain information can be done by attaching a sem:PointOfView to a ckr:Context, which can be assigned a sem:Authority representing the source of the information, or opinion. For example, Indonesia, in 1947, has either the type "republic" or "controlled region", depending on the source of information. This would be encoded with two distinct sem:placeType triples in two distinct ckr:Context named graphs. The fact that these two contexts are stated by two different parties is encoded by introducing two sem:Authority instances, respectively dbpedia:Indonesia and dbpedia:Netherlands that are connected to the context nodes with the sem:hasAuthority property. The sem:Authority class is meant as a hook for provenance and trust reasoning, even though SEM itself does not explicitly provide this.

**Properties** SEM's properties are divided in three kinds: sem:eventProperty, sem:type properties and a few miscellaneous properties like sem:hasAuthority and the temporal prop-

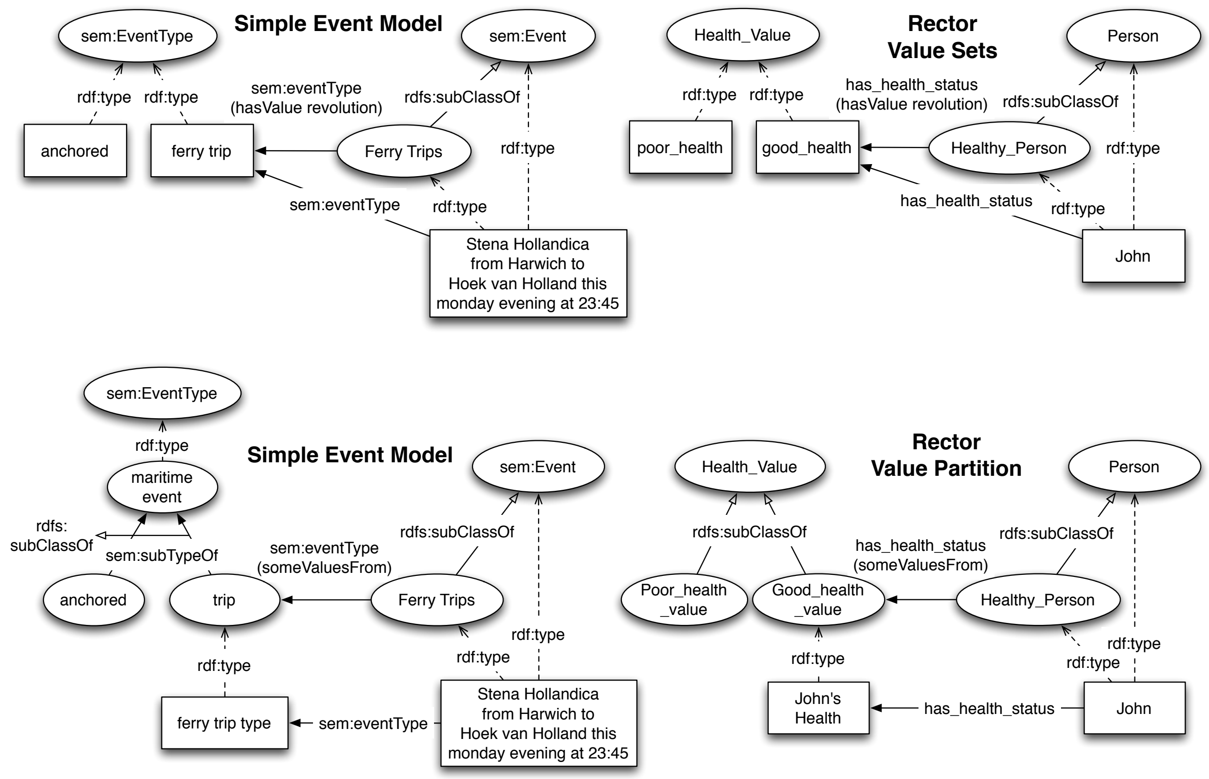


Figure 17: Alan Rector's value sets (top) and value partition (bottom) patterns applied to SEM (left) compared to the original examples from the W3C working group note (right).

Figure 18: The representations of the Role property constraint in SEM and of points of view modeled with Contexts.

erties sem:hasTime and sem:hasTimeValidity. The sem:eventProperty relates sem:Events to other individuals. The sem:eventProperty sem:causes can be used to describe causal chains of sem:Events. A sem:type relates individuals of the sem:Core class[17] to individuals of sem:Type. There are specific subproperties of sem:type for each of the core classes, for example sem:eventType, to facilitate querying. This reduces the strain on reasoners, because the property points directly to an individual of sem:EventType without doing any subsumption reasoning. sem:hasAuthority relates a sem:PointOfView to a sem:Authority and is used to represent opinions, which are linked to the named graph that contains the actual facts that comprise the opinion by means of the sem:hasPointOfView property. The classes ckr:Module and ckr:Context come from the Contextualized Knowledge Repository (CKR) framework [Bozzato and Serafini, 2013]

There are two aggregation relations amongst the sem:eventProperty and sem:type properties: sem:subEventOf and sem:subTypeOf. These can be used to indicate that respectively

[17]They also relate sem:Role to its sem:RoleType.

a sem:Event or sem:Type is related to another more generic sem:Event or sem:Type, without any further commitments. We decided not to model subtypes as subproperties of skos:broader/narrower, because we do not want to inherit the disjointness of skos:broader with skos:related. More specific relations between events and types are not part of SEM and should be taken from other ontologies, like GEM [Worboys and Hornsby, 2004].

There are two temporal properties: sem:hasTime, and sem:hasTimeValidity. Both refer to an OWL Time owltime:TemporalEntity, such as owltime:Interval or owltime:Instance, which are subclasses of sem:Time. The owltime:TemporalEntity is a abstract identifier for a certain moment or interval in time. A concrete timestamp can be (and should be) attached to the temporal entity by means of owltime:inXSDDateTime or owltime:inXSDDate, which refers to literal of the XML Schema datatype xsd:dateTime and xsd:date, which in turn follow the ISO 8601 date and time specification. An alternative way to label time instances is to use the sem:hasTimeStamp property to point to a rdf:XMLLiteral containing a TIMEX time element,[18] which supports both ISO 8601 time and relative time indications, such as "early yesterday morning".

A similar distinction between symbols and values exists when expressing places. There are symbolic places and coordinates. In SEM the individuals of the sem:Place class are symbolic places. Their location can be attached by using various constructs, like georss:point,[19] or wgs84:lat and wgs84:long.[20] Complex geometries like polygons can be encoded in GML[21] in an rdf:XMLLiteral pointed at by georss:where.

**Example** The historical example mentioned in the introduction of this section can be expressed in SEM+ as shown in Figure 19. The instance of the event ex:FirstPoliceAction in the example has one sem:Actor, which plays different roles according to two authorities. This is modeled with two separate triples that are both rdfs:subPropertyOf sem:hasActor. Each triple resides in a named graph that represents a Contextual Knowledge Repository ckr:Module. These modules are collected together into ckr:Contexts. In the case of this example, there two contexts, the context of the opinion of dbpedia:Netherlands and that of dbpedia:Indonesia. Another difference in opinion is which sem:placeType to assign to tgn:1000116 (Indonesia). The same construct can be used to limit the temporal validity of statements, which is outside the scope of this example. In that case, one could attach a sem:hasTimeValidity property to the node representing the ckr:Context pointing at a owltime:TemporalEntity denoting the period at which the statements in the modules of the context are considered valid.

[18]http://timex2.mitre.org/
[19]http://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/
[20]http://www.w3.org/2003/01/geo/
[21]http://www.opengeospatial.org/standards/gml

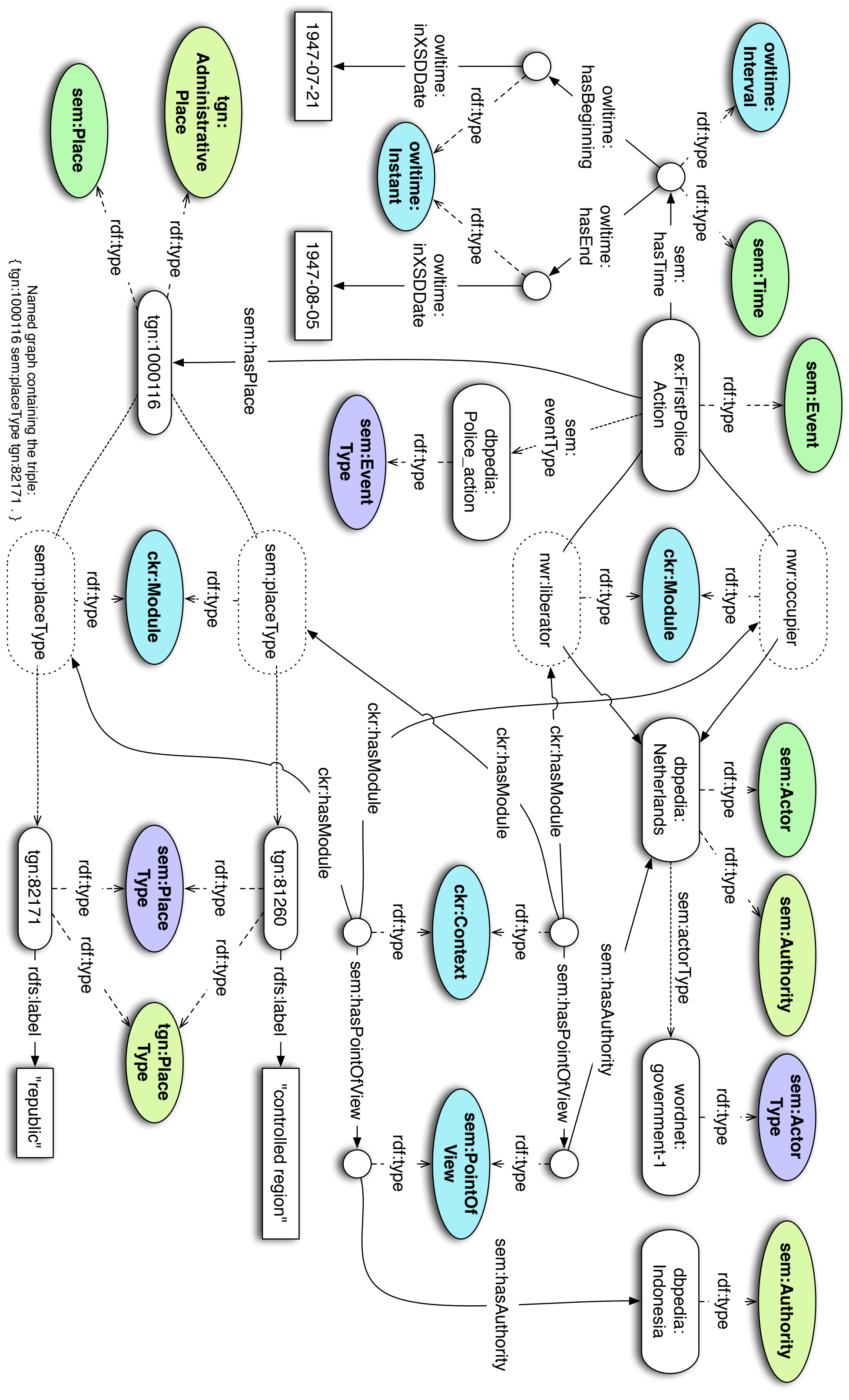


Figure 19: A representation of the historical example event in SEM+.

## 3.4 GAF

The previous sections described NAF and SEM+. NAF is used to represent linguistic information and SEM+ is the basic model we use to represent the interpretation of this information: *what* happened (events), *where* (locations) and *when* (time interpretations) and *who* was involved (participants). This sections describes the Grounded Annotation Framework [Fokkens *et al.*, 2013, GAF]. GAF allows us to relate the information in SEM+ to the linguistic analysis represented in NAF. The final output of our NLP analyses, which also forms the input for the KnowledgeStore, is expressed in GAF. This section provides a desciption of the framework.

### 3.4.1 Motivation

GAF is a new model for representing information that was designed specifically for the purpose of event representation. The framework was primarily designed to address two issues with current approaches with other event representations used in the NLP community. First, we want to be able to link information from text to information coming from non-textual sources such as databases. Second, we wanted to avoid using a 'trigger' or 'anchor' event mention for identifying a group of coreferring events. We will explain both of these aspects in detail below.

Events are not only described in textual documents, they are also represented in many other non-textual sources. These sources include videos, pictures, sensors or evidence from data registration such as mobile phone data, financial transactions and hospital registrations. Nevertheless, many approaches to textual event annotation consider events as text-internal-affairs, possibly across multiple documents but seldom across different modalities. It follows from the above that event representation is not exclusively a concern for the NLP community. It also plays a major role in several other branches of information science such as knowledge representation and the Semantic Web, which have created their own models for representing events.

GAF allows us to interconnect different ways of describing and registering events, including non-linguistic sources. GAF representations can be used to reason over the cumulated and linked sources of knowledge and information to interpret the often incomplete and fragmented information that is provided by each source. We make a clear distinction between *mentions* of events in text or any other form of registration and their formal representation as *instances* in a **semantic layer**.

Mentions in text are identified by our NLP pipeline and represented in NAF. The final semantic output is realized using Semantic Web technologies and standards. In this semantic layer, instances are denoted with Uniform Resource Identifiers (URIs). Attributes and relations are expressed according to the extended Simple Event Model [van Hage *et al.*, 2011, SEM+] that was described in the previous section and other established ontologies. Statements are grouped in named graphs based on provenance and (temporal) validity, enabling the representation of conflicting information. External knowledge can be related to instances from a wide variety of sources such as those found in the Linked Open Data

Cloud [Bizer *et al.*, 2009].

Optionally, instances in the semantic layer can be linked to one or more mentions in text and other sources. Because not all instances have to be linked to text, our representation offers a straightforward way to include information that can be inferred from text, such as implied participants or whether an event is part of a series that is not explicitly mentioned in any text. Due to the fact that each URI is unique, it is clear that mentions connected to the same URI have a coreferential relation. Other relations between instances (participants, causality, subevents, temporal relations, etc.) are represented explicitly in the semantic layer.

### 3.4.2 Background and related work

Annotation of events and of relations between them has a long tradition in NLP. The MUC conferences [Grishman and Sundheim, 1996] in the 90s did not explicitly annotate events and coreference relations, but the templates used for evaluating the information extraction tasks indirectly can be seen as annotation of events represented in newswires. Such events are not ordered in time or further related to each other. In response, [Setzer and Gaizauskas, 2000] describe an annotation framework to create coherent temporal orderings of events represented in documents using closure rules. They suggest that reasoning with text independent models, such as a calendar, helps annotating textual representations.

More recently, generic linguistically based corpora, such as Propbank [Palmer *et al.*, 2005] and the Framenet corpus [Baker *et al.*, 2003] have been built. The annotations aim at properly representing verb structures within a sentence context, focusing on verb arguments, semantic roles and other elements. In ACE 2004 [Linguistic Data Consortium, 2004b], event detection and linking is included as a pilot task for the first time, inspired by annotation schemes developed for named entities. They distinguish between event mentions and the trigger event, which is the mention that most clearly expresses its occurrence [Linguistic Data Consortium, 2004a]. Typically, agreement on the trigger event is low across annotators (around 55% [Moens *et al.*, 2011]). Timebank [Pustejovsky *et al.*, 2006] is a more recent corpus for representing events and time-expressions that includes temporal relations in addition to plain coreference relations.

All these approaches have in common that they consider the textual representation as a closed world within which events need to be represented. This means that mentions are linked to a trigger event or to each other but not to an independent semantic representation. More recently, researchers started to annotate events across multiple documents, such as the EventCorefBank [Bejan and Harabagiu, 2010]. Cross-document coreference is more challenging for establishing the trigger event, but it is in essence not different from annotating textual event coreference within a single document. Descriptions of events across documents may complement each other providing a more complete picture, but still textual descriptions tend to be incomplete and sparse with respect to time, place and participants. At the same time, the comparison of events becomes more complex. We thus expect even lower agreement in assigning trigger events across documents. [Nothman *et al.*, 2012] define the trigger as the first new article that mentions an event, which is easier than to find the clearest description and still report inter-annotator agreement of .48 and .73, respectively.

Recent approaches to automatically resolve event coreference (cf. [Chambers and Jurafsky, 2011], [Bejan and Harabagiu, 2010]) use some background data to establish coreference and other relations between events in text. Background information, including resources, and models learned from textual data do not represent mentions of events directly but are useful to fill gaps of knowledge in the textual descriptions. They do not alter the model for annotation as such.

We aim to take these recent efforts one step further and propose a grounded annotation framework (GAF). Our main goal is to integrate information from text analysis in a **formal context** shared with researchers across domains. Furthermore, GAF is **flexible** enough to contain contradictory information. This is both important to represent sources that contradict each other and to combine alternative annotations or output of different NLP tools. Because conflicting information may be present, **provenance** of information is provided in our framework, so that we may decide which source to trust more or use it as a feature to decide which interpretation to follow. Different models of event representation exist that can contribute valuable information. Therefore our model is compliant with prior approaches regardless of whether they are manual or automatic. Finally, GAF makes a clear distinction between *instances* and *instance mentions* avoiding the problem of determining a trigger event. Additionally, it facilitates the integration of information from extra-textual sources and information that can be inferred from texts, but is not explicitly mentioned. The next section will explain how we can achieve this with GAF.

### 3.4.3 An introduction to GAF

This section explains the basic idea behind GAF by using texts on earthquakes in Indonesia. GAF provides a general model for event representation (including textual and extra-textual mentions) as well as exact representation of linguistic annotation or output of NLP tools. Simply put, GAF is the combination of textual analyses and formal semantic representations in RDF.

We selected newspaper texts on the January 2009 West Papua earthquakes from the shared task dataset of the 2013 workshop on Events[22] to illustrate GAF. This choice was made because (1) using a dataset that comes from a shared task allows other researchers to compare their approaches to GAF (making GAF more accessible to other researchers) and (2) the topic "earthquake" illustrates the advantage of sharing URIs across domains.

[Gao and Hunter, 2011] propose a Linked Data model to capture major geological events such as earthquakes, volcano activity and tsunamis. They combine information from different seismological databases with the intention to provide more complete information to experts which may help to predict the occurrence of such events. The information can also be used in text interpretation. We can verify whether interpretations by NLP tools correspond to the data and relations defined by geologists or, through generalization, which

[22]`https://sites.google.com/site/cfpwsevents/`

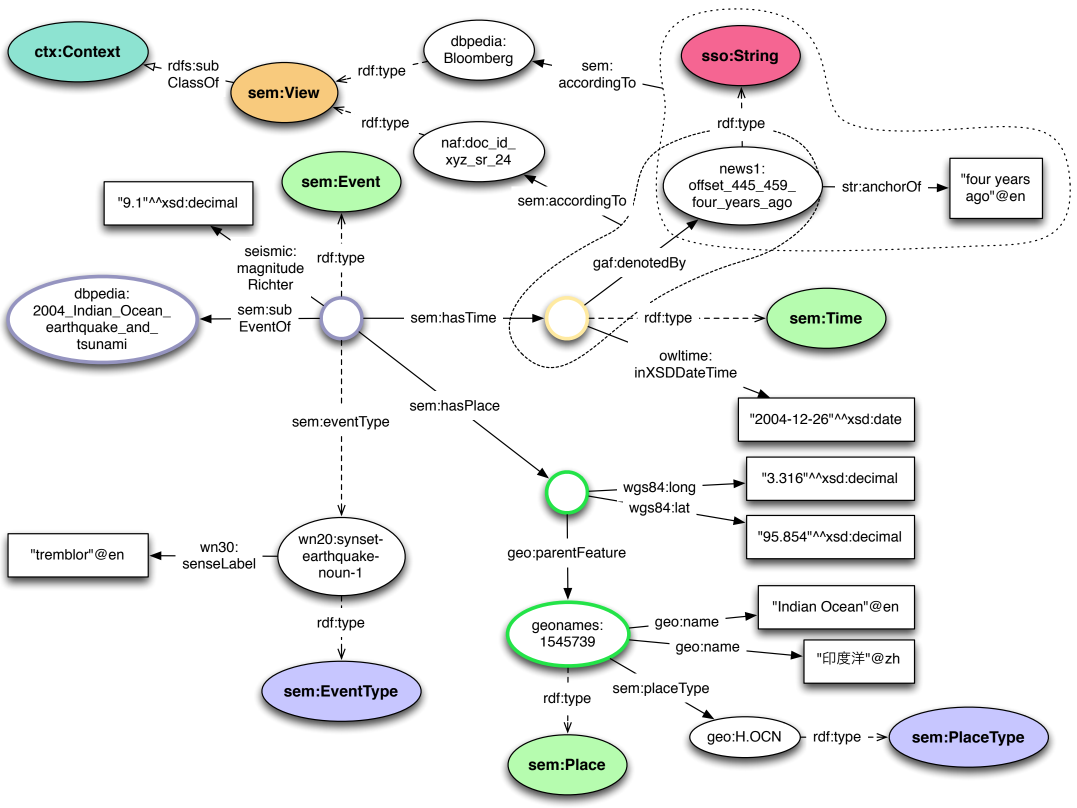


Figure 20: Partial SEM representation of December 26th 2004 Earthquake

interpretation is the most sensible given what we know about the events. General information on events obtained from automatic text processing, such as event templates [Chambers and Jurafsky, 2011] or typical event durations [Gusev *et al.*, 2010] can be integrated in SEM in a similar manner. Provenance indications can be used to indicate whether information is based on a model created by an expert or an automatically derived model obtained by a particular approach.

Figure 20 provides a fragment of a SEM representation for the earthquake and tsunami of December 26 2004.[23] The model is partially inspired by [Gao and Hunter, 2011]'s proposal. It combines information extracted from texts with information from DBpedia. The linking between the two can established either manually or automatically through an entity linking system.[24] The combined event of the earthquake and tsunami is represented by a

[23]A larger representation including several events and the sentences it represents can be found at `http://semanticweb.cs.vu.nl/GAF/Earthquakes.pdf`.

[24]Entity linking is the task of associating a mention to an instance in a knowledge base. Several approaches and tools for entity linking w.r.t. DBpedia and other data sets in the Linked Open Data cloud are available and achieve good performances, such as DBpedia Spotlight [Mendes *et al.*, 2011]; see [Rizzo and Troncy, 2011] for a comparison of tools.

DBpedia URI. The small circle on its right (linked through a sem:subEventOf, technically an RDF blank node) represents the earthquake itself.[25]

The unambiguous representation of the 2004 earthquake leads us to additional information about it, for instance that the earthquake is an event (sem:Event) and that the sem:EventType is an earthquake, in this case represented by a synset from WordNet, but also the exact date it occurred and the exact location (cf sem:hasTime, sem:hasPlace). The geo:parentFeature can provide more information on the event's location. Here, it indicates that the location is in the Indian Ocean represented by a unique identifier in GeoNames. It should be noted that URIs (even if they sometimes contain English words) are language-independent. Still, it is possible to ground the model to specific linguistic information by attaching it to URIs; the GeoNames location geonames:1545739 is associated to its English and Chinese names.

NAF annotations are converted to SEM relations using basic mapping rules.[26] For example, NAF semantic roles are translated to sem:hasActor relations, and the TLINK relations are translated to sem:hasTime. We use the relation gaf:denotedBy to include mentions of an instance in the text. The circle in the right upper corner represents an unambiguous pointer to an expression in the text, which is of type sso:String. The form of the expression is included through the attribute str:anchorOf. Text and sentence structure are modeled using definitions from the NLP Interchange Format (NIF) [Hellmann *et al.*, 2013][27], which we will also use in future versions of NAF. The origin of the annotation or the text can be represented with named graphs as explained in Section 3.3.[28] The output of the aggregation module mentioned in Section 3.1 outputs GAF and can be converted directly to RDF in named graph.

The relation sem:accordingTo indicates provenance. It can relate to the source of the text (dbpedia:Bloomberg is the source of *four years ago*), as well as to the source that links the mention to the instance. The connection between the instance mention *four years ago* and the time instance of when the earthquake took place in Figure 20 comes from a specific NAF annotation represented by naf:doc_id_xyz_sr_24.

### 3.4.4 GAF Earthquake Examples

This section takes a closer look at a few selected sentences from the text that illustrate different aspects of GAF. Figure 20 showed how a URI can provide a **formal context** including important background information on the event. Several texts in the corpus refer to the tsunami of 26 December 2004, *a 9.1 temblor in 2004 caused a tsunami* and *The catastrophe four years ago*, among others. Compared to time expressions such as *2004*

[25]A blank node was used in this illustration for reasons of space. In NewsReader, we will avoid using blank nodes as much as possible. Instances that do not have an identifier of their own will receive one that is generated by our system.

[26]This describes the basic implementation for translating NAF to SEM. More elaborate approaches may be used in future work.

[27]http://nlp2rdf.org/nif-1-0

[28]The use of named graphs in this way to denote context is compatible with the method used by [Bozzato *et al.*, 2012].

and *four years ago*, time indications extracted from external sources like DBpedia are not only more precise, but also permit us to correctly establish the fact that these expressions refer to the same event and thus indicate the same time. The articles were published in January 2009: a direct normalization of time indications would have placed the catastrophe in 2005. The **flexibility** to combine these seemingly conflicting time indications and delay normalization can be used to correctly interpret that *four years ago* early January 2009 refers to an event taking place at the end of December 2004.

A fragment relating to one of the earthquakes of January 2009: *The quake struck off the coast [...] 75 kilometers (50 miles) west of [....] Manokwari* provides a similar example. The expressions *75 kilometers* and *50 miles* are clearly meant to express the same distance, but not identical. The location is most likely neither exactly 75 km nor 50 miles. SEM can represent an underspecified location that is included in the correct region. The exact location of the earthquake can be found in external resources. We can include both distances as expressions of the location and decide whether they denote the general location or include the normalized locations as alternatives to those from external resources.

Different sources may report different details. Details may only be known later, or sources may report from a different perspective. As **provenance** information can be incorporated into the semantic layer, we can represent different perspectives, and choose which one to use when reasoning over the information. For example, the following phrases indicate the magnitude of the earthquakes that struck Manokwari on January 4, 2009:

*the 7.7 magnitude quake* (source: Xinhuanet)
*two quakes, measuring 7.6 and 7.4* (source: Bloomberg)
*One 7.3-magnitude tremor* (source: Jakartapost)

The first two magnitude indicators (7.7, 7.6) are likely to pertain to the same earthquake, just as the second two (7.4, 7.3) are. Trust indicators can be found through the provenance trace of each mention. Trust indicators can include the date on which it was published, properties of the creation process, the author, or publisher [Ceolin *et al.*, 2010]. Furthermore, because the URIs are shared across domains, we can link the information from the text to information from seismological databases, which may contain the exact measurement for the quake.

Similarly, external information obtained through shared links can help us establish coreference. Consider the sentences in Figure 21. There are several ways to establish that the same event is meant in all three sentences by using shared URIs and reasoning. All sentences give us approximate time indications, location of the affected area and casualties. Reasoning over these sentences combined with external knowledge allows us to infer facts such as that *undersea [...] off [...] Aceh* will be in the Indian Ocean, or that the affected countries listed in the first sentence are *countries around the Indian Ocean*, which constitutes the *Indian Ocean Community.* The number of casualties in combination of the approximate time indication or approximate location suffices to identify the earthquake and tsunami in Indonesia on December 26, 2004. The DBpedia representation contains

```
There have been hundreds of earthquakes in Indonesia since a 9.1 temblor in 2004 caused a
tsunami that swept across the Indian Ocean, devastating coastal communities and leaving more
than 220,000 people dead in Indonesia, Sri Lanka, India, Thailand and other countries.
(Bloomberg, 2009-01-07 01:55 EST)

The catastrophe four years ago devastated Indian Ocean community and killed more than 230,000
people, over 170,000 of them in Aceh at northern tip of Sumatra Island of Indonesia.
(Xinhuanet, 2009-01-05 13:25:46 GMT)

In December 2004, a massive undersea quake off the western Indonesian province of Aceh
triggered a giant tsunami that left at least 230,000 people dead and missing in a dozen
countries facing the Indian Ocean.  (Aljazeera, 2009-01-05 08:49 GMT)
```

Figure 21: Sample sentences mentioning the December 2004 Indonesian earthquake from sample texts

additional information such as the magnitude, exact location of the quake and a list of affected countries, which can be used for additional verification. This example illustrates how a **formal context** using URIs that are shared across disciplines of information science can help to determine exact referents with limited or imprecise information.

### 3.4.5 Summarizing GAF

GAF is an event annotation framework in which textual mentions of events are grounded in a semantic model that facilitates linking these events to mentions in external (possibly non-textual) resources and thereby reasoning. We illustrated how GAF combines NAF and SEM through a use case on earthquakes. We explained that we aim for a representation that can combine textual and extra-linguistic information, provides a clear distinction between instances and instance mentions, is flexible enough to include conflicting information and clearly marks the provenance of information.

GAF ticks all these boxes. All instances are represented by URIs in a semantic layer following standard RDF representations that are shared across research disciplines. They are thus represented completely independent of the source and clearly distinguished from mentions in text or mentions in other sources. The formal semantic model (SEM) provides the flexibility to include conflicting information as well as indications of the provenance of this information. This allows us to use inferencing and reasoning over the cumulated and aggregated information, possibly exploiting the provenance of the type of information source. This flexibility also makes our representation compatible with all approaches dealing with event representation and detections mentioned in Section 3.4.2 as well as our own representations in NAF. It can include automatically learned templates as well as specific relations between events and time expressed in text. Moreover, it may simultaneously contain output of different NLP tools.

The proposed semantic layer may be simple, its flexibility in importing external knowledge may increase complexity in usage as it can model events in every thinkable domain. To resolve this issue, it is important to scope the domain by importing the appropriate vocabularies, but no more. When keeping this in mind, reasoning with SEM is shown to be rich but still versatile [Van Hage *et al.*, 2012].

While GAF provides us with the desired granularity and flexibility for the event annotation tasks we envision, a thorough evaluation still needs to be carried out. This includes an evaluation of the annotations created with GAF compared to other annotation formats, as well as testing it within a greater application. A comparative study of top-down and bottom-up annotation will also be carried out. We are currently using the CAT tool to create manual annotations and convert those to SEM and are planning to adapt CROMER (CRoss-document Main Event and entity Recognition, described in Deliverable 3.1 [Tonelli and Sprugnoli, 2013]) so that we can annotate the semantic layer directly for this comparative study.

# 4 Linguistic Processing

The goal of the NewsReader linguistic pipeline is to process daily news-streams in four languages (English, Dutch, Spanish and Italian) and extract the relevant events mentioned on texts, so that it can gather knowledge on what happened to whom, when and where, removing duplication, complementing information, registering inconsistencies while keeping track of original sources.

The NewsReader approach to event mining involves the deepest processing of text currently available. Furthermore, NewsReader requires such technology to be applied on realistic volumes of text within hard time constraints. In this section will describe the main approach taken by the project to fulfil these requirements.

We first describe the virtual machine approach used in the project. Then, we describe the distributed pipeline for NLP processing implemented in the first year of the project. Section 4.3 describes the NLP processing actually performed in this first year and reports statistics regarding execution times, memory consumed and overall throughput of the deployed NLP modules. Finally, we describe the design of the final NLP processing architecture which is able to continually consume document streams in a short time.

## 4.1 Virtual machines for inter-operability

Linguistic processors are complex software packages which often require a large set of dependencies to be met in order to effectively perform their tasks. Deploying LPs often requires pre-installing a large set of common software modules on the same machine, which must be accessible to the LP. The capacity of replicate the results is very important within the project. One LP module applied to a particular input text has to produce the same output regardless the software framework (machine, operating system, etc.) where it is installed. Therefore, special care has to be taken on guaranteeing that the same version of the LP modules, along with the exact same dependencies, are deployed among machines.

The aforementioned reasons have lead us to use virtual machine (VM) technologies for deploying the LP modules. Virtualization is a widespread practice that increases the server utilization and addresses the variety of dependencies and installation requirements. Besides, virtualization is a 'de-facto' standard on cloud computing solutions, which offer the possibility of installing many copies of the virtual machines on commodity servers.

In the NewsReader project will create one VM per language and pipeline. Inside the VM the involved partners will install the required LP modules (along with the dependencies) so that a full LP processing in one language can be processed on a single VM. Project deliverable D4.2 "Event Detection, version 1"[29] describe the LP modules considered to be installed for the first year of the project. Appendix A contains the detailed instructions for deploying NLP modules inside the shared NewsReader VM, as well as for copying and deploying the VM into several hosts.

[29] `http://www.newsreader-project.eu/files/2012/12/NewsReader-316404-D4.2.pdf`

## 4.2 A distributed pipeline for NLP processing

Scalable NLP processing requires parallel processing of textual data. The parallelization can be effectively performed at several levels, from deploying copies of the same LP processor among servers to the reimplementation of the core algorithms of each module using multi-threading, parallel computing. This last type of fine-grained parallelization is clearly out of the scope of NewsReader project, as it is unreasonable to expect it to reimplement all the modules needed to perform such a complex task as mining events. We rather aim to processing huge amount of textual data by defining and implementing an architecture for NLP processing which allows the parallel processing of documents.

To this end, we project will use VM as building blocks of the linguistic pipeline. As described in section 4.1, we will define one VM per language and install all LP modules in there. This approach allows the project to scale horizontally (or scale out) as a solution to the problem of dealing with massive quantities of data. In the first year of the project we will scale out our solution for NLP processing by deploying all NLP modules into VMs and making as many copies of the VMs as necessary to process an initial document batch of documents on time.

Inside each VM the modules are managed using the Storm framework for streaming computing. As explained in section 2.1, Storm processing is based on *topologies*: a graph of computation which processes messages forever. Storm topologies are thus top level abstractions which describe the processing that each message undergoes. Topology nodes fall into two categories: the so called *spout* and *bolt* nodes. *Spout* nodes are the entry points of topology and the source of the initial messages to be processed. *Bolt* nodes are the actual processing units, which receive messages, process them, and pass the processed messages to the next stage in the topology. The data model of Storm is the *tuple*, i.e., each node in the topology consumes[30] and produces tuples. The tuple is an abstraction of the data model, and is general enough to allow any data to be passed around the topology.

Although the Storm pipeline is focused on a streaming computing scenario, in the first year of the project we will follow a batch approach to analyze the documents. This batch approach have two main purposes:

- It will allow the project to have an initial set of events, spanning through a specified time span, into which new informations and event will be integrated.
- The initial processing will serve as a testbed of the NewsReader technology.

Once the VMs are installed and copied among machines, a set of documents will be manually split into several batches and sent to different VMs. The output of the processing stage will be then batch uploaded to the KnowledgeStore.

Table 5 shows the modules installed into the English pipeline VM. Each LP module is wrapped as a *bolt* node inside the Storm topology. When a new tuple arrives, the *bolt* node calls an external command sending the tuple content to the STDIN standard stream. The output of the LP module is received from the STDOUT stream and passed to the next node

[30]Unlike *spout* nodes, which are the initial nodes and therefore do not consume tuples.

| **Module** | **Description** | **Required Input** | **Output layer(s)** |
|---|---|---|---|
| TOK | Tokenizer, Sentence splitter | Raw text | Tokens, Sentences |
| POS | POS tagger | Tokens | Lemmas, POS tags |
| Parse | Constituency parser | Raw text | Parse trees |
| timex | Time expressions | Lemma, POS tags | Time expressions |
| NERC | Named Entity Recognition | Lemmas, POS tags | Named Entities (NE) |
| WSD | Word Sense Disambiguation | Lemmas, POS tags | Synsets |
| NED | Named Entity Disambiguation | NE | Disambiguated NE (DNE) |
| Coref | Coreference resolution | Parse trees, Synsets, DNE | Coreference relations |
| Dep | Dependency parser | Lemmas, POS tags | Dependency relations |
| SRL | Semantic Role Labeling | Dependency relations | Semantic Roles |
| Fact | Factuality | Event relations, Time expressions, Dependency relations | Factuality indications |
| eCoref | Event coreference | Event relations, Factuality indications, Coreference relations, Synsets, Semantic Roles | Event coreferences |

Table 5: LP modules installed on the English pipeline VMs. The required input and produced output is also shown.

in the topology. The inter-operability among modules is guaranteed by using a common format for NLP annotations, i.e., the NAF (see section 3.2). Each module thus receives a NAF document with the (partially annotated) document and adds new annotations onto it. The tuples in our Storm topology comprise two elements, a document identifier and the document itself, encoded as a string with the XML serialization of the NAF document.

If one module fails for whathever reasons to produce a valid NAF document, the input document is moved to a specific directory and a log entry is created. The processing of this particular document is stopped at this point, and the system starts processing the next document in the input directory.

Inside the VM there is an initial *spout* which scans for a particular directory. When a new document arrives, the *spout* passes the document to the first node in the pipeline, which in turn will pass its output to the next stage, etc. This setting is similar to a standard pipeline architecture but has a main advantage: when a module finishes the processing, it passes the annotated document to the next step, and start processing the next document. Therefore, in this setting there are as many documents processed in parallel as stages in the pipeline.

Note that in this approach Storm is used within a single VM and that this setting is

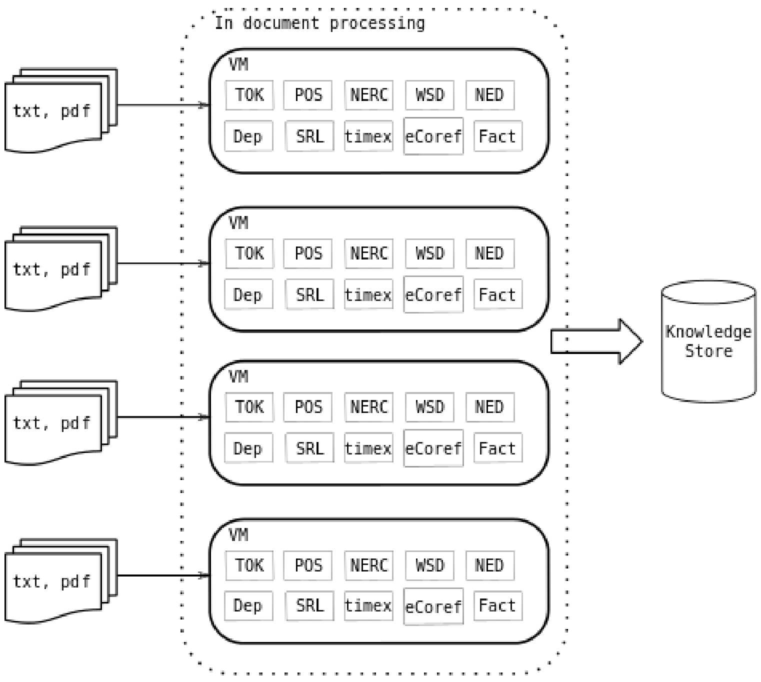


Figure 22: Linguistic processing, first year.

not ideal nor the architecture type Storm framework is meant to be used for. However, using Storm as the controlling backbone of the LP modules installed within each VM has the following advantages:

- Inside each VM, the Storm topology is able to run many LP modules in parallel.
- Having implemented this *batch* approach using Storm, it is straightforward to adopt a fully *streaming* architecture as described in section 4.4, where LP modules reside on several distributed VMs.
- The initial experiments performed using the batch approach will give important insights as to the need of using specialized VMs containing very time consuming LP processors.

## 4.3 Running the pipeline

We have linguistically analyzed 64.540 documents using the setting described in the previous section. The document were selected by first performing a query on the LexisNexis News database by selecting specific keywords related to car companies ("Alfa Romeo", "Aston Martin", "BMW", etc). This initial query retrieved around 6 million News documents, which are further filtered by considering only documents spanning between 2001 and 2013 years and containing more than 2 car company names. The goal is to obtain

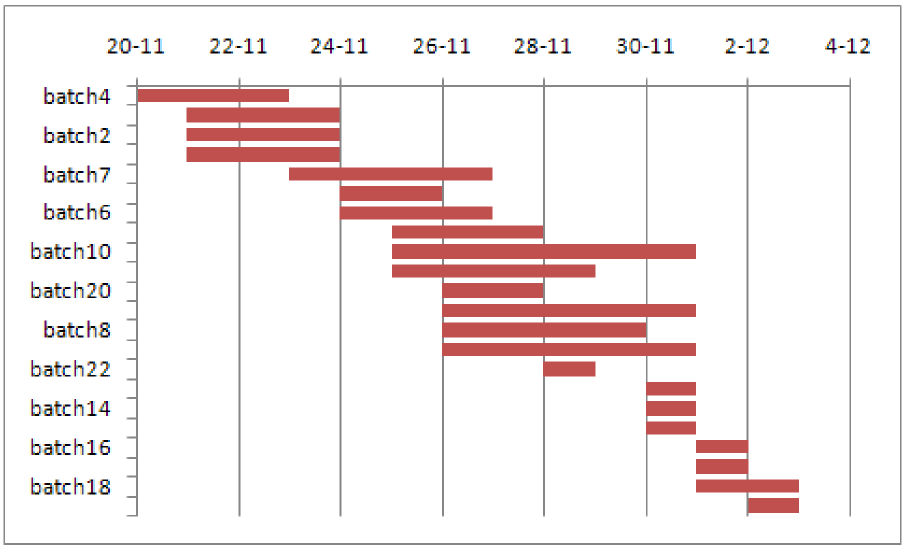


Figure 23: Gantt diagram of the processed batches.

interesting documents describing events which involve two or more car companies. As said before, the resulting dataset comprises 64.540 documents.

The dataset is split into 22 batches containing circa 3000 documents, and each batch is sent to the LP pipeline for linguistic processing. In total, we used 8 VMs distributed among the project partners.

Figure 23 shows the gantt diagram for whole LP processing. The elapsed times are computed by analyzing the time spans of each output documents[31]. Because of the manual involvement when uploading batches into the VMs, as well as due to problems when installing the VM copies into the host computers of each partner, the figure shows that we did not managed to run all VMs in parallel. However, we can see that if we group the batches into 8 parallel lines the whole processing would be done circa 5 days

Table 6 shows the total time spent by each module, the total percentage of the time, and the number of elements extracted. The elapsed times shown in the Table do not consider that many of those documents are actually processed in parallel. The Table shows that if the documents were processed sequentially, they would require 37.8 days to process. Having 8 VM running on parallel, this number were actually processed in around 5 days, as stated above.

Table 6 also suggests an unbalance regarding the time spent of each module of the pipeline. The Semantic Role Labeling (SRL) module takes more than 70% of the processing time, and is by far the module needing more time to complete its task. This results suggests that SRL is a good candidate for parallelization; if we were able to execute several instances of the SRL module in parallel, the overall performance of the linguistic processing would boost considerably.

[31]Each LP processor writes a time span into the NAF header which allows us to analyze the time elapsed by each component.

| Module | Total time | Percent | # elements |
|---|---:|---:|---:|
| TOK | 12,152s | 0.37% | 35,187,862 |
| POS | 45,352s | 1.38% | 34,527,492 |
| eCoref | 73,098s | 2.23% | 3,747,382 |
| NED | 75,086s | 2.29% | 1,960,604 |
| WSD | 78,080s | 2.38% | 595,874 |
| Opinion | 82,127s | 2.51% | 114 |
| Fact | 91,960s | 2.81% | 4,327,233 |
| NERC | 112,643s | 3.44% | 2,475,062 |
| DEP | 121,092s | 3.70% | 31,943,943 |
| multiword | 261,586s | 7.99% | 636,292 |
| SRL | 2,322,472s | 70.90% | 5,246,967 |
| Total | 3,263,496 | 100 % | 120,648,825 |

Table 6: Total time (in seconds) and percentage taken by each module in the pipeline. The last column indicates the number of elements extractec from the text.

The Storm framework allows several instances of each topology node, thus allowing the actual parallel processing. Following the so called *parallelism hint*, it is possible to specify how many instances of each topology node will be actually running. Therefore, we performed a separate experiment with the aim of measuring the expected performance gain when executing time consuming modules in parallel. We defined a small pipeline comprising only four modules: the tokenizer, the part-of-speech tagger, the Named Entity Recognizer and the Word Sense Disambiguation modules. The experiment setting is the usual one, where the four modules are executed using a Storm pipeline which mimics a pipeline architecture (each module running sequentially one after the other). The experiments were ran on a single PC computer with a Intel Core i5-3570 3.4GHz processor with 4 cores and 4GB RAM, running Linux. We tested the NLP pipeline with 8 documents, each one comprising about 1500 words and 60 sentences. We performed experiments for 4 documents ($6,773$ words and 271 sentences) and the complete set of 8 documents ($12,077$ words and 462 sentences).

In the pipeline the WSD module is the most time consuming: for the 70 seconds needed to process the documents, almost 60 seconds are spent in it. We thus experimented three alternatives (dubbed $Storm_3$, $Storm_4$ and $Storm_5$), with 3 instances (respectively, 4 and 5 instances) of the WSD module running in parallel.

Table 7 shows the time elapsed by processing the documents. The first four rows correspond to the processing of 4 documents and the last four rows to the processing of 8 documents. As the Table shows, the baseline Storm topology runs at a performance of about 95 words per second. The results also show that running multiple instances of WSD does increase the overall performance significatively. The major gain is obtained with four copies of WSD, with an increase of 60% in the overall performance. This result is expected, giving the fact that the machine where the experiments were ran has 4 CPU cores.

All in all, this initial experiments have shown that there is a big room for improvement

| | Total time | words per sec | sent. per sec. | gain |
|---|---|---|---|---|
| | | Four documents | | |
| Storm | 1m11.372s | 94.9 w/s | 3.8 sent/s | 0.0% |
| $\text{Storm}_3$ | 0m38.274s | 177.0 w/s | 7.1 sent/s | 46.4% |
| $\text{Storm}_4$ | 0m33.540s | 201.9 w/s | 8.1 sent/s | 53.0% |
| $\text{Storm}_5$ | 0m34.472s | 196.5 w/s | 7.9 sent/s | 51.7% |
| | | Eight documents | | |
| Storm | 1m57.277s | 103.0 w/s | 3.9 sent/s | 0.0% |
| $\text{Storm}_3$ | 0m58.492s | 206.5w/s | 7.9s/s | 54.5% |
| $\text{Storm}_4$ | 0m50.415s | 239.6w/s | 9.2s/s | 60.8% |
| $\text{Storm}_5$ | 0m54.544s | 221.4w/s | 8.5s/s | 57.6% |

Table 7: Performance of a small NLP pipeline in different settings. Storm is the basic pipeline topology. $\text{Storm}_3$ is the Storm pipeline with 3 instances of the WSD module ($\text{Storm}_4$ has 4 instances and $\text{Storm}_5$ 5 instances, respectively).

regarding NLP processing performance. A careful identification of the most time and resource consuming NLP modules allows creating parallel topologies which yield better performance. With the use of large clusters with many nodes, we expect a significant boost in the performance of overall NLP processing.

## 4.4 Towards a full streaming NLP processing

So far we have described an architecture which allows parallel execution of several NLP modules and a batch processing of many documents in a short time. However, the ultimate goal of the project is to allow a streaming processing of documents, i.e., dealing with the continuous processing of streams of documents reaching at any time. This section will describe the design of such an architecture, using the pipeline described on the previous section as a starting point.

The basic streaming architecture is shown in Figure 24. The documents enter to the processing stage via a single entry-point and are put into a document queue. The pipeline is implemented following a Storm topology, as usual. In this setting, each node of the topology may reside on a different physical machine; the Storm controller (called *Nimbus*) is the responsible to send the tuples among the different machines, and guarantees that each message undergoes all the nodes in the topology. In our approach, the *Nimbus* supervisor node retrieves the documents, converts them into tuples, and send them to the processing topology.

Unlike the approach described in the section above, where a document is completely analyzed inside a single VM, this setting allows distributed processing of documents into several VMs. That is, different stages of the pipeline a document undergoes are actually executed on different machines. This approach allows multiple copies of LP modules running inside a specialized VMs or even dynamically increasing the computing power for the LP modules which require more computational resources.

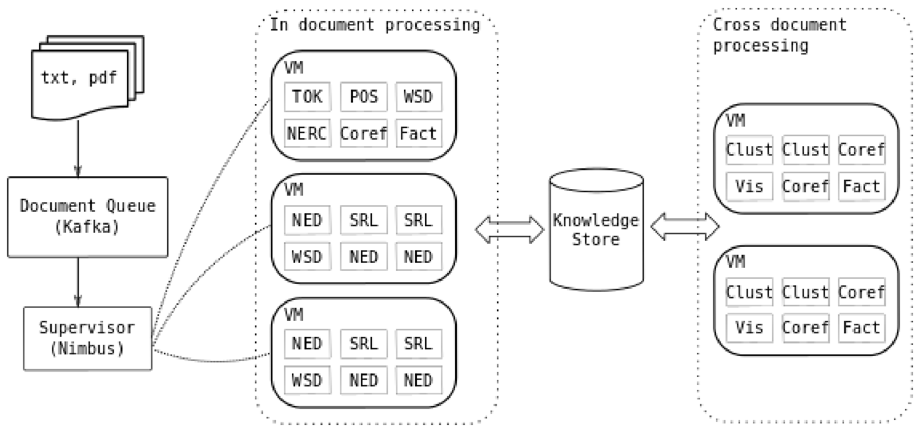


Figure 24: Linguistic processing, streaming computing approach.

As can be seen in the Figure, the KnowledgeStore plays a fundamental role in this setting, as it serves as central repository where all intermediate results are stored. The KnowledgeStore has to allow parallel access for storing and retrieving documents, as otherwise it would be a major bottleneck in the whole architecture. Besides, partial results have to provide pointers to the original NAF documents, so that the document can be rebuilt from the partial annotations it contains.

In future experiments we want to try the following settings:

- Non linear topologies. The architectures described so far follow the pipeline approach, but in principle we could also consider executing non linear topologies, where two modules are processing the same document at the same time. Non linear topologies require considering the following aspects:
  - We need to clearly identify the pre- and post-requisites of each module, thus deducting the indications as to which modules must precede which and which modules can be ran in parallel on the same document.
  - We need a special *bolt* which receives input from many NLP modules *bolts* (each one conveying different annotations on the same document) and *merges* all this information producing a single, unified document.
- Granularity. NLP modules work at different type of granularity. For instance, a POS tagger works at a sentence level, the WSD module works at paragraph level, whereas a coreference module works at a document level. We want to experiment splitting the input document into pieces of the required granularity, so that the NLP modules can quickly analyze those pieces, thus increasing the overall processing speed.

# 5 KnowledgeStore

The initial design of the KnowledgeStore is documented in Deliverable D6.1 ("KnowledgeStore Design")[32]. In this section, we briefly recall the main characteristics of the current version of the KnowledgeStore Design, pointing the interested reader to Deliverable D6.2 ("KnowledgeStore version 1") for a more comprehensive exposition.

## 5.1 Introduction

The rate of growth of digital data and information is nowadays continuously increasing. While the recent advances in Semantic Web Technologies (e.g., the Linked Data[33] initiative), have favoured the release of large amounts of data and information in structured machine-processable form (e.g., RDF dataset repositories), a huge amount of content is still available and distributed through websites, company internal Content Management System (CMS) and repositories, in an unstructured form, for instance as textual documents, web pages, and multimedia material (e.g., photos, diagrams, videos). Indeed, as observed in [Gantz and Reinsel, 2011], unstructured data accounts for more than 90% of the digital universe.

Although bearing a clear dichotomy for what concern their form, the content of structured and unstructured resources is far from being separated: they both speak about *entities* of the world (e.g., persons, organizations, locations, events), their properties, and relations among them. Indeed, coinciding, contradictory, and complementary facts about these entities could be available in structured form, unstructured form, or both. Therefore, partially focusing on the content distributed in only one of these two forms may not be appropriate, as complete knowledge is a requirement for many applications, especially in situations where users have to make (potentially critical) decisions. Moreover, some applications inherently require considering both types of content: an example is *question answering* [Ferrucci *et al.*, 2010], where often a user query can only be answered by combining information in structured and unstructured sources.

Despite the last decades achievements in natural language and multimedia processing, now supporting large scale extraction of knowledge about entities of the world from unstructured digital material, frameworks enabling the seamless integration and linking of knowledge coming both from structured and unstructured content are still lacking.

We present the implementation of the first version of the KnowledgeStore, a framework that contributes to bridge the unstructured and structured worlds, enabling to jointly store, manage, retrieve, and semantically query, both typologies of contents. Figure 25 shows schematically how the KnowledgeStore manages these contents in its three *representation layers*. On the one hand (and similarly to a file system) the *resource layer* stores unstructured content in the form of resources (e.g., news articles, multimedia files), each having a textual or binary representation and some descriptive metadata. Information

[32]Further additional details are provided in [Rospocher *et al.*, 2013]
[33]`http://linkeddata.org`

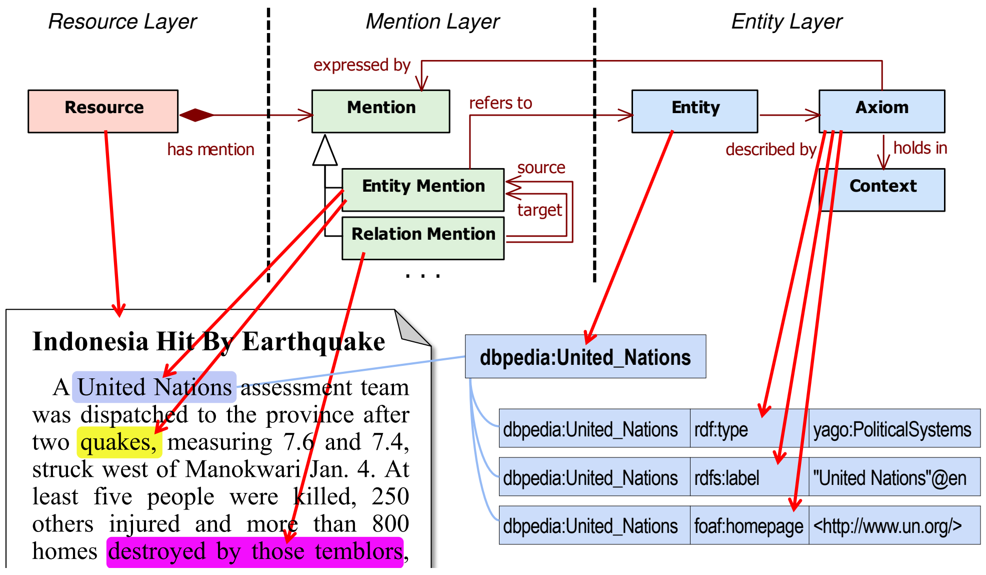

Resource Layer
Mention Layer
Entity Layer
expressed by
Resource
Mention
refers to
Entity
Axiom
has mention
described by
holds in
source
Entity Mention
Context
target
Relation Mention
. . .
Indonesia Hit By Earthquake
A United Nations assessment team was dispatched to the province after two quakes, measuring 7.6 and 7.4, struck west of Manokwari Jan. 4. At least five people were killed, 250 others injured and more than 800 homes destroyed by those temblors,
dbpedia:United_Nations
dbpedia:United_Nations rdf:type yago:PoliticalSystems
dbpedia:United_Nations rdfs:label "United Nations"@en
dbpedia:United_Nations foaf:homepage <http://www.un.org/>

Figure 25: KnowledgeStore Content.

stored in this level is typically noisy, ambiguous, and redundant, with the same piece of information potentially represented in different ways in multiple resources. On the other hand, the *entity layer* is the home of structured content, that, based on Knowledge Representation and Semantic Web best practices, consists of *axioms* (a set of ⟨subject, predicate, object⟩ triples), which describe the *entities* of the world (e.g., persons, locations, events), and for which additional metadata is kept to track their provenance and to denote the formal *contexts* where they hold (e.g., in terms of time validity and point of view). Differently from the resource layer, the entity layer aims at providing a formal and concise representation of the world, abstracting from the many ways it can be encoded in natural language or in multimedia, and thus allowing the use of automated reasoning to derive new statements from asserted ones [De Bruijn and Heymans, 2007]. Between the aforementioned two layers is the *mention layer*. It indexes *mentions*, i.e., snippets of resources (e.g., some characters in a text document, some pixels in an image) that denote something of interest, such as a an entity or an axiom of the entity layer. Mentions are automatically extracted by the modules of the NLP pipeline, that enrich them with additional attributes about how they denote their referent (e.g., with which name, qualifiers, "sentiment"). Far from being simple pointers, mentions present both unstructured and structured facets (respectively snippets and attributes) not available in the resource and entity layers alone, and are thus a valuable source of information on their own.

Thanks to the explicit representation and alignment of information at different levels, from unstructured to structured knowledge, the KnowledgeStore enables the development of enhanced applications, and favours the design and empirical investigation of several information processing tasks otherwise difficult to experiment with. Effective *decision making support* could be provided by exploiting the possibility to semantically query the content of the KnowledgeStore with requests that combine structured and unstructured content (a.k.a.

mixed queries), like e.g., "retrieve all the documents mentioning that person Barack Obama participated to a sport event"[34]. The KnowledgeStore favours the implementation and evaluation of tools which exploit available structured knowledge to improve the performance of *coreference resolution* tasks (i.e., identifying that two mentions refer to the same entity of the world), as shown in [Bryl *et al.*, 2010], especially in cross-document / cross-resource settings. Finally, the joint storage of extracted knowledge, the resources it derives from, and extraction metadata provides an ideal scenario for developing, training, and evaluating ontology population techniques. In particular, the KnowledgeStore data model favours the exploration of a number of computational strategies for *knowledge fusion*, i.e., the merging of possibly contradicting information extracted from different sources, and *knowledge crystallization*, i.e., the process through which information from a stream of multimedia documents is automatically extracted, compared, and finally integrated into background knowledge.

Given the KnowledgeStore ambition to cope with a huge quantity of data and resources (potentially in the range of billions of documents), as required by today / next future applications, the development of the KnowledgeStore vision is necessarily driven by *scalability* aspects: performances in storing, accessing, and querying the KnowledgeStore have to gracefully scale with respect to the size of managed content. For this reason the implementation of the KnowledgeStore is based on technologies compliant with the deployment in distributed hardware settings, like clusters and cloud computing.

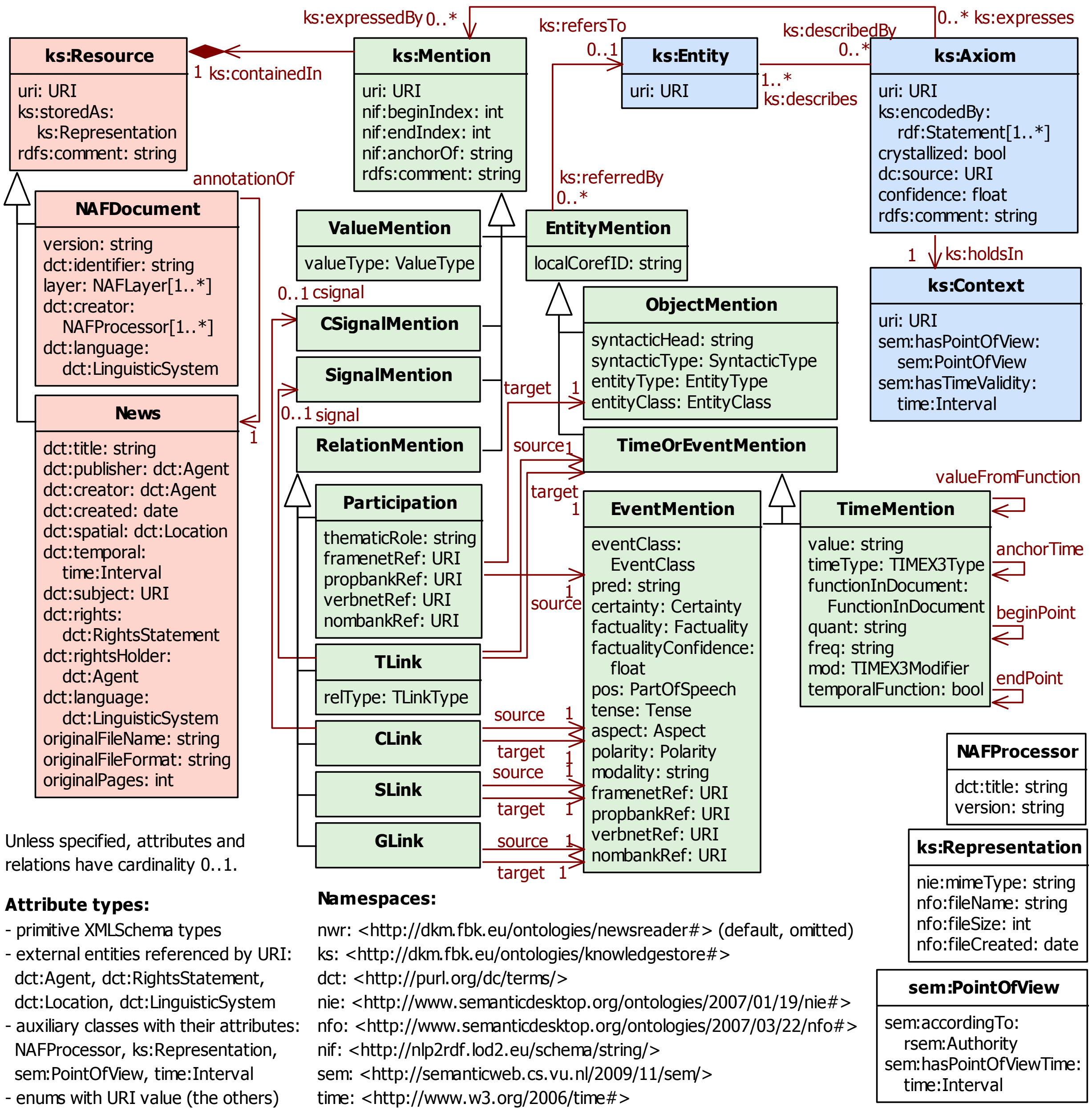
The remaining of this section of the deliverable is organized as follows. In Section 5.2 we present in details the NewsReaderKnowledgeStore data model. In Section 5.3 we illustrate how the other NewsReader modules can interact with the KnowledgeStore, detailing in particular the type of (semantic) requests they can submit to it, while in Section 5.4 we describe the KnowledgeStore architecture, presenting the modules composing the framework. Section 5.5 concludes presenting some initial experiments to asses the scalability performance of the KnowledgeStore and the technologies it relies on.

## 5.2 The Knowledge Store Data Model

The data model defines what information can be stored in the KnowledgeStore, in accordance with the modelling and specification work of WP3, WP4, WP5 and, in particular, the annotation format (NAF). It serves both as a basis for the design of the KnowledgeStore architecture and interfaces, and as a shared model that permits WP4 and WP5 linguistic processors and the decision support tool suite of WP7 to cooperate.

The NewsReader KnowledgeStore data model is depicted in the UML class diagram of Figure 26. The model is organized in the three *resource*, *mention* and *entity* layers. Resources and mentions are described using a closed but configurable set of types, attributes and relations, while entities are described with an open set of axioms annotated with

[34]Fulfilling this request involves: (i) to reason in the structured part about which events "Barack Obama" participated to that are of type "sport event", and identify the corresponding participation statements; (ii) to exploit the links to the mentions those statements have been extracted from; and (iii) to exploit the linking between those mentions and the resources containing them.



Unless specified, attributes and relations have cardinality 0..1.

**Attribute types:**
- primitive XMLSchema types
- external entities referenced by URI: dct:Agent, dct:RightsStatement, dct:Location, dct:LinguisticSystem
- auxiliary classes with their attributes: NAFProcessor, ks:Representation, sem:PointOfView, time:Interval
- enums with URI value (the others)

**Namespaces:**
nwr: <http://dkm.fbk.eu/ontologies/newsreader#> (default, omitted)
ks: <http://dkm.fbk.eu/ontologies/knowledgestore#>
dct: <http://purl.org/dc/terms/>
nie: <http://www.semanticdesktop.org/ontologies/2007/01/19/nie#>
nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#>
nif: <http://nlp2rdf.lod2.eu/schema/string/>
sem: <http://semanticweb.cs.vu.nl/2009/11/sem/>
time: <http://www.w3.org/2006/time#>

Figure 26: NewsReader data model.

metadata attributes (e.g., for provenance) and holding inside specific contexts. Resources, mentions, entities, axioms, and contexts are identified by externally-assigned URIs. The model is assimilable to an OWL 2 ontology [Motik *et al.*, 2009] that can be encoded in RDF with Named Graph [Carroll *et al.*, 2005] used for metadata. This allows to encode both the model definition and its instance data using RDF [Beckett, 2004] (e.g., for interfacing with the KnowledgeStore or for Linked Data publishing). Named Graphs [Carroll *et al.*, 2005] can be used to encode metadata and contexts, and other Semantic Web technologies—such as the SPARQL query language[35]—can be leveraged to deal with represented data. Next, we provide a brief overview of the content representable in each layer.

**Resource layer** The resources of interest in NewsReader are News and NAFDocuments. News are described using metadata from the Dublin Core vocabulary (dct:* attributes), augmented with NewsReader-specific attributes to encode additional information (e.g., the originalFileName of the imported news). For NAF annotations only the NAF version, publicId (dct:identifier), available layers, employed NAF processors (dct:creator) and link to annotated news are represented, as they are useful in accessing and selecting data in the KnowledgeStore, while annotation data is stored in the NAF resource file itself.

**Mention layer** The representation of mentions derives from the NAF specification. The offset of a mention in a news, as well as its extent, are encoded using attributes from the NLP Interchange Format (NIF) vocabulary[36], thus enabling interoperability with tools consuming NIF data. Four main types of mentions are distinguished:

- *Entity mentions* denote entities in the domain of discourse (linked via refersTo), and are further characterized based on the type of entity. Object mentions refer to persons, locations, organizations, products, financial objects and mixed entities (discriminated via entityType), like e.g. "Barak Obama", "NASDAQ Index", "a family", "500 cars". Object mentions are further characterized by a syntactic head, a syntactic type (e.g., name, nominal or pronoun) and a linguistic entity class (e.g., specific referential). Time mentions are described using the subset of TIMEX3 properties selected for NAF (e.g., TIMEX3 type, normalized value and function within the document). Event mentions are characterized using a number of NAF attributes: the linguistic class of the event (e.g., speech-cognitive); the lemma of the token conveying the event (pred); the part-of-speech (pos), e.g., noun or verb; the certainty and factuality of the event, including a factuality confidence value; the tense, aspect, polarity and modality of the verbal form used. In addition, references to external resources further specifying the type of event are stored (framenetRef, verbnetRef, propbankRef, nombankRef). For all the entity mentions, an optional localCorefID attribute can be used to group mentions coreferring within a document.

- *Relation mentions* express relations between two entities, whose mentions are identified by source and target links. Different kinds of relation mentions are stored. Causal links (CLink) express a causal relation between two events, while temporal links (TLink) denote a certain temporal relation (relType, e.g., before, include, simultaneous) among two events or time expressions. Subordinate links (SLink) express certain structural relations among events, while GLinks denote a grammatical relation among events (as in "the share drop came on the same day", with "drop" and "came" being events). Participation mentions denote the participation of an entity to

[35]`http://www.w3.org/wiki/SPARQL`
[36]`http://nlp2rdf.org/nif-1-0`

an event in a certain thematic role (semRole), possibly further specified by references to external resources (framenetRef and similar).

- *Signal and CSignal mentions* identify pieces of text supporting the existence of, respectively, a temporal or causal relation, to which they are linked by relations signal.

- *Value mentions* are numerical expressions used for quantities (cardinal numbers in general), percentages and monetary expressions; the type of value is stored.

**Entity layer** Different kinds of entities are stored, including persons, organizations, geo-political entities or locations, events, points and intervals in time extracted from text. The type of entity is conveyed by an axiom consisting of an rdf:type triple. The context in which an axiom holds in is described and identified in terms of temporal validity (sem:hasTimeValidity) and point of view (sem:hasPointOfView, e.g., "Financial Times"); the Simple Event Model (SEM) [van Hage *et al.*, 2011] and the OWL Time[37] vocabularies are used to that purpose. Axiom metadata consists of a confidence value (confidence), a provenance indication (dct:source) and a crystallized flag (crystallized). Confidence is represented on a $0.0 - 1.0$ scale and quantifies how reliable an extracted axiom is. Provenance is stored for a background knowledge axiom and denotes the external source it has been imported from (e.g., DBPedia).[38] The crystallized flag is set for axioms belonging to background knowledge or assimilated to it after repeated extraction of the conveyed information, according to some crystallization algorithm. This algorithm (to be defined as part of WP6 T6.2) will exploit information such as how many mentions an axiom has been extracted from (attribute ks:extractedFrom) and in which time frame, as well as which resources it was extracted from (e.g., which kind of news) and how reliably; it will also consider pre-existing background knowledge, in form of TBox constraints and other ABox assertions an axiom has to be consistent with.

## 5.3 The Knowledge Store Interfaces

The KnowledgeStore presents a number of interfaces, offered as part of the KnowledgeStore API, through which external clients may access and manipulate stored data. In the definition of these interfaces for the first version of the KnowledgeStore, some design choices were made:

- coarse-grained operations, that operate on whole sets of objects at a time (e.g., the simultaneous update of all the mentions of a certain resource), are provided in order to enable the efficient retrieval and modification of large amounts of data;

[37]`http://www.w3.org/TR/owl-time/`

[38]The adoption of a provenance model to track sources, authority, and tool processing activities, is under definition at project level at the time of writing this deliverable. The data model here presented will thus be revised according to the resulting provenance model.

- operations work according to a request-response message exchange pattern: the client issues the request to the KnowledgeStore and waits for its reply;
- API calls modifying a set of objects (e.g., resoures, mentions, entities) are applied to each object in the set in a transactional way that satisfies ACID properties[39], i.e., each object in the set is either successfully modified or not modified at all, with the change permanently persisted and no concurrent client seing intermediate states during the modification of the object;[40]
- data validation on input data is performed for each API request, in order to check the preconditions which are instrumental to the successful completion of the operation (e.g., presence and validity of object identifier and mandatory attributes);
- client authentication (based on username/password credentials) is enforced to restrict access only to authorized clients.

All the technical partners of the consortium were involved in the definition of the operations to be implemented by the KnowledgeStore. These operations are offered to the user clients as part of the KnowledgeStore API through two endpoints: the *CRUD endpoint*, that provides the basic operations to access and manipulate the objects stored in all the layers the KnowledgeStore, and the *SPARQL endpoint*, that enables flexible access to the semantic content stored in the entity layer.

**CRUD Endpoint** The CRUD endpoint provides the basic operations to access and manipulate (CRUD: create, retrieve, update, and delete) any object stored in any of the layers of the KnowledgeStore. CRUD operations are defined in terms of sets of objects, in order to enable bulk operations as well as operations on single objects. In details, the following operations are provided:

- `create (object descriptions) : assigned URIs and/or creation errors`
  Stores new objects based on their supplied descriptions.
- `retrieve (condition, output attributes) : object descriptions`
  Returns all the objects matching a supplied *condition*.
- `update (condition, object description, merge criteria) : update errors`
  Updates all the objects matching a supplied condition, setting one or more of their attributes (or entity axioms) to a particular value; if the attributes were already set, *merge criteria* can be optionally used to combine old values with new ones.

[39]Transactions are units of work—either a single operation or a sequence of operations—to which certain properties are associated, such as the *ACID* properties of relational databases: *atomicity*, *consistency*, *isolation* and *durability*.

[40]Transactional guarantees on the whole API call are significantly more complex and expensive to provide, as the call may affect millions of records for which a transactional log should be kept. Such guarantees are not feasible with the technologies currently adopted for the KnowledgeStore. Should this situation change, they will be considered for a future release of the KnowledgeStore.

- `delete (condition) : deletion errors`
  Deletes all the objects matching a supplied condition.
- `merge (object descriptions, merge criteria) : merge errors`
  Updates a set of objects given their identifiers, setting one or more attributes (or entity axioms) to specific values and possibly applying merge criteria to combine old and new values.
- `count (condition) : # matching objects`
  Returns the number of objects matching a supplied condition.
- `match (conditions and output attribute URIs at resource, mention, entity and axiom level) : matching <resource, mention, entity, axioms> tuples`
  Returns a set of ⟨resource, mention, entity, axioms⟩ 4-tuples whose mention occurs in the resource, refers to the entity and supports the extraction of the axioms, and such that the attributes on all the four components satisfy certain conditions; for each tuple, a specified set of output attributes for the four components is returned.

**SPARQL Endpoint** As the entity layer of the KnowledgeStore is grounded in Knowledge Representation and Semantic Web best practices, a further dedicated access mechanism is provided to access in a flexible way (i.e., by means of queries in SPARQL, a standard query language able to retrieve and manipulate data stored in Semantic Web repositories) entities and axioms[41] stored in the KnowledgeStore. Recall that, in our approach, each axiom corresponds to a set of ⟨subject, predicate, object⟩ triples within a named graph [Carroll *et al.*, 2005] that encodes the context where the axiom holds; the named graph is uniquely identified by the context URI associated to the axiom. Therefore, clients interacting with the KnowledgeStore through SPARQL have to be aware of this contextual dimension when submitting the query to the KnowledgeStore, as well as when interpreting the results obtained.

Here below is the description of the KnowledgeStore `sparqlQuery()` operation:[42]

`sparqlQuery(query, dataset) : query solutions or triples`
  Evaluates the supplied SPARQL `query` on indexed statements or a subset of them identified by the `dataset` parameter.

Both endpoints are made available to the external KnowledgeStore user clients via an HTTP ReST API, which favours the integration of the KnowledgeStore in complex frameworks with clients developed using different languages, technologies or execution platforms. For the specific and relevant case of clients written in Java, a Java client library is also offered; it builds on the HTTP ReST API and enables easy integration in Java-based tools.

[41]To be more precise, as described in 5.4, only crystallized and background knowledge axioms are accessible via SPARQL.

[42]The definition of the `sparqlQuery()` operation is based on the SPARQL protocol standard [Feigenbaum *et al.*, 2013]; indeed, the SPARQL protocol is used to implement this API operation.

## 5.4 The Knowledge Store Architecture

From a functional point of view, the KnowledgeStore is a storage server: the other NewsReader modules are KnowledgeStore clients that utilize the services it exposes to store and retrieve all the shared contents they produce and need. The KnowledgeStore is a passive component, without any active role concerning the orchestration of other NewsReader modules. The lower part of figure 5.4 shows the KnowledgeStore client-server architecture. A specifically developed *Frontend Server* implements the KnowledgeStore functionalities and its two CRUD and SPARQL endpoints on top of a number of distributed and scalable components: the Hadoop HDFS filesystem for storing resource representations; HBase with the OMID transaction manager and ZooKeeper synchronization service for storing resource, mention and entity data and the Virtuoso triple store for indexing entity axioms and supporting SPARQL querying. Next, we detail the main components.

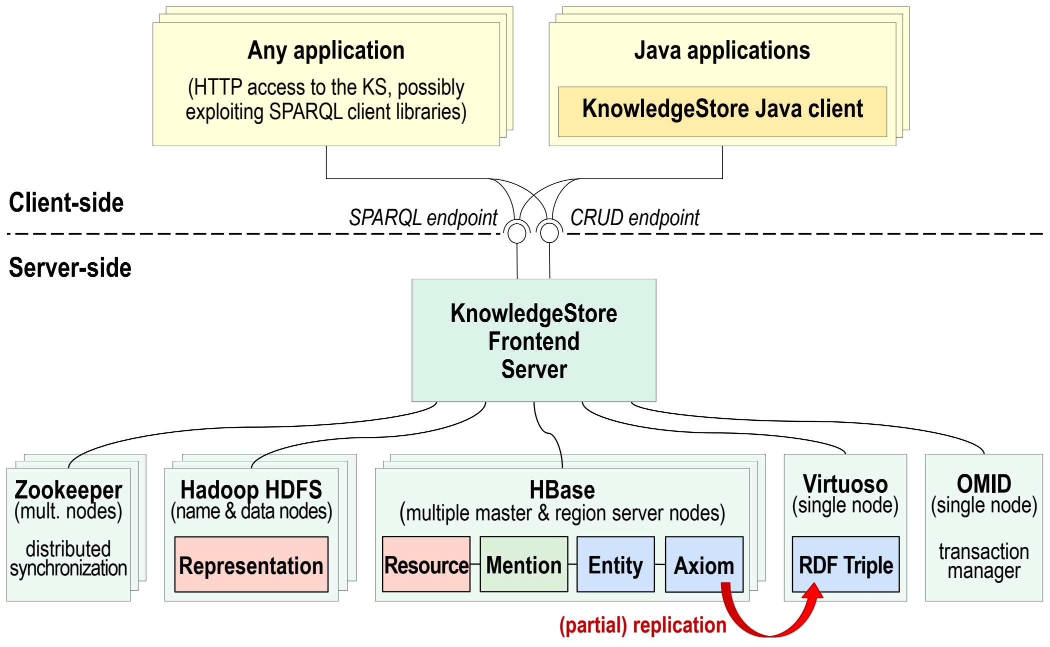


Figure 27: KnowledgeStore architecture.

**Hadoop HDFS component** Hadoop[43] is a frameworks developed by Apache to realize scalable, distributed file systems supporting processing of huge amounts of data using the MapReduce framework. The KnowledgeStore utilizes the Hadoop distributed file system (DFS) to store resource representations, that is the physical files such as news documents or the NAF annotations produced by the linguistic processors.

[43]`http://hadoop.apache.org`

**HBase component** HBase[44] is a frameworks developed by Apache realizing a scalable, distributed *column-oriented* database. Distributed computation on multiple nodes, replication and fault tolerance with respect to single node failure are its key features.

HBase is used as a database to store, with dedicated tables, resource metadata, mentions, contexts and axioms attributes. The last type of object—axioms—deserves a special description, as they are also stored in the Virtuoso component, but with different purposes. Full information about axioms is stored in the HBase component, where each axiom is stored with an URI, uniquely identifying it, a collection of triple ⟨subject, predicate, object⟩ defining it, the URI of the context where the axiom holds, and axiom-level metadata attributes (e.g., provenance and confidence values). This solution allows for a compact representation of statement metadata and fast lookup, but it's not enough for supporting SPARQL queries [Seaborne and Harris, 2013], which are instead provided by the Virtuoso component. The latter holds only a subset of the axioms (e.g., only the crystallized ones, so to reduce the load on Virtuoso), and just stores their triples and context components, with the first encoded as a set of RDF triples and the latter as the named graph containing those triples. Axiom metadata is not stored in Virtuoso, as (i) it is often irrelevant to user queries, and (ii) its representation would require the use of expensive and impractical techniques such as RDF reification.[45]

**Virtuoso component** Axioms are indexed in a triple store so to enable efficient, inference-aware SPARQL-based query answering. Indexing affects only those axioms that satisfy certain configurable criteria; this allows, for instance, to exclude from inference non-crystallized axioms or axioms whose extraction confidence level is below a given threshold.

As remarked in Section 5.3, each axiom is stored as a set of ⟨subject, predicate, object⟩ triples within a named graph [Carroll *et al.*, 2005] that encodes the context where the axioms holds. Contexts are defined by additional triples (placed in a *global* graph) that encode the contextual attributes such as point of view (sem:hasPointOfView) and time validity (sem:hasTimeValidity) identifying the context. As anticipated, additional axiom metadata are not indexed. By resorting to a triple store, indexed axioms can be easily queried using SPARQL both as a language and access protocol (via so-called SPARQL endpoint offered by triple stores).

Different triple store implementations offer different performance, scalability and fault-tolerance characteristics, as well as licenses (e.g., open source vs commercial). The first implementation of the KnowledgeStore will be based on the Open Source Edition of the Virtuoso triple store[46]; more details about the selection of Virtuoso and some experiments to assess its performance and scalability characteristics for use in the KnowledgeStore are reported in Section 5.5. However, the use of the OpenRDF Sesame Java API[47] will permit

[44]`http://hbase.apache.org`

[45]Note that in the KnowledgeStore implementation it is always possible to go back and forth from one representation to the other, since axioms are uniquely identified by the set of triples defining them, which are stored both in HBase and in the Triple Store.

[46]`http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/`

[47]`http://www.openrdf.org/`

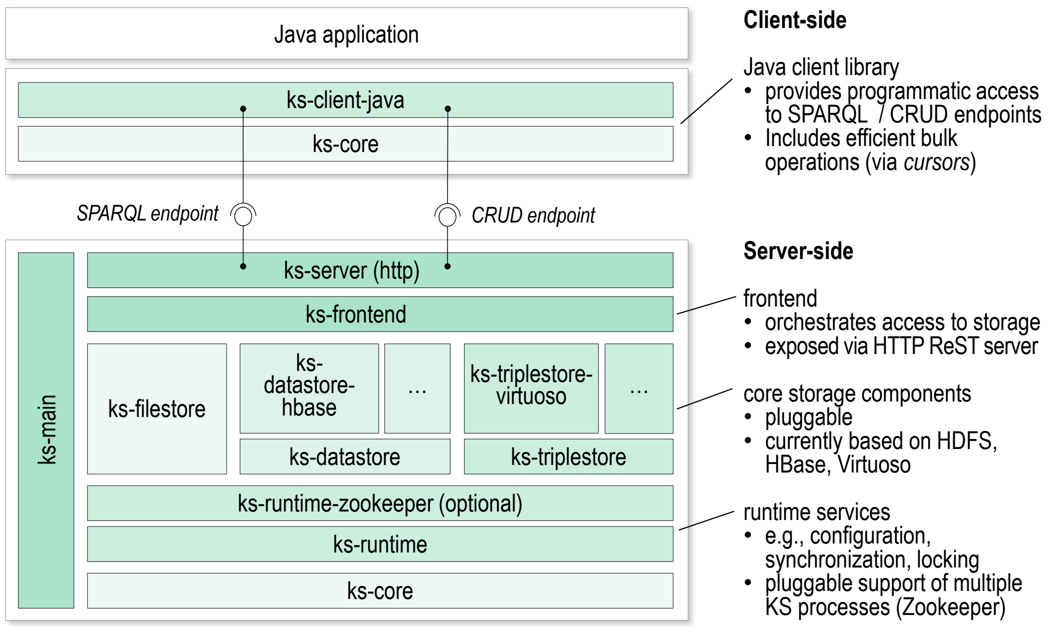

Java application
ks-client-java
ks-core
SPARQL endpoint
CRUD endpoint
ks-server (http)
ks-frontend
ks-main
ks-filestore
ks-datastore-hbase
...
ks-triplestore-virtuoso
...
ks-datastore
ks-triplestore
ks-runtime-zookeeper (optional)
ks-runtime
ks-core
Client-side
Java client library
• provides programmatic access to SPARQL / CRUD endpoints
• Includes efficient bulk operations (via cursors)
Server-side
frontend
• orchestrates access to storage
• exposed via HTTP ReST server
core storage components
• pluggable
• currently based on HDFS, HBase, Virtuoso
runtime services
• e.g., configuration, synchronization, locking
• pluggable support of multiple KS processes (Zookeeper)

Figure 28: Detailed internal modules organization of the KnowledgeStore

the KnowledgeStore to largely abstract from the adopted triple store implementation[48], thus allowing to change it within (and beyond) the scope of NewsReader.

**KS Frontend component** The Frontend component implements the external API of the KnowledgeStore by dispatching client requests to the appropriate internal components; it also controls the indexing of statements in the Virtuoso component.

As mentioned in Section 5.3, the KnowledgeStore offers through the KS Frontend two complementary endpoints: the CRUD endpoint and the SPARQL endpoint. These endpoints are made available via an HTTP ReST API that can be directly accessed by client applications or accessed indirectly through the released Java Client Library.

Figure 28 shows the detailed internal modules organization of the KnowledgeStore. It is worth noticing here that, for the sake of scalability, the KnowledgeStore is expected to be deployed on a cluster (potentially in a cloud environment), therefore additional tools are implemented to deal with the complexity of such deployment. This includes, for example, management scripts for operations at system level such as infrastructure (daemons) start-up & shut-down, data backup & restoration, statistics gathering, distributed synchronization services (e.g., Zookeeper).

[48]Methods for efficient bulk data ingestion are specific to each triple store implementation.

## 5.5 Preliminary scaling experiments

We conducted some preliminary experiments to assess the scalability of the technologies adopted in the KnowledgeStore. In particular, we performed some tests on loading the HBase component with documents, mentions, and entities, while we performed some experiments on loading and querying the triple store component (Virtuoso).

### 5.5.1 HBase and Hadoop scaling experiments

A loading test has been performed on a prototype release of the KnowledgeStore based exclusively on the HBase and Hadoop components, since the triple store was non included in that architecture.

Table 8: Loading statistics.

| Resources | Mentions | Entities | Contexts | Size |
|---|---|---|---|---|
| 235860<br>(txt 86.19%<br>img 13.65%<br>video 0.15%) | 733,738<br>(org 10.38%<br>loc+gpe 20.77%<br>per 34.22%<br>time 24.61%) | 30493 | 723 | 40.7 GB (HDFS) |

Table 8 shows the statistics in terms of Resources, Mentions, Entities and Contexts which the KS has been successfully populated with – the type of Resources and Mentions are reported in parentheses. Moreover the total occupancy of the above objects in the HDFS is also presented. Some remarks are worth noticing here: first, Entities and Contexts comes from background knowledge. Second, the infrastructure of the HBase and Hadoop components was limited to a single machine with 32 GB of RAM, hosting all the daemons needed for the pseudo-distributed setup. Such configuration has proved to be inadequate in terms of efficiency other than scalability. For the first release of the NewsReader KnowledgeStore we are using a cluster of at least four machines.

### 5.5.2 Triple Store scaling experiments

The addition of a triple store component, motivated by the need to support better access to entity data including structured (SPARQL) queries, is one of the major changes with respect to the prototype version of the KnowledgeStore previously mentioned. For that reason, specific activities were conducted in order to select a suitable triple store and assess its performance and scalability characteristics in a concrete deployment.

Three requirements were considered for selecting the triple store: (i) it must be an open source product; (ii) it must provide good SPARQL 1.1 query performances; and (iii) it must handle 500M triples in a single server deployment. The 500M triples figure is a rough estimate of the dataset size in NewsReader: it consists of a ∼400M triples subset from DBPedia with the information most relevant to NewsReader and covering ∼4M

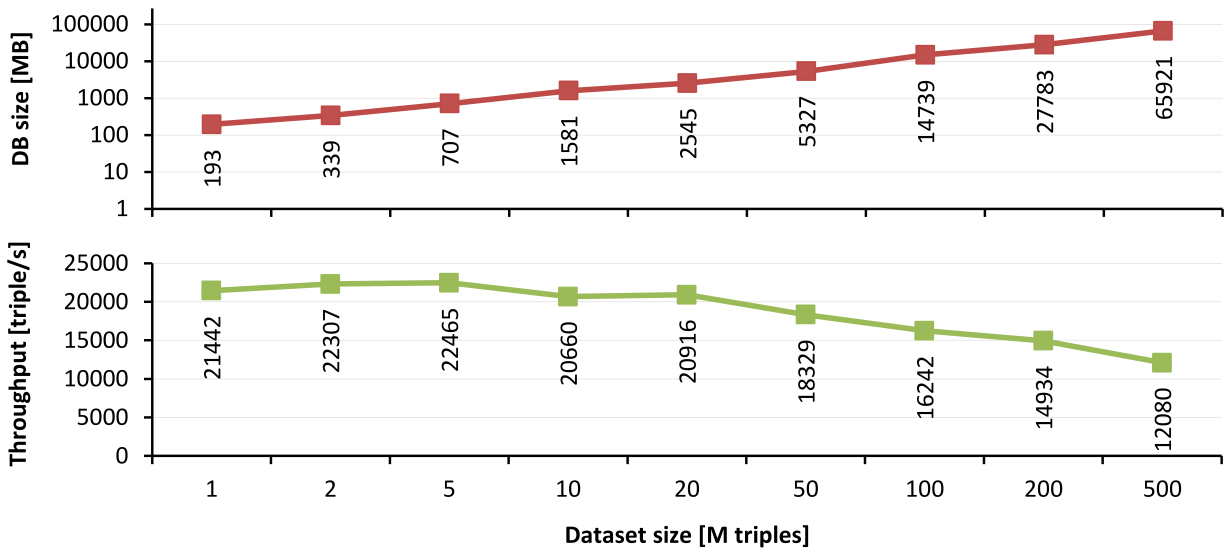


Figure 29: Performances of data loading.

entities/events, plus ∼100M triples extracted from news (roughly: 25 triples for the same 4M entities/events). Given these requirements and available triple store benchmarks[49]—in particular the April 2013 results of the Berlin SPARQL Benchmark (BSBM)[50]—we restricted our focus to the Open Source Edition of the Virtuoso triple store.[51] This edition is limited to a single server deploy (our initial environment), where it has been shown to easily handle a billion of triples given appropriate hardware; multi-server deployment with additional scalability and transparent fault tolerance can be obtained with the (commercial) Enterprise Edition.

We used the BSBM dataset generator and evaluation suite to evaluate Virtuoso in a concrete deployment with two goals: (i) better understand the hardware requirements of the triple store and its performances when deployed on a machine less powerful—and more easily available—than the one used in the April 2013 results[52]; and (ii) gather valuable knowledge on how to configure and access Virtuoso for optimal performances, to be leveraged in the subsequent development of the KnowledgeStore. More in details, we deployed Virtuoso 6.1.6 on a RedHat 6.4 (Linux 2.6) machine with an Intel$^{(R)}$ Core$^{(TM)}$ i7 CPU, 16 GB RAM and 500 GB disk (the type of machines available to the KnowledgeStore development team), and evaluated performances of data loading and SPARQL query answering with different dataset sizes (from 1M to 500M triples generated with the BSBM tool).

Figure 29 shows the size on disk and the load throughput in triples per second when varying the dataset size from 1M triple to 500M triples. Size on disk increases almost linearly, while throughput decreases for larger dataset sizes, meaning that loading time

[49]`http://www.w3.org/wiki/RdfStoreBenchmarking`

[50]`http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V7/`

[51]`http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/`

[52]Dual Intel$^{(R)}$ Xeon$^{(TM)}$ E5-2650 CPU, 256 GB RAM, three 1.8GB disks in RAID 0.

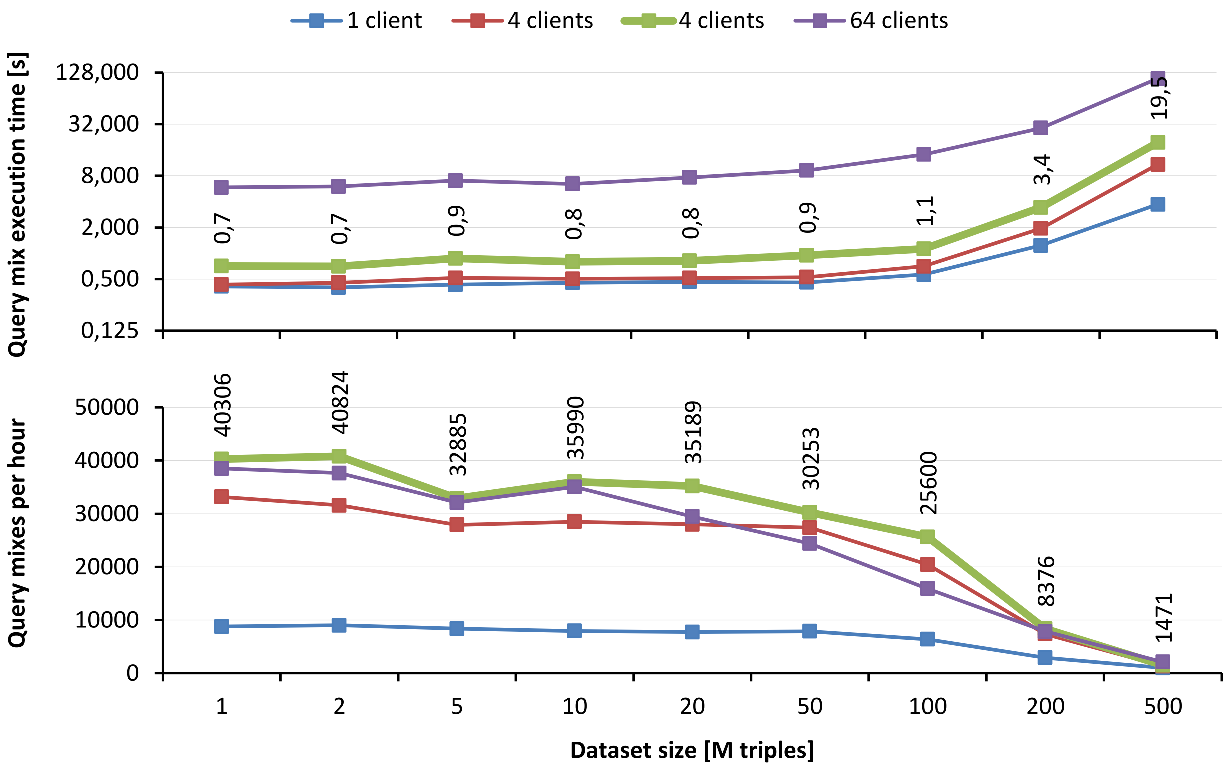


Figure 30: Performances of query answering.

increases more than proportionally with dataset size (loading 500M triples takes 11h 30m, 6.7 times rather than 5 times the 1h 42m required for loading 100M triples); however, it is worth noticing that at 500M triples a throughput of 12K triples/s is still supported.

Figure 30 shows execution time and throughput of query answering with different dataset sizes and number of concurrent clients (1, 4, 8, 64 clients). The base unit for both graphs is the 'query mix', i.e., the execution of a set of 25 parameterized queries (with random choice of parameters) over the triple store. As expected, execution time of a query mix increases with the increase of the dataset size and the number of concurrent clients. The behaviour of throughput, measured in query mixes per hour, is instead more complex. For small dataset sizes, it can be noticed that throughput increases when moving from 1 to 8 concurrent clients, while it decreases for 64 clients. This is explained by the total load being essentially limited by CPU and not by disk (in fact, for small sizes data is almost all buffered in RAM and disk is seldom accessed): as the CPU has 8 cores (4 real $\times$ 2 due to hyper-threading), 8 concurrent clients represent an optimal situation, whereas additional clients cause a decrease of throughput due to the overhead introduced by the contention of the 8 cores. For larger dataset sizes, instead, throughput figures for different numbers of clients tend to coincide. This is explained by the total load being now limited by disk and not CPU, so the overhead on the CPU introduced after 8 clients is no more relevant. It is worth noticing that this change in behaviour, as well as the major change in the steepness of execution time curves, occur around a size of 100M triples, a value for which the dataset size (∼14 GB measured on disk) approaches the amount of RAM available to Virtuoso, and after which buffering of the whole dataset in RAM is no more possible and disk accesses become increasingly frequent. This shows how RAM is the factor impacting the most on Virtuoso performances, and suggest us to deploy the triple store on a machine with as much RAM as possible.

# 6 Decision Support System

## 6.1 Introduction

The main point of end-user interaction with the NewsReader event indexes is the Decision Support System (DSS). The DSS is a graphical user interface that is meant to support users when making strategic decisions.

In essence, the DSS assists the decision making task by providing insight into the sequences of events that led up to a current situation so that a user can extrapolate to what might happen in the future.

The DSS will not aim to predict future events, but aid users in drawing their own conclusions. This is because a predictive system can only work well under the assumptions that all relevant data is available and that the mechanics of consequence are sufficiently and correctly modeled. Both of these assumptions are likely to fail in an open domain such as the news.

The main problem with investigating information structures as complex as the NewsReader event indexes is that there is a huge space of possible interactions and correlations that might be relevant. A common way to search for relevant parts of such large feature spaces is to use feature selection, regression, and other statistical techniques. However, many facts relevant to the decision making process are rare events, that by themselves will never stand out in statistics. Also, the relevance of facts can only really be judged by the end users themselves.

We attempt to solve these problems in a novel way, which has only recently become feasible due to the advances made in parallel and distributed systems and hardware graphics accelleration. As opposed to preselecting features we aim to allow the end user to interactively navigate through the large space of correlated features.

This leads us to two essential requirements. One being that the DSS needs to be a visual environment. The other, that the DSS needs to show the actual data (in the case of NewsReader, the underlying news articles from which events were extracted), with which the user is acquainted, and not derived information that can appear unfamiliar to the user.

In the remainder of this section of the deliverable we expand on these requirements (Section 6.2), demonstrate an adapted version of SynerScope's Marcato visualization tool (Section 6.3), show how Marcato will be technically embedded within the rest of the project (Section 6.4) and how it will be extended with NewsReader-specific additions. We conlude with an overview and the system requirements for Marcato.

## 6.2 Requirements

As discussed in Section 6.1, the DSS needs to 2 core characteristics in order to be a useful tool for decision making by end users: It needs to be visual and it must show the actual underlying data. In this section, we expand these generic characteristics to more specific requirements.

**No Divination** The main goal of the DSS should be to provide insight in past events. The DSS does not provide extrapolations based on confidence values or probabilities.

**No Tunnel Vision** The DSS must be able to rapidly change the perspective of the investigation, depending on the user's changing understanding of and thus changing interest in the data.

**No Reports** The DSS must be a graphical tool.

**No Magic** The DSS must show actual data, which is recognizable.

**No Hiding** The DSS must show all the data, not just aggregate statistics that hide important details.

**No Waiting** The DSS must be fast and scalable.

**No Bias** The DSS should not use a predefined vocabulary of concepts, but show whatever occurs in the data.

## 6.3 SynerScope Marcato

The technical implemenation of the DSS will largely consist of the SynerScope Marcato tool, a visual analytics application that delivers real time interaction with dynamic network-centric data. Marcato supports simultaneous views and coordinates user interaction, enabling the user to identify causal relationships and to uncover unforeseen connections. In this section we describe Marcato in more detail. Specifically, we describe how data can be imported into Marcato, which visualization options are available, and how the end-user can interact with the data in each visualization.

### 6.3.1 Importing Data

Marcato is designed to work with a very basic information schema, called the SynerScope Interface Schema (SIS). SIS consists of two object types: Nodes and Links. Links connect two Nodes. Both Nodes and Links can have additional attributes of a number of data types, including integers, floating point numbers, free text, date and time, latitude and longitude. Nodes and Links need a key attribute. This attribute is used to connect Nodes and Links. It is not predefined which kinds of data objects can be Nodes, Links, or attributes of Nodes or Links. This can be decided at the time of import.

A common decision, in the case of simple events such as transactions, communication, and interpersonal relations, is that events are modeled as Links between entity Nodes. Alternatively, events and entities can both Nodes while Links are simple associations between them.

Currently, Nodes and Links can be imported in tabular form from Relational Database Systems, like PostgreSQL[53], ParAccel[54], or MonetDB[55]. Alternatively, they can be imported from Character Separated Value (CSV) files stored on disk. The details of this process are described in Deliverable 7.1.

### 6.3.2 Visualizing Data

Marcato offers several types of visualizations. These are rendered using OpenGL to take advantage of hardware-accelerated rendering. Each visualization is described in detail in its own section below.

**Table View** The Table View provides a traditional spreadsheet view on the data. For each of the SIS data source types, i.e. each type of Nodes and each type of Links, there is a separate sheet. The Table View shows all the data as a table of values. Rows in the table correspond to either Links or Nodes, depending on which sheet is selected. The columns represent the attributes of the Nodes or Links. The user can select rows, which are then also highlighted in other Views. The Table View can also be used to sort Nodes or Links by the values of a certain field by clicking the header cells of the columns in the table.

Figure 31 shows a typical Table View.

**Hierarchy Editor View** The SIS allows for Nodes and Links. There can be multiple types of Nodes or Links. For example, if we consider Nodes to be people or books and links to be book reviews linking a person to a book they wrote a review about, then we have two Node types, people, and books, and one Link type, reviews.

In the Hierarchy Editor View it is possible to define a hierarchical ordering (of arbitrary depth) on each Node type. Each layer of the hierarchical ordering groups nodes together based on some common attribute value. For example, if the people Nodes have two attributes, "city of residence" and "age", we can define two hierarchical levels, one based on city, the other on age. Categorical attributes are grouped by distinct values, numerical attributes are distributed over a certain number of bins. The Node hierarchies can be changed in real time. All Nodes in a hierarchy categories can be selected together, which allows for quick exploration of common properties of the Nodes in that category in other Views.

Figure 32 shows the hierarchy editor on the right and the resulting hierarchy applied in the Hierarchical Edge Bundling (HEB) view (see next visualization) on the left.

**Hierarchical Edge Bundling View** The Hierarchical Edge Bundling View (HEB) is the primary network view in Marcato. Each Node is visualized as a point on a circle, and each Link is visualized as a curved line between its source and target Node. The Nodes

[53]http://www.postgresql.org/
[54]http://www.paraccel.com/
[55]http://www.monetdb.org/

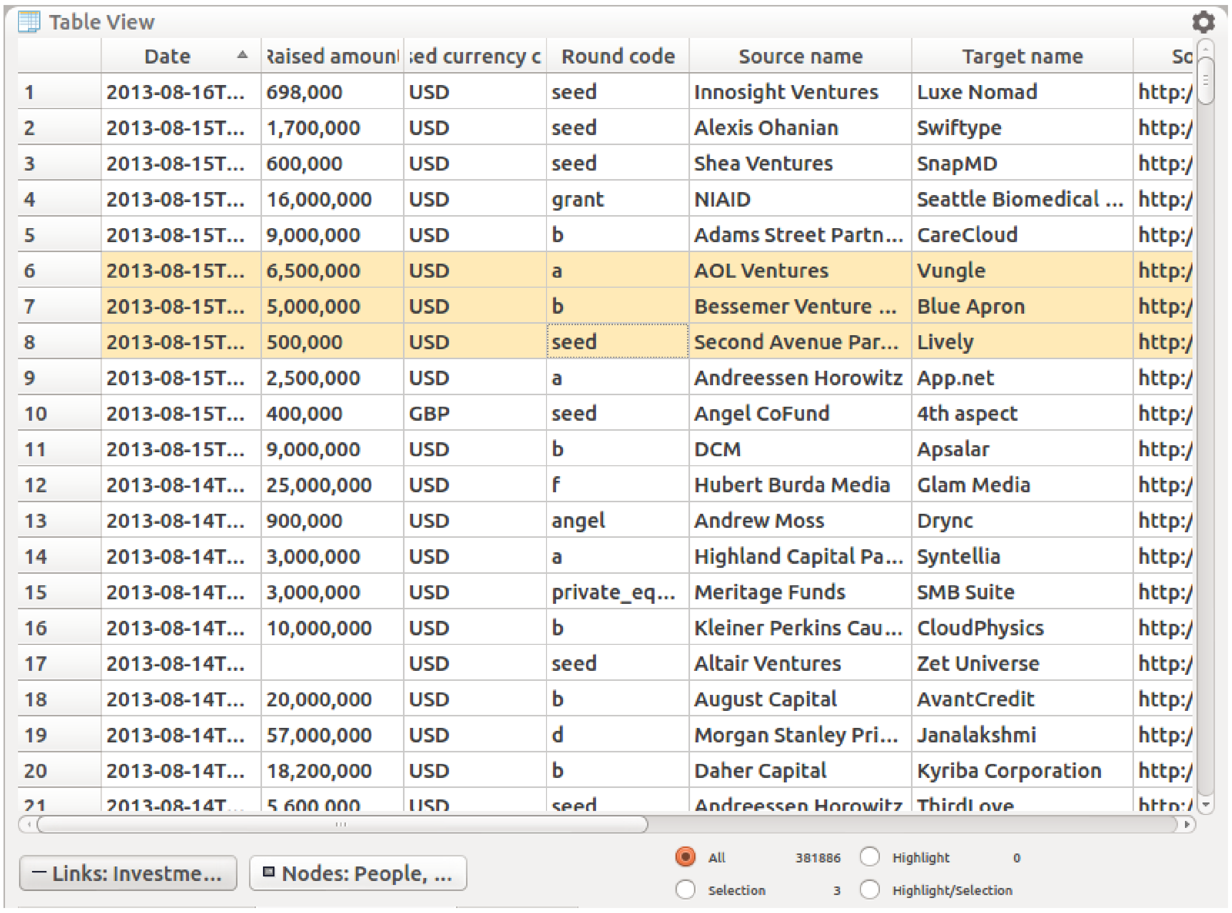
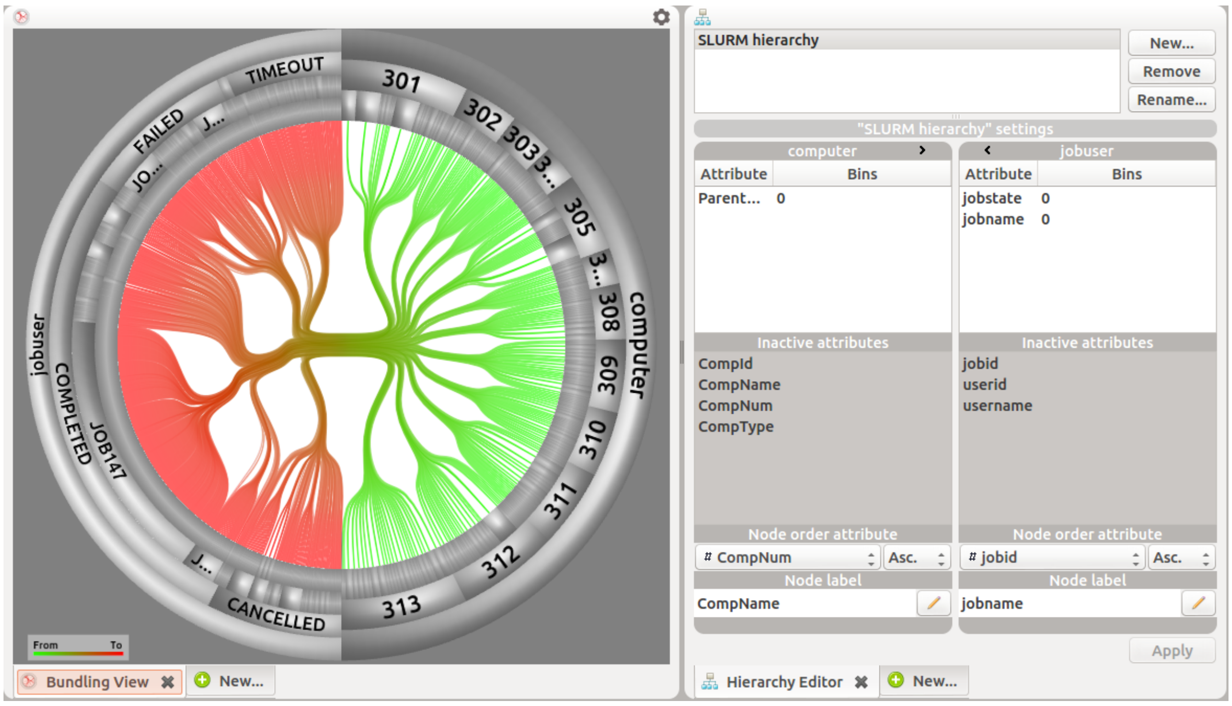


Table View

| | Date | Raised amoun | ed currency c | Round code | Source name | Target name | Sc |
|---|---|---|---|---|---|---|---|
| 1 | 2013-08-16T... | 698,000 | USD | seed | Innosight Ventures | Luxe Nomad | http:/ |
| 2 | 2013-08-15T... | 1,700,000 | USD | seed | Alexis Ohanian | Swiftype | http:/ |
| 3 | 2013-08-15T... | 600,000 | USD | seed | Shea Ventures | SnapMD | http:/ |
| 4 | 2013-08-15T... | 16,000,000 | USD | grant | NIAID | Seattle Biomedical ... | http:/ |
| 5 | 2013-08-15T... | 9,000,000 | USD | b | Adams Street Partn... | CareCloud | http:/ |
| 6 | 2013-08-15T... | 6,500,000 | USD | a | AOL Ventures | Vungle | http:/ |
| 7 | 2013-08-15T... | 5,000,000 | USD | b | Bessemer Venture ... | Blue Apron | http:/ |
| 8 | 2013-08-15T... | 500,000 | USD | seed | Second Avenue Par... | Lively | http:/ |
| 9 | 2013-08-15T... | 2,500,000 | USD | a | Andreessen Horowitz | App.net | http:/ |
| 10 | 2013-08-15T... | 400,000 | GBP | seed | Angel CoFund | 4th aspect | http:/ |
| 11 | 2013-08-15T... | 9,000,000 | USD | b | DCM | Apsalar | http:/ |
| 12 | 2013-08-14T... | 25,000,000 | USD | f | Hubert Burda Media | Glam Media | http:/ |
| 13 | 2013-08-14T... | 900,000 | USD | angel | Andrew Moss | Drync | http:/ |
| 14 | 2013-08-14T... | 3,000,000 | USD | a | Highland Capital Pa... | Syntellia | http:/ |
| 15 | 2013-08-14T... | 3,000,000 | USD | private_eq... | Meritage Funds | SMB Suite | http:/ |
| 16 | 2013-08-14T... | 10,000,000 | USD | b | Kleiner Perkins Cau... | CloudPhysics | http:/ |
| 17 | 2013-08-14T... | | USD | seed | Altair Ventures | Zet Universe | http:/ |
| 18 | 2013-08-14T... | 20,000,000 | USD | b | August Capital | AvantCredit | http:/ |
| 19 | 2013-08-14T... | 57,000,000 | USD | d | Morgan Stanley Pri... | Janalakshmi | http:/ |
| 20 | 2013-08-14T... | 18,200,000 | USD | b | Daher Capital | Kyriba Corporation | http:/ |
| 21 | 2013-08-14T... | 5,600,000 | USD | seed | Andreessen Horowitz | ThirdLove | http:/ |

[✓ All] 381886 Highlight 0
Selection 3 Highlight/Selection

Figure 31: The Table View with a Selection in orange. In this figure, each line is one investment round with properties such as a date, currency, and amount of money.



SLURM hierarchy

"SLURM hierarchy" settings

| computer | |
|---|---|
| Attribute | Bins |
| Parent... | 0 |

Inactive attributes: CompId, CompName, CompNum, CompType

Node order attribute: CompNum, Asc.

Node label: CompName

| jobuser | |
|---|---|
| Attribute | Bins |
| jobstate | 0 |
| jobname | 0 |

Inactive attributes: jobid, userid, username

Node order attribute: jobid, Asc.

Node label: jobname

Figure 32: The Hierarchy Editor View on the right side defining a hierarchy for the two Node types in this data set. The resulting respectively one and two-level hierarchy is applied to the HEB on the left.

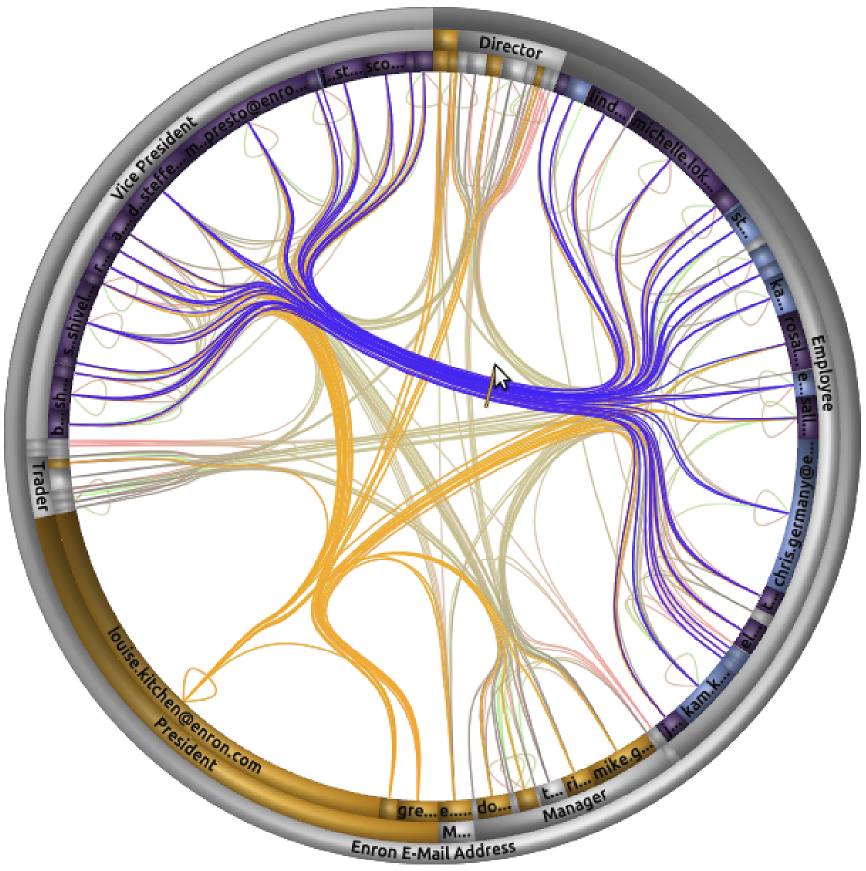


Figure 33: The Hierarchical Edge Bundling View, showing a Selection in orange, a Highlight in blue, and the overlap between the two as dark blue. The nodes on the circle represent employees (grouped categorically) while the edges between them represent e-mails. The bundle between the Vice President category and Employee category is being Highlighted with a slicing mous gesture.

are sorted so that changes in the network over time have a minimal impact on the location of the Nodes on the circle. This makes for a stable view of the network.

The Nodes are grouped hierarchically, based on the ordering defined in the Hierarchy Editor View. The Links between Nodes of the same hierarchical category are bundled together (as if they were tied together with a cable tie). The thickness of the bundle is caused by overlapping lines (each line is still shown individually) and shows the amount of interaction between categories. Bundles can be selected together by making a slicing gesture by clicking and dragging the mouse over a bundle. The HEB is illustrated in Figure 33.

**Massive Sequence View** The Massive Sequence View (MSV) is the primary temporal view in Marcato. Each Node gets a fixed position on the horizontal axis. Nodes are grouped hierarchically in the same fashion as in the HEB. Links between Nodes are represented by a

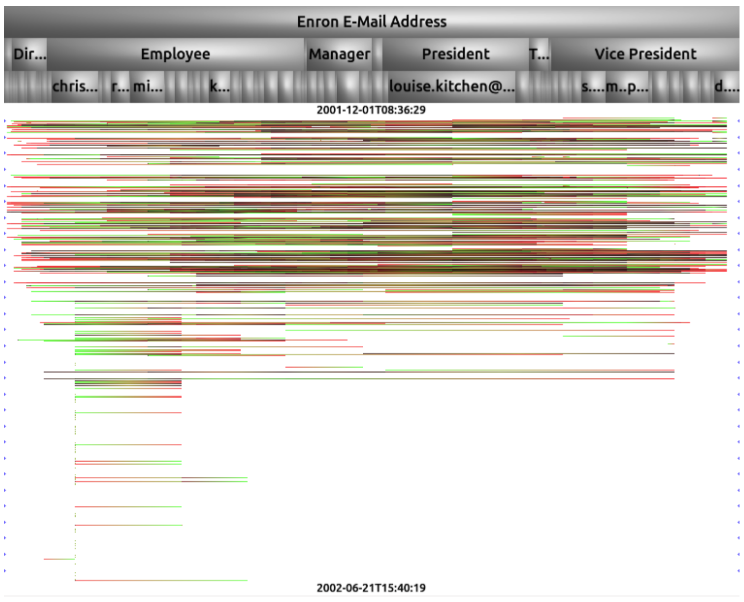


Figure 34: The Massive Sequence View, showing the same interactive hierarchy as Figure 33. Again, the nodes (now on top) represent employees (grouped categorically) while the edges between (now ordered by time from top to bottom) them represent e-mails.

horizontal line between the respective positions of the Nodes. On the vertical axis the user can select a scalar attribute, typically a time or date. This orders the Links temporally.

The MSV is typically used to inspect patterns in time. When certain categories of Nodes interact with each other in some temporal pattern, this becomes instantly recognizable in the MSV. An example of a time line, showing the same Nodes and Links as Figure 33 is shown in Figure 34.

**Map View** The Map View is the primary spatial view in Marcato. The user can select two attributes from any Node or Link data source to interpret as WGS84 latitude and longitude coordinates. These attributes are used to plot the Nodes (not the Links) on a map as points. By default, the Open Street Map[56] tile server is used over an internet connection to show a map background behind the points. The points are plotted in such a way that the user can see the point density. An additional scalar attribute can be selected to indicate point size on the map. Figure 35 shows a typical Map View presentation.

**Scatter Plot View** The Scatter Plot View uses Cartesian coordinates to relate the values of two attributes of either Nodes or Links. Dots are drawn on a two-dimensional chart, the positioning relative to the horizontal and vertical axis being determined by the attribute's values. A third attribute can used to set the size of the dots.

[56]http://www.openstreetmap.org

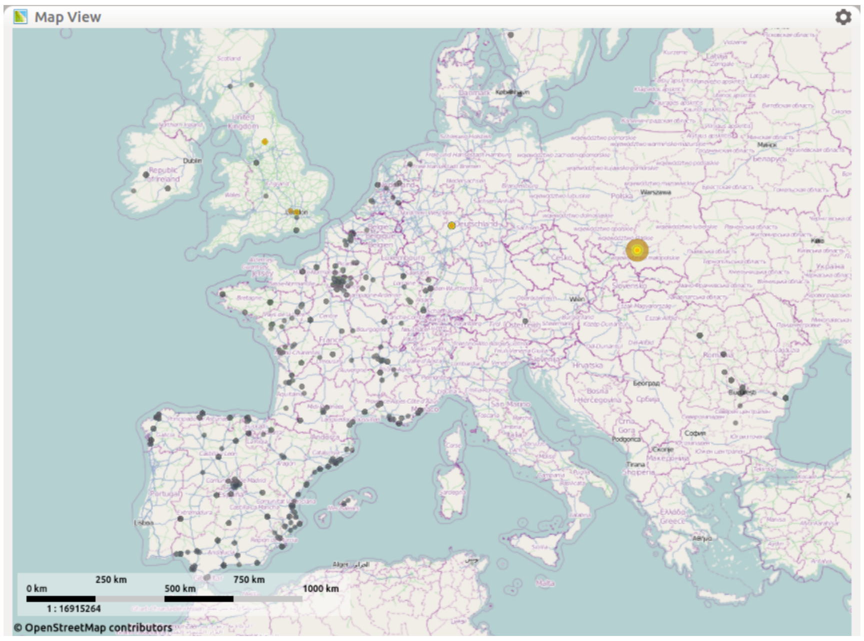


Figure 35: The Map View, showing geographic point density and size.

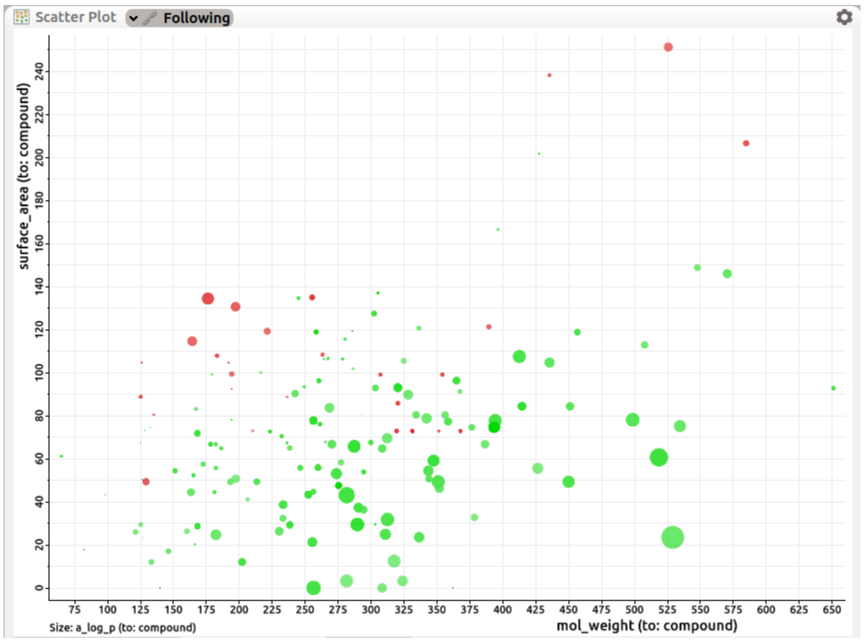


Figure 36: The Scatter Plot View, showing three variables: $x$, $y$, and magnitude, with the sign of the magnitude depicted as red (negative) or green (positive). Shown is a plot of biochemical data, specifically molecule weight ($x$) versus surface area ($y$).

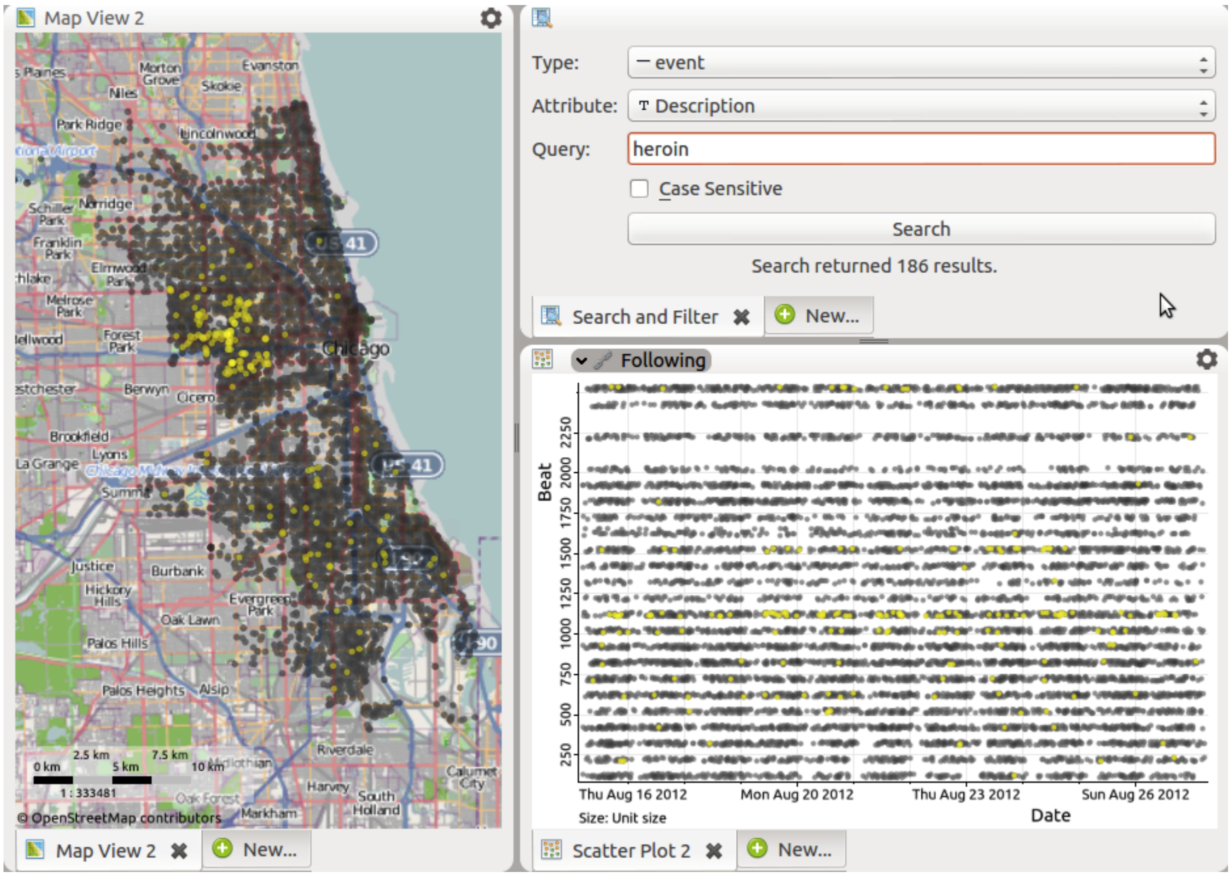


Figure 37: The Search and Filter View, showing that the search results are Selected in all other Views, in this case, the Map View and Scatter Plot View. This Figure also shows that all color coding can be customized to provide accessibility for the color blind. In this case the Selection is shown in yellow instead of orange. Shown are criminal events in the city of Chicago.

**Search and Filter View** The Search and Filter View is an interactive view that allows the user to select Nodes or Links by searching by value. The user can type in a search query that is matched against the value of a certain attribute for a given Node or Link type. The current version of the Search and Filter View allows for case sensitive or insensitive exact string matching.

### 6.3.3 Interacting with Data

The user can interact with the data in SynerScope Marcato's views in several ways: By selecting and highlighting, drilling down or up, and expanding selections. Every interaction method is coordinated across multiple views and over the node hierarchy. This principle, together with the details of each interaction method, is described below.

**Coordinated Views** SynerScope Marcato uses multiple coordinated or linked views to provide different but simultaneous visualizations of the same data. Selections and Highlights are applied across views. This means that if a node or link is highlighted in one particular view, it will also be highlighted in all other views. Similarly, if a node or link is selected in one view, it is selected in all other views.

**Selection and Highlight** One of the main interaction paradigms of Marcato is the combined use of Selections and Highlights. When a user expresses that his or her attention is focused on a certain Node, Link, or category/bundle of Nodes or Links, Marcato Highlights these objects with a color that makes them stand out from the other objects. This Highlight is simultaneously performed in all visible Views. The Highlight is changed dynamically and designed to support fast real-time interaction. When a user confirms that the Highlight is of interest by clicking then the Highlight turns into a more static Selection. The Selection stays the same until a new Selection is made. The Selection typically gets a different color than the Highlight, which allows the user to contrast the two sets to see commonalities and differences. Commonalities between the Selection and the Highlight get their own color that makes it stand out from the other objects.

**Drill down/up** There are two methods to navigate large data sets in Marcato. The first is controling the set of items that is imported into Marcato from the source database with filters. The second is Drilling down and Drilling up.

Drilling down rescopes the domain of investigation to the current Selection. Essentially, it allows the user to zoom in on an interesting part of the data set to get a more detailed view. Marcato always shows all the data, therfore drilling down does not add more detail, but by hiding the rest of the data it provides the Selection with more room to be visualized, which provides a clearer view of the individual data points.

Drilling up performs the converse operation. It rescopes the domain of investigation to the entire data set, preserving the current Selection. This can allow the user to select small parts of the data that are of interest in a Drilled down view, after which the user can Drill up to see the context of the Selection in the entire data set. An important point to note is that Drilling down can change the distribution of the data values, which makes different patterns in the data stand out.

**Network expansion** Network expansion allows the user to extend the currently Selected set of Nodes or Links with respectively Links or Nodes that are directly connected. This way it is possible to let the entire Selection grow hop-by-hop over the Link network. Network connectivity, or the lack of connectivity, can be investigated in this way.

Drilling down provides more detail, but it also cuts away the context, so another common use is to gather the local network context of a Selection of Nodes before Drilling down, which can provide the user with a minimal relevant collection of contextual information to judge the patterns in the Drilled down view. A number of fine-grained expansion tools are also available, such as expansion within the domain of the selection. This expands the currently Selected set of Nodes and Links with all the Links between currently selected Nodes that are not selected.

**Node hierarchies** The Node hierarchy that was made in the Hierarchy Editor View is always visible in a number of different Link-centric views, such as the HEB and MSV. The individual Nodes always reside at the bottom of the hierarchy as leaves on a tree. The branches of the tree are partitioned at every level, and the root of the tree corresponds to the entire current scope (either the entire data set, or the current Drill down). Each category in the hierarchy functions as a button, which when clicked selects all the Nodes it subsumes. An important point to node about the hierarchy is that it can be changed in real time. This makes it possible to rapidly change the categorization of the data, and hence the bundles of links that appear in the HEB or MSV. This means creating a specific hierarchy essentially constitutes a hypothesis that the grouping imposed by that hierarchy correlates with the actual occurrence of Links in the data set. When this hypothesis is true, then patterns will show up in the HEB or MSV, otherwise adding the hierarchy does not reveal any order.

For example, the data shown in Figure 33 is communication traffic between people. The hierarchy on the nodes is based on the role of the communicating people in their organization. The corresponding implicit hypothesis is that people's respecitive roles influence whether or not they talk to each other. In any company one would expect this hypothesis to hold. If this were not the case, then the internal communication does not correspond to the organizational structure of the company. The fact that there are clearly visible bundles in Figure 33 shows that the hypothesis is true.

## 6.4 Embedding and Extending Marcato

### 6.4.1 Connection to KnowledgeStore

The current Marcato importer provides interactive mapping functionality for the projection of tabular Node and Link data onto the SIS. Projecting RDF data onto the SIS is a slightly more complex and currently unsupported process, that requires a new specific importer to be made.

Within the NewsReader project, SynerScope will develop an interactive, graphical SPARQL query builder that allows the user to define a mapping from event data in RDF to SIS without writing program code. This query builder will be guided by the available data and help the user with building queries. The resulting query will be used to extract tabular results from a SPARQL endpoint that will then automatically be imported into Marcato without further mapping.

Additionally, within Marcato, the user should have access to the original documents from which a certain event was derived. The KnowledgeStore's CRUD API will allow for the retrieval of the original document given a certain event retrieved through its SPARQL endpoint, so SynerScope will develop a connector for the CRUD API as well.

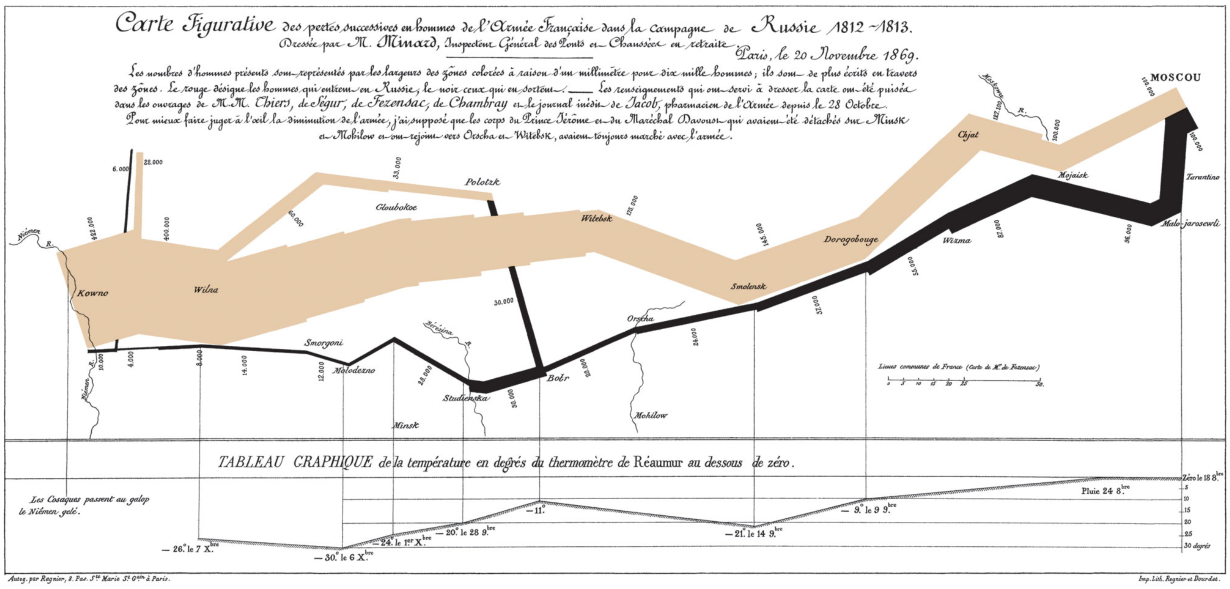

Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812–1813.
Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite.
Paris, le 20 Novembre 1869.
MOSCOU
Kowno
Wilna
Smolensk
Polotzk
Witebsk
Moïaisk
Malo-jarosewli
Tarantino
Smorgoni
Molodezno
Studienska
Minsk
Bobr
Orscha
Mohilow
Dorogobouge
Wixma
Chjat
Gloubokoe
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.
Les Cosaques passent au galop le Niémen gelé.
Pluie 24 8.bre
Zéro le 18 8.bre
– 26° le 7 X.bre
– 30° le 6 X.bre
– 24 le 1er X.bre
– 20° le 28 9.bre
– 11°
– 21° le 14 9.bre
– 9° le 9 9.bre

Figure 38: From left to right, the thickness of the beige line represents the size of Napoleon's army as it marches from Kaunas to Moscow. From right to left, the black line represents the army's size as it retreats. The line chart at the bottom shows the temperature during the retreat.

### 6.4.2 Plugin System

In addition to the existing HEB, MSV, map, scatter plot, and search-and-filter views, SynerScope Marcato can be customized by developing plugin views using HTML5 and JavaScript with AJAX calls. These plugin views can interact with the other views as part of the Multiple and Coordinated Views set through a JavaScript object that provides bidirectional communication between the Plugin View and the rest of Marcato. This allows users to make Javascript routines that influence the current Highlight and Selection or that access the data. HTML5-based views are typically much less performant than the other, QT and OpenGL-based Views.

**Narrative Charts** Within the NewsReader project, one of the additions to Marcato will be an implementation of interactive Narrative Charts as a Marcato plugin. Narrative charts show the movements and interactions of actors over time. An early example of a narrative chart (shown in Figure 38) is the flow map drawn by Charles Joseph Minard to visualize Napoleon's troop movements during his Russian campaign of 1812 [Minard, 1869]. A more recent example developed at the University of Waterloo [57] is a chart of character interactions in a Tintin comic, shown in Figure 39.

Suppose a user has configured Marcato to show entities as Nodes with the Links between entities being events. This will allow the user to visualize the storylines of a limited

[57] http://csclub.uwaterloo.ca/˜n2iskand/?page_id=13

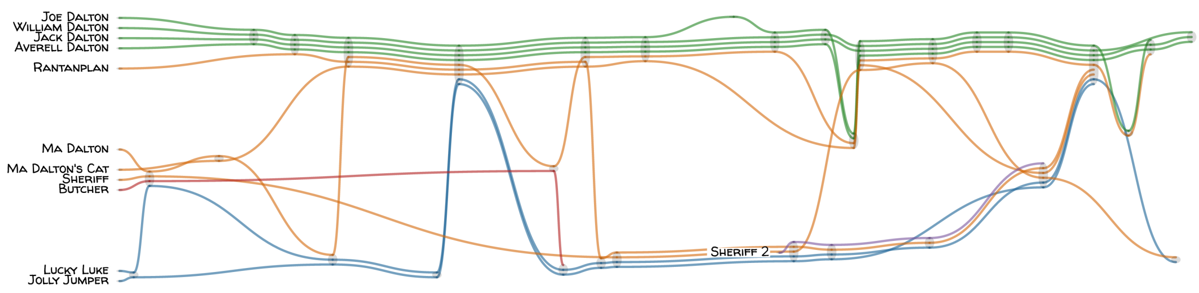


Figure 39: The point where the lines (originating at a character's name on the left) meet represents an interaction of those characters in the story.

Selection of Nodes/Entities as a Narrative Chart. The Narrative Chart will be a Coordinated View in Marcato. This means, for example, that the user can Select a number of Links/Events in the HEB and the corresponding Narrative Chart will be rendered automatically when the Narrative Chart Plugin View is open. Highlighting parts of the Narrative Chart will consequently Highlight the corresponding parts of other views, including the HEB.

Narrative Charts are related to the Massive Sequence View. The main difference between the two is that in the MSV Nodes have a fixed position depending on the current hierarchy, which in Narrative Charts they have a varying location depending on which Nodes/Entities they share Links/Events with in a certain time frame. Another difference is that in the MSV, Links/Events are shown as lines between Nodes/Entities that bridge the gap between the source and target Node/Entity, while in Narrative Charts the Nodes/Entities move towards each other so that Links/Events become a point. This makes it impossible to draw a consistent visualization when a Node/Entity can have simultaneous Links/Events to two unrelated and hence distant Nodes/Entities.

Force direction can partially overcome this problem, but will not solve it completely. Narrative Charts can be very informative for a limited number of events and participants, but become incomprehensible for large numbers. This, combined with the limited performance of Web plugins motivates the decision to limit the scope of Narrative Charts to a small number of storylines in the initial version of the DSS.

**Web Lookup** Another addition that will be made to Marcato is a Web Lookup plugin that performs automatic lookup of contextual information over a HTTP connection and presents the results either in the form of a ranked result list, like results from a search engine, or in the form of the actual displayed contextual information, like Web pages in a browser. The Web Lookup plugin will have two modes of interaction.

First, users will be able to find documents without typing in a query, but simply by selecting interesting patterns in the storylines shown in the rest of Marcato. The corresponding source documents will be automatically retrieved and displayed. For instance, there is a sudden burst of activity surrounding a certain Node that strikes the user in the

MSV. Selecting this burst will yield the corresponding news items that allow the user to directly judge the relevance of the burst.

Second, users will be able to use the Web Lookup plugin as a search engine to find documents and automatically see the surrounding storyline context of the content of that document in the rest of Marcato. For instance, the user will be able to search for a certain topic by tying in a query in a search box which will yield a number of documents, ranked by relevance. These documents discuss parts of the storylines that touch upon this topic. The corresponding Nodes and Links will be automatically selected in the rest of Marcato.

## 6.5 Conclusion

Marcato has all of the properties that are required for the interactive manipulation of events from the news, but has a few technical shortcomings. Most notably, it requires interfacing with the NewsReader KnowledgeStore (see Section 6.4), and a number of NewsReader specific visualization techniques, such as narrative charts (i.e. metro line pictures of the life lines of actors) and context visualization (e.g. hypertext view of the original news articles that describe parts of the investigated story).

We conclude our overview of the DSS by noting that SynerScope Marcato can be run locally on the end users own machine, if said machine meets the minimum hardware and software requirements as defined in Deliverable 7.1. Alternatively, Marcato can be run remotely on a virtual machine in the cloud and streamed to the end users machine. This allows users to user the Decision Support Tool Suite from their own computer regardless of the technical specification, even a tablet computer is sufficient, as long as the network access to the cloud server is sufficiently stable.

# 7 Conclusions and Future Work

This deliverable describes the first version of the **System Design** architecture developed in NewsReader to process large and continuous streams of English, Dutch, Spanish and Italian news articles.

We have described many aspects related to the system design, such as:

- Describe a general architecture for event extraction which is capable to scale and process a large number of documents in short time.
- Define the NAF data format which is the basis for the inter-operability of the NLP tools integrated into the project, as well as a proper API and library for NLP modules to deal with NAF annotations.
- Describe the GAF and SEM data formats for formally representing events structures which is independent of the mentions as present in the documents.
- Describe the central data repository, the KnowledgeStore, which enables to jointly store, manage, retrieve, and semantically query its contents.
- Describe the data visualization module, which represents highly dense and dynamic data as produced by the project, thus allowing decision markers to make well informed decisions.

In the future, we plan to implement a fully parallel architecture for streaming processing which is able to continuously accept new documents and process them quickly. This architecture will allow many modules work on the same document in parallel. It will also allow having many copies of the most time consuming tasks. The experiments done so far suggest a significant boost in the performance.

Regarding the KnowledgeStore, we plan to evaluate it from a functional perspective based on its capability to (i) store an overwhelming daily stream of economical and financial contents (news articles and data), (ii) support a complex NLP pipeline in extracting knowledge from those contents, and (iii) provide suitable online and offline query capabilities for use in a decision support tool for professional decision-makers. In the same context, we also plan to carry out an extensive performance evaluation to test the scalability of the KnowledgeStore in a real (clustered) production environment, in terms of scalability with respect to data size, query load, and tolerance to nodes and network failures. Specific experiments will be carried out to test the average response times of typical and most frequently used KnowledgeStore services, including, for example: (i) loading of a daily set of news and extracted mentions in NAF format; (ii) lookup of each type of object; and (iii) retrieval of sets of objects filtered by typical conditions.

Regarding the DSS system, we plan to run remotely the DSS component on a virtual machine in the cloud and streamed to the end users machine. This allows users to user the Decision Support Tool Suite from their own computer regardless of the technical specification, even a tablet computer is sufficient, as long as the network access to the cloud server is sufficiently stable.

## References

[Agerri *et al.*, 2013] Rodrigo Agerri, Itziar Aldabe, Zuhaitz Beloki, Egoitz Laparra, Maddalen Lopez de Lacalle, German Rigau, Aitor Soroa, Marieke van Erp, Piek Vossen, Christian Girardi, and Sara Tonelli. Event detection, version 1. NewsReader Deliverable 4.2.1, 2013.

[Allemang and Hendler, 2009] D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Safari Books Online. Elsevier Science, 2009.

[Alon *et al.*, 1996] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. ACM.

[Baker *et al.*, 2003] Collin F. Baker, Charles J. Fillmore, and Beau Cronin. The structure of the FrameNet database. *International Journal of Lexicography*, 16(3):281–296, 2003.

[Beckett, 2004] Dave Beckett. RDF/XML syntax specification (revised). Recommendation, W3C, February 2004. `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`.

[Bejan and Harabagiu, 2010] Cosmin Bejan and Sandra Harabagiu. Unsupervised event coreference resolution with rich linguistic features. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1412–1422, 2010.

[Bizer *et al.*, 2009] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[Bosma *et al.*, 2009] Wauter Bosma, Piek Vossen, Aitor Soroa, German Rigau, Maurizio Tesconi, Andrea Marchetti, Monica Monachini, and Carlo Aliprandi. KAF: a generic semantic annotation format. In *Proceedings of the 5th International Conference on Generative Approaches to the Lexicon GL 2009*, Pisa, Italy, 2009.

[Bozzato and Serafini, 2013] Loris Bozzato and Luciano Serafini. Materialization calculus for contexts in the semantic web. In *Proceedings of the 26th Description Logics Workshop*, 2013.

[Bozzato *et al.*, 2012] Loris Bozzato, Francesco Corcoglioniti, Martin Homola, Mathew Joseph, and Luciano Serafini. Managing contextualized knowledge with the ckr (poster). In *Proceedings of the 9th Extended Semantic Web Conference (ESWC 2012)*, May 27-31 2012.

[Brito *et al.*, 2011] Andrey Brito, André Martin, Thomas Knauth, Stephan Creutz, Diogo Becker, Stefan Weigert, and Christof Fetzer. Scalable and Low-Latency Data Processing with StreamMapReduce. In *3rd IEEE International Conference on Cloud Computing Technology and Science*, pages 48 –58. IEEE Computer Society, Nov 2011.

[Bryl *et al.*, 2010] Volha Bryl, Claudio Giuliano, Luciano Serafini, and Kateryna Tymoshenko. Supporting natural language processing with background knowledge: Coreference resolution case. In *Proc. of 9th Int. Semantic Web Conference (ISWC'10)*, volume 6496 of *LNCS*, pages 80–95. Springer, 2010. `http://dx.doi.org/10.1007/978-3-642-17746-0_6`.

[Carroll *et al.*, 2005] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *Proc. of the 14th Int. Conference on World Wide Web (WWW'05)*, pages 613–622, New York, NY, USA, 2005. ACM. `http://doi.acm.org/10.1145/1060745.1060835`.

[Ceolin *et al.*, 2010] Davide Ceolin, Paul Groth, and Willem Robert Van Hage. Calculating the trust of event descriptions using provenance. *Proceedings Of The SWPM*, 2010.

[Chambers and Jurafsky, 2011] Nathanael Chambers and Dan Jurafsky. Template-based information extraction without the templates. In *Proceedings of ACL-2011*, 2011.

[De Bruijn and Heymans, 2007] Jos De Bruijn and Stijn Heymans. Logical foundations of (e)RDF(S): complexity and reasoning. In *Proc. of 6th Int. Semantic Web Conference (ISWC'07) and 2nd Asian Semantic Web Conference (ASWC'07), Busan, Korea*, pages 86–99, Berlin, Heidelberg, 2007. Springer-Verlag. `http://dl.acm.org/citation.cfm?id=1785162.1785170`.

[Dean and Ghemawat, 2008a] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[Dean and Ghemawat, 2008b] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[Feigenbaum *et al.*, 2013] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. SPARQL 1.1 protocol. Recommendation, W3C, March 2013. `http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/`.

[Ferrucci *et al.*, 2010] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An overview of the DeepQA Project. *AI Magazine*, 31(3), 2010. `http://www.aaai.org.proxy.lib.sfu.ca/ojs/index.php/aimagazine/article/view/2303`.

[Fokkens *et al.*, 2013] Antske Fokkens, Marieke van Erp, Piek Vossen, Sara Tonelli, Willem Robert van Hage, Luciano Serafini, Rachele Sprugnoli, and Jesper Hoeksema.

GAF: A grounded annotation framework for events. In *Proceedings of the first Workshop on Events: Definition, Dectection, Coreference and Representation*, Atlanta, USA, 2013.

[Gantz and Reinsel, 2011] John Gantz and David Reinsel. Extracting value from chaos. Technical report, IDC Iview, June 2011. `http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf`.

[Gao and Hunter, 2011] Lianli Gao and Jane Hunter. Publishing, linking and annotating events via interactive timelines: an earth sciences case study. In *DeRiVE 2011 (Detection, Representation, and Exploitation of Events in the Semantic Web) Workshop in conjunction with ISWC 2011*, Bonn, Germany, 2011.

[Grishman and Sundheim, 1996] Ralph Grishman and Beth Sundheim. Message understanding conference - 6: A brief history. In *Proceedings of the 16th conference on Computational linguistics (COLING'96)*, pages 466–471, 1996.

[Gusev *et al.*, 2010] Andrey Gusev, Nathanael Chambers, Pranav Khaitan, Divye Khilnani, Steven Bethard, and Dan Jurafsky. Using query patterns to learn the duration of events. In *Proceedings of ISWC 2010*, 2010.

[Hellmann *et al.*, 2013] Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. Integrating nlp using linked data. In *Proceedings of the 12th International Semantic Web Conference (ISWC)*, 2013.

[Ide *et al.*, 2003] Nancy Ide, Laurent Romary, and Éric Villemonte de La Clergerie. International standard for a linguistic annotation framework. In *Proceedings of the HLT-NAACL 2003 Workshop on Software Engineering and Architecture of Language Technology Systems (SEALTS)*. Association for Computational Linguistics, 2003.

[Linguistic Data Consortium, 2004a] Linguistic Data Consortium. Annotation Guidelines for Event Detection and Characterization (EDC). `http://projects.ldc.upenn.edu/ace/docs/EnglishEDCV2.0.pdf`, 2004.

[Linguistic Data Consortium, 2004b] Linguistic Data Consortium. The ACE 2004 Evaluation Plan. Technical report, LDC, 2004.

[McCreadie *et al.*, 2013] Richard McCreadie, Craig Macdonald, Iadh Ounis, Miles Osborne, and Sasa Petrovic. Scalable distributed event detection for twitter. In *Proceedings of IEEE International Conference on Big Data*, 2013.

[Mendes *et al.*, 2011] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 1–8, 2011.

[Minard, 1869] Charles Joseph Minard. *Carte figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813*. Regnier et Dourdet, Paris, 1869.

[Moens *et al.*, 2011] Marie-Francine Moens, Oleksandr Kolomiyets, Emanuele Pianta, Sara Tonelli, and Steven Bethard. D3.1: State-of-the-art and design of novel annotation languages and technologies: Updated version. Technical report, TERENCE project – ICT FP7 Programme – ICT-2010-25410, 2011.

[Motik *et al.*, 2009] Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language structural specification and functional-style syntax. Recommendation, W3C, October 2009. `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`.

[Neumeyer *et al.*, 2010] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ICDMW '10, pages 170–177, Washington, DC, USA, 2010. IEEE Computer Society.

[Nothman *et al.*, 2012] Joel Nothman, Matthew Honnibal, Ben Hachey, and James R. Curran. Event linking: Grounding event reference in a news archive. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 228–232, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[Palmer *et al.*, 2005] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2013/03/12 2005.

[Pustejovsky *et al.*, 2006] James Pustejovsky, Jessica Littman, Roser Saurí, and Marc Verhagen. Timebank 1.2 documentation. Technical report, Brandeis University, April 2006.

[Pustejovsky *et al.*, 2010] James Pustejovsky, Kiyong Lee, Harry Bunt, and Laurent Romary. Iso-timeml: An international standard for semantic annotation. In *LREC*, 2010.

[Rizzo and Troncy, 2011] Giuseppe Rizzo and Raphaël Troncy. NERD: A framework for evaluating named entity recognition tools in the Web of data. In *Workshop on Web Scale Knowledge Extraction, colocated with ISWC 2011*, 2011.

[Rospocher *et al.*, 2013] Marco Rospocher, Francesco Corcoglioniti, Roldano Cattoni, Bernardo Magnini, and Luciano Serafini. Interlinking unstructured and structured knowledge in an integrated framework. In *Proc. of 7th IEEE International Conference on Semantic Computing (ICSC), Irvine, CA, USA*, 2013. (to appear).

[Sakr *et al.*, 2013] Sherif Sakr, Anna Liu, and Ayman G. Fayoumi. The family of mapreduce and large scale data processing systems. *CoRR*, abs/1302.2966, 2013.

[Saurí and Pustejovsky, 2009] Roser Saurí and James Pustejovsky. Factbank: a corpus annotated with event factuality. *Language resources and evaluation*, 43(3):227–268, 2009.

[Seaborne and Harris, 2013] Andy Seaborne and Steve Harris. SPARQL 1.1 query language. Recommendation, W3C, March 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[Setzer and Gaizauskas, 2000] Andrea Setzer and Robert J. Gaizauskas. Annotating events and temporal information in newswire texts. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, Athens, Greece, 2000.

[Tonelli and Sprugnoli, 2013] Sara Tonelli and Rachele Sprugnoli. Definition of the annotation module, version 1. NewsReader Deliverable 4.2.1, 2013.

[van Hage *et al.*, 2011] Willem Robert van Hage, Véronique Malaisé, Roxane Segers, Laura Hollink, and Guus Schreiber. Design and use of the Simple Event Model (SEM). *J. Web Sem.*, 9(2):128–136, 2011. `http://dx.doi.org/10.1016/j.websem.2011.03.003`.

[Van Hage *et al.*, 2012] Willem Robert Van Hage, Marieke Van Erp, and Véronique Malaisé. Linked open piracy: A story about e-science, linked data, and statistics. *Journal on Data Semantics*, 1(3):187–201, 2012.

[Worboys and Hornsby, 2004] Michael Worboys and Kathleen Hornsby. From objects to events: Gem, the geospatial event model. In *Geographic Information Science*, pages 327–343. Springer, 2004.

[Yu and Chen, 2013] Wei Yu and Junpeng Chen. The state-of-the-art in web-scale semantic information processing for cloud computing. *CoRR*, abs/1305.4228, 2013.

# A NewsReader Virtual Machines

This document is a guideline for copying the NewsReader Virtual Machine (VM), as well as for installing new NLP modules into the VMs.

## A.1 Files

You need two files for running the VM.

```
https://siuc05.si.ehu.es/~sisfetek/datuak_deskargatzeko/centos64newsreader.img
https://siuc05.si.ehu.es/~sisfetek/datuak_deskargatzeko/newsreader-vm.xml
```

The first is a big file with an "empty" VM with Centos 6.4 Linux operating system and one NLP module installed. The second is an short XML document needed for running the VM.

## A.2 Prerequisites

The are some pre-requisites the host machine has to fulfil for running the VM inside it:

- Linux operating system (any recent flavor would do it)
- In-kernel KVM virtualization capabilities. You can also determine if your system processor supports KVM by running the following command:

```
% grep -E 'vmx|svm' /proc/cpuinfo
```

if this command returns output, then your system supports KVM. You also have to verify that the KVM-related feature is enabled in the machine's BIOS.

- 64 bit CPU (x86_64)

## A.3 Running the VM

For running the VM, follow these steps:

### A.3.1 Preliminary steps (do it once):

1. Install the necessary software into the host machine. On Deban/Ubuntu machines this includes the following packages:
    - `qemu-kvm`
    - `libvirt-bin`
    - `virt-manager`

2. Download `centos64newsreader.img` and `newsreader-vm.xml` into the host system.
3. Make sure that the path to the `centos64newsreader.img` file is accessible/readable to the user `qemu`
4. Make a copy of the XML doc and rename it to a proper name. For the sake of this document, the XML doc name will be `newsreader-EHU.xml`
5. Tweak the the XML file:
   (a) Put a proper name into the `<name>` element (line 3). For example, "newsreader-EHU".
   (b) Create a new UUID using 'uuidgen' program and paste it lo line 5 (into `<uuid>` element)
   (c) Put the absolute path to the IMG file in line 27 in the "file" attribute of `<source>` element.
   (d) Currently the VM is configured to use 8Gb RAM. You can change this value by editing around line 6 (`<memory>` and `<currentMemory>` elements).
6. If you experience problems running the VM, maybe you need to change line 23 and put the name of the kvm emulator executable in your system.
7. Alternatively, you can create a bare new VM using the IMG image. Use the `virt-manager` tool for this. The following link maybe useful:

`https://docs.google.com/document/d/1exv1X3zmtGT6lZihlKW9T-EXKLzj_ODOWmndMe4I-c8/edit?usp=sharing`

### A.3.2 Accessing the VM

From the host machine, cd to where the IMG and XML documents are and run the following:

```
% virsh create newsreader-EHU.xml
Domain newsreader-EHU created from newsreader-EHU.xml
```

The machine should be running now. You can test this using the `virsh list` command:

```
% virsh list
 Id    Name                           State
----------------------------------------------------
 17    newsreader-EHU                 running
```

Note: The VM needs around 3/4 minutes to completely load all the modules and services, so please be patient until the login screen appears.

Now you can connect to the VM. There are several ways to run it:

### Connecting from the console

Being on the host computer you can connect from the console using this command:

```
% virsh console newsreader-EHU
Connected to domain newsreader-EHU
Escape character is ^]
```

You can exit from the VM at any time by pressing the ˆ] key (Ctrl + ']').

### Connecting from VNC

Start VNC in the host machine and connect to "localhost" using the "VNC" protocol. It will automatically show the console of the VM. Alternatively, run the `virt-manager` program, and double click into the running VM. It will open a console window.

### Connecting via ssh

The VM is configured to get an IP address using DCHP. The VM receives a local IP address `192.168.122.X` from inside the host machine. To exactly know which local IP it has, you have to first access the VM via console or VNC. Once inside, you can get the IP address of the VM for instance running the following command:

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:8E:CD:B1
          inet addr:192.168.122.98  Bcast:192.168.123.255  Mask:255.255.252.0
          inet6 addr: fe80::5054:ff:fe8e:cdb1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:568 errors:0 dropped:0 overruns:0 frame:0
          TX packets:253 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:49026 (47.8 KiB)  TX bytes:83040 (81.0 KiB)
```

In this case, the IP is `192.168.122.98` (look at the `inet addr` section).

Once we know which local IP the VM has, and being on the host computer, just ssh to the VM. In the example above, just type:

```
% ssh  newsreader@192.168.122.98
```

If you want to access the VM guest outside the host machine, perhaps the best way is to use a bridged network configuration (not explained here). Alternatively, you can use iptables for allowing external access through ssh to the VM. For example, you can access guest's ssh port using host's 2222:

```
iptables -t nat -A PREROUTING -p tcp --dport 2222 -j DNAT --to-destination 192.168.12
iptables -t nat -A POSTROUTING -p tcp --dport 22 -d 192.168.122.98 -j SNAT --to 192.1
iptables -D FORWARD 5 -t filter
iptables -D FORWARD 4 -t filter
```

In any case, consult with your IT staff to perform the above steps, as there are many alternatives.

### Changing IP

You can set an static IP for the VM by editing `/etc/sysconfig/network-scripts/ifcfg-eth0` file inside the VM. For example, this lines would assign local IP `192.168.122.99`:

```
DEVICE=eth0
HWADDR=52:54:00:8e:cd:b1
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
IPADDR=192.168.122.99
NETMASK=255.255.252.0
GATEWAY=192.168.122.1
DEFROUTE=yes
```

and then restarting network service:

```
$ /etc/init.d/network restart
```

## A.4 Shut down the VM

Logout from VM user and then, from the host computer:

```
% virsh destroy newsreader-ixa
```

Alternatively, and being on the VM, run the following command:

```
$ sudo shutdown -h now
```

## A.5 User and password

The VM has one user:

```
login: newsreader
pwd: ----------
```

the `root` user has the same password as `newsreader`. You can run root commands within the newsreader user using the `sudo` command.

## A.6 Directory structure

All the NLP modules and document directories are under the `/home/newsreader` directory. This directory structure is as follows:

`~/components`

Here lay the actual NLP modules.

`~/opt`

The dependencies of the modules should be installed under this directory (not system-wide). The idea is that we can synchronize the `~/components` and `~/opt` directories when a module is updated or a new module is deployed.

`~/docs`

The documents are stored here. Initially, input documents are placed in `~docs/input`. When the document is successfully processed, the compressed output NAF is placed into the `~docs/output` directory and removed from `~docs/input`. Alternatively, if the document can not be processed, it is moved to the `~docs/error` directory (and removed from the original place).

## A.7 Using the NLP pipeline

This section describes how to actually use the pipeline. Documents are uploaded to the VM from outside, typically from the host machine. In th examples we use `IP` and `PORT` for specifying the VM IP address and the port of the service. See section on `Changing IP` above to know which IP the VM has. `PORT` will be usually 80.

### A.7.1 Sending documents to VM

Use the `curl` command to send documents to the processing pipeline. For sending the document `doc.txt` to the virtual machine (with `IP` and `PORT`), use the following command:

```
% curl --form "file=@doc.txt" http://IP:PORT/cm_upload_text_file.php
```

### A.7.2 Getting output documents

As said above, the documents uploaded to the VM are stored in the `~docs/input` directory. Once the document is processed, and if there is no error, the output NAF will be put in `~docs/output`. The name of the output NAF will be:

```
~docs/output/name.extension_MD5.naf.bz2
```

In the example above, the `doc.txt` document could get a name like:

```
~docs/output/doc.txt_8b45b51a553d702777bc627f262ea091.naf.bz2
```

Note that the output documents are compressed using the `bzip2` program.

### A.7.3 Status of NLP processing

The VM has a service for knowing its internal status. Running this command:

```
% curl IP:PORT/cm_sysinfo.php
```

it gives information about the VM status:

```
VM: newsreader-EHU
Uptime:  15:25:58 up 7 days,  3:16,  7 users,  load average: 2.41, 1.58, 0.91
Free Memory:          1201708 kB
Free Disk: 12940.1640625 MB

Pipeline processing status:
Pending files-> 5
Finished files-> 3
Failed files-> 2
```

## A.8 Deploying NLP modules

This Section explains how to deploy new NLP modules into the VM. All the modules should be installed under the `newsreader` user. The directory structure is as follows:

- Install the modules under the `~/components` directory, creating a subdirectory as appropriate (for instance, `~/components/EHU-ukb`).
- If the modules have dependencies, install the dependencies into the `~/opt` directory. If this is not an option, please let us know.
- There has to be a `run.sh` script inside each module which reads input from `STDIN`, runs the module, and write the output (NAF) to `STDOUT`. This script has no parameters.
- The `run.sh` script has to be callable (and will be called) from outside the directory where the module is. So make sure the `run.sh` script uses absolute paths or `cd`'s into the component's directory first.
- Please create an `INSTALL` document inside the module clearly specifying which steps are needed for deploying the module (how to install dependencies, etc).

You will find an example of a deployed module in `~components/EHU-ukb`.

## A.9 Updating NLP modules

Modules are updated on the "master" VM at EHU. The address of the master VM is `u017940.si.ehu.es`, and the ssh port is 2223. Thus, the way to connect to the master VM is:

```
ssh -p 2223 newsreader@u017940.si.ehu.es
```

Once the modules are updated on the master VM, each VM copy can synchronize and update the modules by just running the following command:

```
~/update_nlp_components.sh
```

The script will connect to the master VM (asks for the usual password), and update all components.

## A.10 If something goes wrong

If the processing stalls or there is any other kind of problems, the easiest way to proceed is to just reboot the VM. From a shell command inside the VM, just run

```
$ sudo shutdown -r now
```

and the system will reboot. When the machines starts again, it will scan the input doc directory and start processing the documents present there. Remember that the machine needs 3/4 minutes to boot and launch all the services and daemons.

If you have any question, please do not hesitate to contact us:

```
Kike Fernandez <kike.fernandez@ehu.es>
Aitor Soroa <a.soroa@ehu.es>
```