DELIVERABLE SUBMISSION SHEET

To: Susan Fraser *(Project Officer)*

EUROPEAN COMMISSION
Directorate-General Information Society and Media
EUFO 1165A
L-2920 Luxembourg

From:

Project acronym: PHEME Project number: 611233

Project manager: Kalina Bontcheva

Project coordinator The University of Sheffield (USFD)

The following deliverable:

Deliverable title: PHEME Integrated Veracity Framework – v1.0 (Revised)

Deliverable number: D6.1.2

Deliverable date: 31 December 2015

Partners responsible: ATOS Spain SA (ATOS)

Status: ☒ Public ☐ Restricted ☐ Confidential

is now complete. ☒ It is available for your inspection.
☒ Relevant descriptive documents are attached.

The deliverable is:

☒ a document
☐ a Website (URL: .....)
☐ software (.....)
☐ an event
☐ other (.....)

Sent to Project Officer:	Sent to functional mail box:	On date:
Susan.Fraser@ec.europa.eu	*CNECT-ICT*-611233*@ec.europa.eu*	12 May 2016

FP7-ICT Strategic Targeted Research Project PHEME (No. 611233)

Computing Veracity Across Media, Languages, and Social Networks

D6.1.2 PHEME Integrated Veracity Framework - v 1.0

Mateusz Radzimski, Iván Martínez Rodriguez, Tomás Pariente Lobo (ATOS) / Arkaitz Zubiaga (UWAR) / Kalina Bontcheva, Leon Derczynski, Srijith Prabhakaran Nair Kusumam (USFD) / Georgi Georgiev, Laura Tolosi (ONTO) (ONTO) / Thierry Declerck, Piroska Lendvai (USAAR) / Arno Scharl (MODUL)

Abstract

FP7-ICT Strategic Targeted Research Project PHEME (No. 611233)

Deliverable D6.1.2 (WP 6)

This document summarises the activities carried out related to the data and software integration of PHEME

Keyword list: integration, PHEME architecture, acquisition framework

---

Nature: Prototype

Dissemination: PU

Contractual date of delivery: 31/12/2015

Actual date of delivery: 15/01/2016

Reviewed By: Kalina Bontcheva

Web links: www.pheme.eu

PHEME Consortium

This document is part of the PHEME research project (No. 611233), partially funded by the FP7-ICT Programme.

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP, UK
Tel: +44 114 222 1930
Fax: +44 114 222 1810
Contact person: Kalina Bontcheva
E-mail: K.Bontcheva@dcs.shef.ac.uk

Universitaet des Saarlandes
Language Technology Lab
Campus
D-66041 Saarbrücken
Germany
Contact person: Thierry Declerck
E-mail: declerck@dfki.de

MODUL University Vienna GMBH
Am Kahlenberg 1
1190 Wien
Austria
Contact person: Arno Scharl
E-mail: scharl@modul.ac.at

Ontotext AD
Polygraphia Office Center fl.4,
47A Tsarigradsko Shosse,
Sofia 1504, Bulgaria
Contact person: Georgi Georgiev
E-mail: georgiev@ontotext.com

ATOS Spain SA
Calle de Albarracin 25
28037 Madrid
Spain
Contact person: Tomás Pariente Lobo
E-mail: tomas.parientelobo@atos.net

King's College London
Strand
WC2R 2LS London
United Kingdom
Contact person: Robert Stewart
E-mail: robert.stewart@kcl.ac.uk

iHub Ltd.
NGONG, Road Bishop Magua Building
4th floor
00200 Nairobi
Kenya
Contact person: Rob Baker
E-mail: robbaker@ushahidi.com

SwissInfo.ch
Giacomettistrasse 3
3000 Bern
Switzerland
Contact person: Peter Schibli
E-mail:Peter.Schibli@swissinfo.ch

The University ofWarwick
Kirby Corner Road
University House
CV4 8UW Coventry
United Kingdom
Contact person: Rob Procter
E-mail: Rob.Procter@warwick.ac.uk

Executive Summary

This document describes and delivers the software associated with the intermediate version of the PHEME integrated framework. The document focuses on reporting on the data and software integration aspects. The initial integration approach to PHEME was explained in deliverable D6.1.1, while the initial evaluation was presented in the deliverable D6.2.1.

From the data perspective, the document reports the components and repositories that allow PHEME to acquire, pre-process and store social media data. It also reports the results achieved so far with regard to the integration of the different components developed in several technical work packages.

From the software integration perspective, the focus of the second year of the project has been on the identification of the processes and components that take part in the whole rumour classification and verification in social media that PHEME is aiming to cover. One of the most challenging issues from the integration perspective is to enable different components to work with streaming data in real time. Therefore, the development of the software prototype has been conducted by making sure that the system and all its components are able to fulfil the project requirements. These requirements include processing social network data in a streaming fashion for real-time rumour classification and detection, dealing with cross-language and cross-media information, and providing timely results. Where needed, the integration approach should also allow easy and loosely coupled integration and communication for batch processing.

Contents

Index of Figures

1. Relevance to Pheme

1.1. Purpose of this document

This document is a software deliverable. The purpose of the document is to provide an overview of the architecture, data and software integration plans, as well as to report the intermediate prototype of the PHEME Integrated Framework. This document covers the following aspects:

- PHEME architecture and integration approach.
- Data Integration prototype.
- Software integration prototype.
- Next steps

It is worth mentioning that PHEME is a research project that aims to deliver a set of tools as integrated as possible to push the state of the art in rumour and veracity detection. It is not the aim of the project to deliver a commercial toolset. Therefore, PHEME follows a simple integration approach to deliver a coherent set of tools potentially integrated in multiple scenarios.

1.2. Relevance to project objectives

Data and software integration are key aspects of any project, and more in a data-intensive project such as PHEME. Projects that do not pay attention to devise pragmatic and clear integration plans are likely to fail to deliver sound common results, even if the research and the separate components and tools developed within the project are of very good quality. In consequence, project partners have been aware since the inception of the project of the integration procedures and plans, encouraging all to participate actively in the integration discussions.

Therefore, this deliverable aims to provide the baseline to the rest of the project to ensure that the data, components and tools provided by all partners can work nicely together in the different aspects they should interact.

1.3. Relation to other work packages

As stated in the previous paragraphs, this deliverable describes the integration glue for all the technical work done in the project. Therefore, it has relation to all of the technical and use case work packages. It also has connections to the exploitation aspects, as the PHEME integrated framework is one of the main exploitation outputs of the project and its sustainability will depend heavily on how well the integration is done.

1.4. Structure of the document

The document is organised as follows:

- Section 1 gives a brief introduction, outlines the major purpose of the document and explains its relevance to PHEME.
- Section 2 provides a general background of the PHEME architecture and the approach followed for integration.
- Section 3 reports the data and content integration prototype.
- Section 4 reports the software integration prototype.

- Section 5 concludes with consolidated findings so far and the next steps and references are listed in section 6.
- Annexes with extra information about specific components and repositories are given in section 7.

2. Pheme Veracity Framework Architecture and Integration Approach

2.1. Overview of the architecture

The main asset in PHEME project, the Veracity Framework, should be able to integrate the veracity intelligence tools and services developed in WP2, WP3, WP4, and WP5. The emphasis will be on storing and processing large volumes of historical data, coupled with real-time analysis of incoming new content. A view representing the functional blocks in relation to the repositories is given in Figure 1.

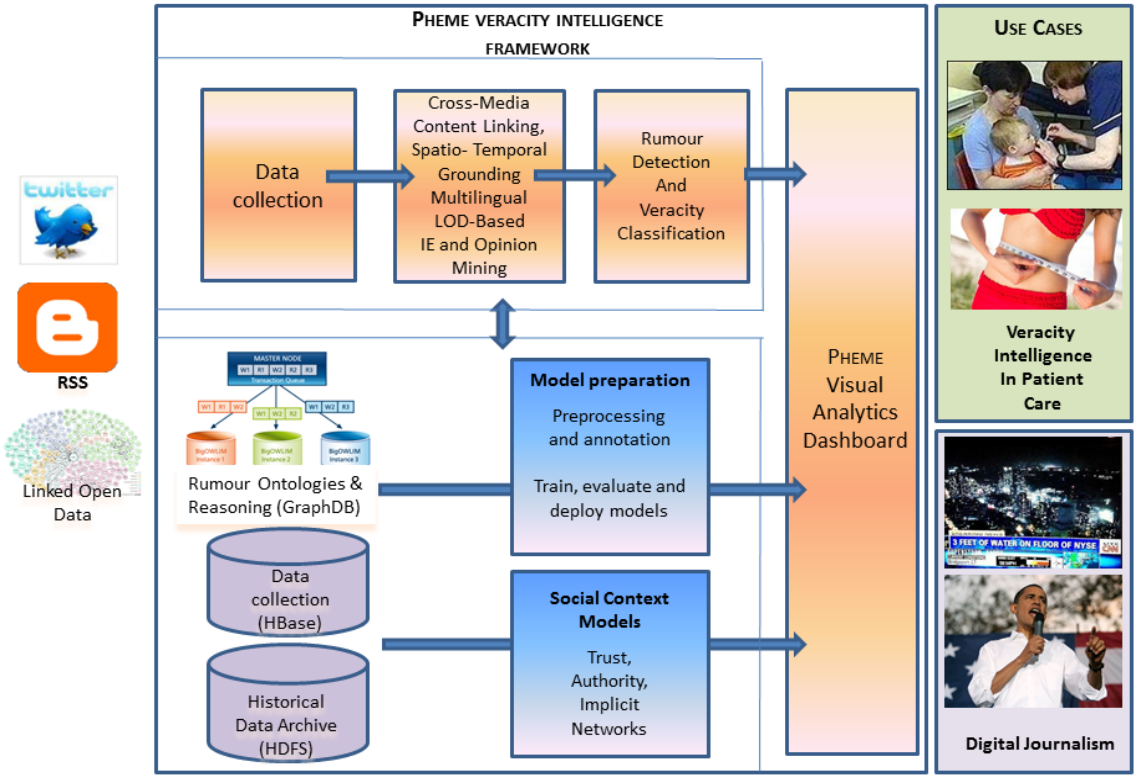


Figure 1. Pheme Veracity Framework Functional blocks perspective.

The figure above presents the main functional elements of the PHEME Veracity Framework:

- The data value chain for veracity checking: At the top of the figure, the different modules represent different steps in the collection, processing and analysis of data. It comprises components for data acquisition from Social Media resources, analytics (i.e. algorithms for cross-media and cross-language analysis, Information Extraction, etc.), and detection and classification of rumours. This is the core of the project and represents work done in most of the technical work packages of the project (WP2, WP3, WP4 and WP6). This core analytical layer is supported by stream and batch processing engines and service or pipelining frameworks,
- The model preparation: It comprises all the steps and processes aiming at annotate, train and evaluate the veracity models in PHEME. This layer is also core to the preparation of the data and models and it is mainly reflected in the technical work done in WP2.
- The usage layer: On the right hand side of the figure, it is the application provided to the end users. According to the different use cases defined in the project (WP7 and WP8), different applications can be built to provide adapted user interfaces according to the

specific requirements. The Application Layer should deal with the configuration of the user interface, and the management of the user interaction to dispatch the events towards the internal system (Service Layer). The PHEME Visualization Dashboard (WP5) will be also in charge of displaying different visualization perspectives to end users to understand how rumours form, spread and die, among other interesting visualization paradigms. A very generic approach will allow easy implementation of new applications adapted for specific business processes, and specifically tailored to the two use cases (WP7 and WP8).

- The persistence of the data: It is in charge of the mechanisms and models to store information. In PHEME, there are two kinds of repositories: The data collection content repository where the data from social networks is stored and pre-processed, and the storage of knowledge (ontologies, data annotations, etc.). For the data collection content repository, due to the data-intensive nature of the project, a NoSQL database has been selected as the main repository. As the formalism to represent the knowledge in PHEME is based on RDF, RDF repositories will have an important role in the architecture.

A high-level architectural diagram of the PHEME framework appears in Figure 2.

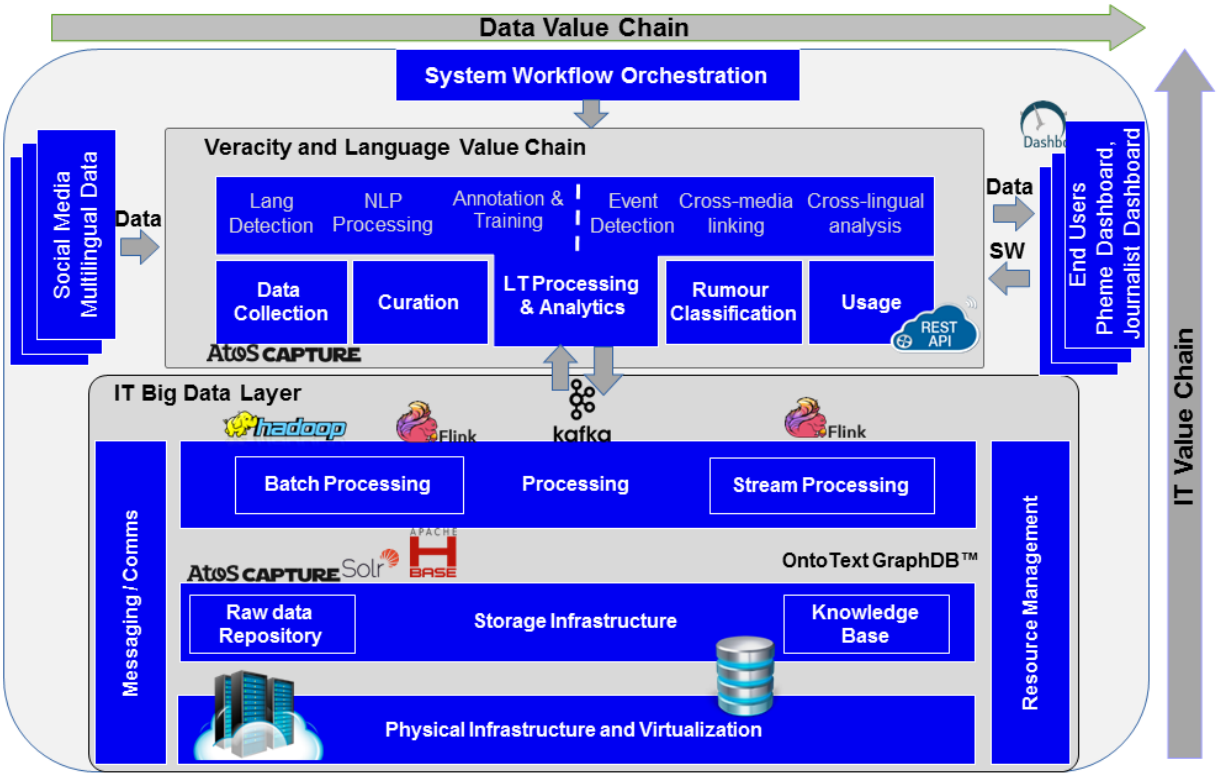


Figure 2. PHEME Veracity Framework Architecture.

Figure 2 shows the reference architecture for PHEME. This figure is following closely the blueprint provided for language technologies by the Coordination and Support Action MLi[1] in the deliverable D2.2. MLi proposes a Reference Architecture divided in two main axes:

- On the vertical axis "IT Value Chain" the value is created by providing more abstract access to the system functionalities, starting from low-level infrastructural services,

[1] http://mli-project.eu/

through LT/MT/NLP domain-specific services until the high-level workflows and LT marketplace services for support of the business services and information and process integration within the Business Entity.

- On the horizontal "Language Value Chain" the value is created by combining LT/MT/NLP services and components into more complex and higher level features that fulfil requirements of the end user.

At first glance, it is clear that PHEME follows closely the MLi Hub reference architecture. This has been done intentionally as common partners of the two projects were interested in collaborating in these architectural aspects. In more detail, the PHEME implementation shows the following:

IT Infrastructure

Pheme implements the IT Infrastructure of the MLi Hub following the same building blocks. The Physical Infrastructure and Virtualization layer is not specified in the diagram further, but it uses virtualised servers and several installations for different use cases. Therefore, specific infrastructure providers can deploy Pheme in multiple environments. So far, there are two separate Physical Infrastructure instantiations for PHEME:

- Main PHEME deployment infrastructure in Sheffield: This infrastructure is deployed on servers residing in Sheffield. It comprises three different servers, one acting as head node and two acting as working nodes. This infrastructure provides enough flexibility to be virtualised and hosts the main components and the underlying software infrastructure required for integration. This installation is used to showcase the project results.
- Secondary PHEME deployment infrastructure in KCL: KCL deployed a similar infrastructure at the end of Y2 to test and showcase the results of their case study. It is completely separated from the previous infrastructure.

One difference between the MLi Reference architecture and PHEME resides in the storage layer. In PHEME, the social media data and the semantic knowledge base are placed in the storage layer instead of the upper processing layer as proposed by MLi. This decision was taken on purpose to separate storage from the processing framework, although conceptually is very similar to the approach followed in the MLi Hub. PHEME proposes NoSQL databases (HBase) and indexed repositories (Solr) for the raw data, while using a semantic database (GraphDB) as Knowledge Base. PHEME needs to analyse streaming and batch data coming from social media, so it proposes the use of batch (Hadoop) and streaming processing engines (Apache Flink).

In both cases, the physical infrastructure is virtualised and secured, so that only authenticated users and services are able to interact with it. PHEME proposes therefore a scalable and distributed infrastructure where different components can be deployed and scaled.

Language Value Chain

The Language Value Chain (LVC) in PHEME proposes several steps from the collection of the data from social networks to its analysis and further usage. LT tools and services and sophisticated algorithms for event detection, cross-language and cross-media linking, along with training of veracity models for rumour detection and classification are also part of the LVC. The result of the analysis is used by the use cases (digital journalism and health) and visualised in dedicated dashboards.

2.2. Data Integration approach

Within WP6, task 6.2 is in charge of investigating how PHEME's diverse kinds of content and knowledge can be stored and accessed. From the perspective of the data usage within PHEME, the project differentiates between the social media data with very little pre-processing (Social Media content), and the more semantic and annotated content (Knowledge content). This section reports on the approach followed to integrate data for the project needs.

PHEME provides two main storage systems for social media and knowledge content in order to guarantee the data flow and data integration for the different services and components developed in PHEME.

The social media content data layer provides storage and access to the raw data acquired from social media. It is based on the Capture module developed in the scope of task 6.1 and it is based on state-of-the-art big data technology to guarantee the scalability of the solution. More details of this module can be found in section 7.1.

Although task 6.2 is scheduled for the end of the second year of the project, by nature data integration is an ongoing task. From the data integration perspective, the focus during the first year of the project was to provide access to both social media data and knowledge, and defining a set of initial access APIs to allow project partners to access and start with data integration experiments. The second year witnessed a progressive and iterative work on data to suit the needs of the different use cases, improving the data acquisition process and enabling the processes and pipelines to store data in the knowledge repository. This work will continue in year 3 with a progressive integration of data and improvement of APIs to access to it. This aggregated data is currently exposed in intermediate indexes, making use of tools such as Apache Solr[2] (e.g. for text indexing), APIs from Capture or GraphDB (mainly SPARQL queries).

The integration of the PHEME infrastructure is dataflow-driven and stream-oriented in order to deal with large amounts of data in a timely fashion. The shift from service-oriented architectures towards the data-flow driven (e.g. message-oriented) architectures allows many obstacles of traditional approaches to be overcome and to focus on relevant requirements. From the point of view of integration, the set of desired requirements towards the architecture are the following:

- High throughput
- Availability and resilience through infrastructure distribution and failover mechanisms (e.g. automatic replication, node failure handling)
- Shift from less efficient data polling (e.g. REQ/REP[3]) to more efficient data pull, with client-side subscription to relevant message channels
- Distributed deployment for consumer/producer or publisher/subscriber data processing paradigms.
- Message queues/logs for dealing with real-time, buffered data streams

In the next sections we present concrete technical solutions that should meet integration requirements.

[2] Apache Solr: http://lucene.apache.org/solr/

[3] Request-Reply, as

2.2.1. Social Media content

Social Media data: This raw data is the baseline for training the models for later rumour classification in real-time data. PHEME has access for research purposes to historical Twitter data. Based on PHEME use cases, a number of continuous data collection task has been established. As we speak, the PHEME infrastructure is collecting Twitter and content data from social networks. The real-time raw data (e.g. JSON representation of Tweets) is stored in two NoSQL databases. Based on types of queries for data retrieval and analytics, the data is stored in (i) Apache HBase[4] for batch processing, bulk loading and querying, and in (ii) Apache Solr for text indexing and advanced retrieval based on multiple data facets.

The Social Media content is stored in the Capture repository. Capture is explained in section 7.1.

2.2.2. Knowledge integration

In D2.2 "Linguistic Pre-processing Tools and Ontological Models of Rumours and Phemes" we described the PHEME ontology, that models veracity in social media. The ontology describes rumours, misinformation, disinformation and disputed claims, together with important network information such as authors (receivers and diffusers), events, and relations. Also, it easily integrates knowledge from LOD datasets, including DBPedia, Wikidata and GeoNames.

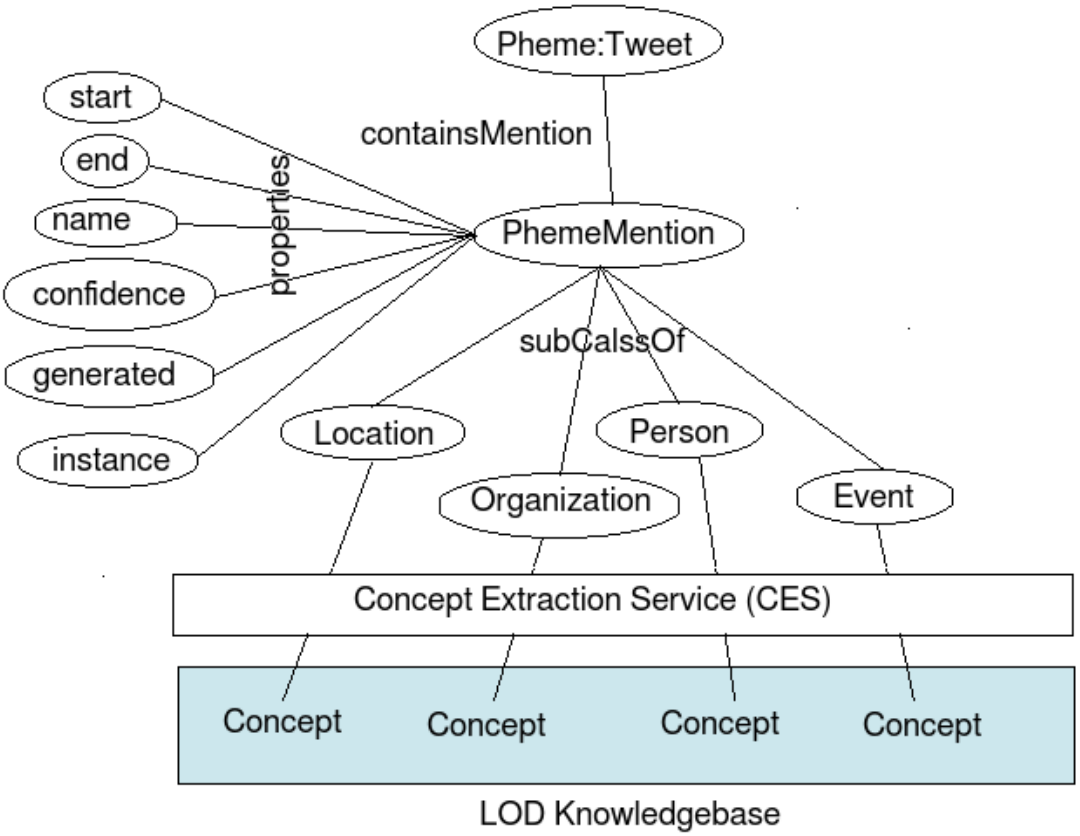


Figure 3 Integration of LOD data with PHEME ontology

Figure 3 shows the integration of LOD data with the PHEME ontology. The textual content of every tweet (or other information resource) represented in the Pheme ontology is enriched with LOD concepts, called PhemeMentions. They can be of type Location, Organization, Person,

[4] Apache HBase: http://hbase.apache.org/

Event, etc. Each mention has properties: start (the position where it starts in the text, in characters), end (the position where it ends), name (the name of the mention), confidence (a probability estimate of the confidence of the tag, given by the Concept Extraction model), generated (a boolean value that is false if the entity is found as a concept in the LOD knowledge repository and true, if it is only predicted by the model, but no corresponding concept was found in the LOD knowledgebase) and instance (the URI of the concept in the LOD knowledgebase, if not generated).

Below we give detailed description of the provenance and properties of the various types of concepts from the LOD Knowledgebase.

We group instances from LOD (DBPedia, Wikidata and GeoNames) based on their relations (sameAs) into clusters and to assign a limited set of properties to each cluster and facilitate automatic processing. These properties are:

- **URI** - the local name from dbpedia is used
- **type** - for each entity is selected exactly one explicit type from
- **type (LOD)** - all types from LOD ontologies are attached
- **preferredLabel** - selected amongst all labels we find. (Wikidata > Dbpedia > Generated from dbpURI)
- **alternativeLabel** - list of alternative labels received from wikidata and dbpedia redirect pages
- **shortDescription** - a very short description of the concept (from wikidata) (if any)
- **description** - more comprehensive description coming from dbpedia abstract property
- **picture** - image representation of the concept (if any)
- **thumbnail** - thumbnail of the picture

Table 1 Generic properties from Wikidata

Property name	**Property code**	**Description**
part of	P361	This item is a part of that item; expresses a whole-part relationship
member of	P463	part of a specific organization or club. Do not use for membership in ethnic or social groups, nor for holding a position such as a member of parliament
has part	P527	object that is part of this subject
start time	P580	Indicates the time an item begins to exist or a statement starts being valid, usually used as a qualifie
end time	P582	Indicates the time an item ceases to exist or a statement stops being valid
point in time	P585	Time and date something took place, existed or a statement was true

replaces	P1365	Person, state or other item who was replaced in an office, position, etc
genre	P136	genre of a creative work or genre in which an artist works
replaced by	P1366	Person or state or item who replaces another in an office, position, etc.
coordinate location	P625	geo coordinates of the subject
event time	P585	Time and date something took place, existed or a statement was true
located in	P131	the item is located on the territory of the following administrative entity
head of government	P6	head of the executive power of a town, city, municipality, state, country, or other governmental body
operating system	P306	Operating system (OS) on which a software works; OS installed on hardware
programming language	P277	The programming language(s) in which the software is developed
author	P50	main creator(s) of a written work
of	P642	qualifiers stating that a statement applies within the scope of a particular item

Wikidata properties for PERSON: *gender* (Sexual identity of the subject), *date of birth* (Date on which the subject was born), *date of death* (Date on which the subject died), *place of birth* (The most specific known (e.g. city instead of country, or hospital instead of city)), *place of death* (The most specific known (e.g. city instead of country, or hospital instead of city)), *country of citizenship* (The object is a country that recognizes the subject as its citizen), *occupation, alma mater, field of work, position held, religion, mother, father, brother, sister, spouse, employer, official website, position at team, club, notable work, instrument, member of political party, noble family*.

Wikidata properties for ORGANIZATION: *coat of arms image, inception, logo image, headquarters location, industry, production, founder, CEO, chairperson, website, affiliation, subsidiaries, parent company.*

Wikidata properties for LOCATION: *continent, country, population, capital, official language, head of state, capital of.*

Wikidata properties for EVENT:*organizer, participant, public holiday, location.*

Wikidata properties for WORK: *genre, contributor, creator, author, publisher, publication.*

A few relevant statistics on the number of concepts by type (as of March 2015):

Table 2 Concept statistics by type

Concept type	Concept sub-type	Count
PERSON:		758924
	Athlete:	270288
	Artist:	105011
	Politician:	41160
	Monarch:	3018
	Journalist:	2970
	Scientist:	18437
	Cleric:	12407
	Economist:	789
ORGANIZATION:		103468
	Company:	95480
	Band:	42780
	EducationalInstitution:	58319
	SportsTeam:	44137
	Broadcaster:	27782
	MilitaryUnit:	16411
	PoliticalParty:	14804
	Non-ProfitOrganisation:	10732
	SportsLeague:	5342
	TradeUnion:	2109
LOCATION:		156593
	PopulatedPlace:	557586
	Building:	140963
	BodyOfWater:	57227
	SportFacility:	8615
	Infrastructure:	70743
	Mountain:	22108
WORK:		572314
	MeanOfTransportation:	46993
	Software:	42903
EVENT:		192736

THING	(any other concept that is not one of the above):	841988

We used the Concept Extraction methodology developed by Ontotext for identifying and tagging concepts with the types described above. On a news dataset, the methodology achieves state-of-the-art performance F1 (for English) 92%. The Concept Extraction methodology is developed for English, German and Bulgarian.

It is expected that the performance on the tweets is poorer than the results on the news dataset, mostly because the training dataset consists of news, therefore the application domain is substantially different, but also because Named Entity Recognition tasks are notoriously difficult on tweets, which are short texts lacking informative context. By visual inspection, the results have acceptable Precision and low Recall, so we adjusted the Concept Extractor's thresholds for confidence in order to output more entities. A precise performance evaluation of Named Entity Tagging on the tweets is planned to be carried out in the third year. Since we expect some differences, we will evaluate separately the performance on the two use cases: journalism and healthcare.

2.3. Software Integration approach

PHEME is a research project that aims to deliver a set of tools as integrated as possible to push the state of the art in rumour and veracity detection. It is not the aim of the project to deliver a near-to-market toolset. However, in the light of what has been discussed so far, PHEME will follow an approach to integration trying to deliver a coherent set of tools that could be easily tailored and integrated in multiple scenarios. This means that the integration approach should be flexible and simple.

Many of the PHEME components already covered in this document need to be pipelined in order to accomplish certain tasks, such as language detection, text pre-processing, processing in several languages, etc. These components are heterogeneous, developed by different partners often using different programming languages (mainy Java and Python) and sometimes even hosted remotely. These facts pose requirements to the integration approach followed in the project.

From the integration perspective, the main goal is to ensure that the whole system and all its components are able to fulfil the project requirements of processing social network data in a streaming fashion for real-time rumour classification and detection, cross-language and cross-media and providing timely results. Some of the components will perform other tasks by themselves, such as training the Machine Learning systems. In these cases, integration with other components is not required. Where needed, the integration approach should also allow easy and loosely coupled integration and communication for batch processing.

The PHEME Integrated Framework is meant to scale. In order to ensure the scalability of the solution the project is on the one hand improving the performance and scalability of the different individual components as reported in previous sections. On the other hand, from the integration perspective, the project is following a global integration strategy to ensure the performance and scalability of the overall system. This global integration strategy presents project-wide approaches orthogonal to the individual scaling plans and common for most technical components. The focus is on integration aspects to provide an infrastructure to integrate components while enabling big data scaling techniques, such as scaling up, scaling out, parallelisation, etc. This global integration strategy takes into account limits of individual components to align them into a common plan.

This real-time processing integration follows the concept of pipelines. Pipelines allow the addition of multiple components in a process. From the integration and scalability perspectives, pipelines should be able to increase the throughput and decrease the latency as much as possible. In order to do that, enabling parallelisation means processing of several inputs coming from components in a pipeline with other identical components that work in parallel. In an optimal scenario, it is simply adding more processing units for the same components that work slower compared to other components in a pipeline. More processing units can be provided to components by scaling horizontally or vertically. Scaling horizontally is achieved by adding more nodes (computers) to a system, while vertical scaling can be achieved by running the whole pipeline on a faster machine.

For programming language independence, messaging systems support multiple platforms and programming languages, which represent a clear solution to the integration problem. There are many popular messaging systems, such as ZeroMQ[5], RabbitMQ[6], Apache Flume[7] and Apache Kafka[8] among others. However, as explained in D6.1.1, the Capture module provides the infrastructure for messaging and several other integration points that PHEME may take advantage of. Capture is built on top of a big data-enabled infrastructure that provides batch and streaming processing engines. In particular Capture uses Apache Kafka pub-sub mechanism for message exchange. This ability has been exploited during the second year of the project as baseline for data and component integration for the veracity framework.

2.3.1. Integration using Apache Kafka

Apache Kafka is a high-throughput publish/subscribe messaging system that provides very handy features for real-time processing systems and it is frequently used to enable data pipelining. It is therefore an adequate approach to be used in the integration of the PHEME real-time components. Kafka is used in PHEME as the main integration middleware.

Kafka is designed to allow a single cluster to serve as the central data backbone for a large organisation. It can be elastically and transparently expanded without downtime. Data streams are partitioned and spread over a cluster of machines to allow data streams larger than the capability of any single machine and to allow clusters of coordinated consumers.

Apache Kafka differs from traditional messaging systems in that:

- It is designed as a very easy to scale out distributed system.
- It offers high throughput for both publishing and subscribing operations.
- It supports multi-subscribers and automatically balances the consumers during failure.
- It persists messages on disk and thus can be used for batched consumption such as ETL, in addition to real time applications.

Kafka is a general purpose publish-subscribe model messaging system, which offers strong durability, scalability and fault-tolerance support. For the pipelining approach of integration of PHEME components, Apache Kafka allows to pass streams (e.g. Twitter streams) from one

[5] http://zeromq.org/

[6] https://www.rabbitmq.com/

[7] https://flume.apache.org/

[8] http://kafka.apache.org/

component to the next. PHEME components and algorithms can therefore publish and subscribe to data streams to decouple the different processes, thus creating pipeline of loosely-coupled components. Figure 4 shows the publish-subscribe mechanism in Kafka.

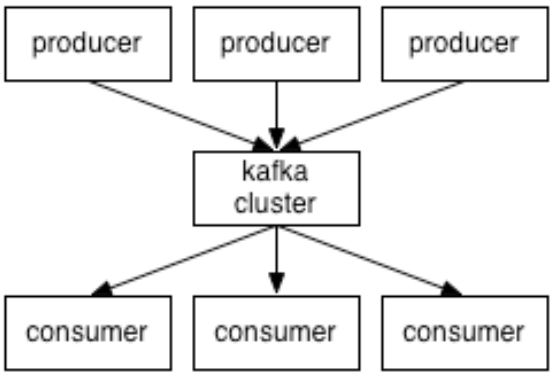


Figure 4. The publish-subscribe paradigm in Kafka

The Capture module provides an Apache Kafka installation that allows the subscription from PHEME components to the social networks data streams collected in real time by Capture. As an example, implementing a new Kafka Consumer for the streaming incoming flow is just a matter of creating a new Kafka Topic and subscribing to it, as shown in Figure 5 in an example taken from Capture.

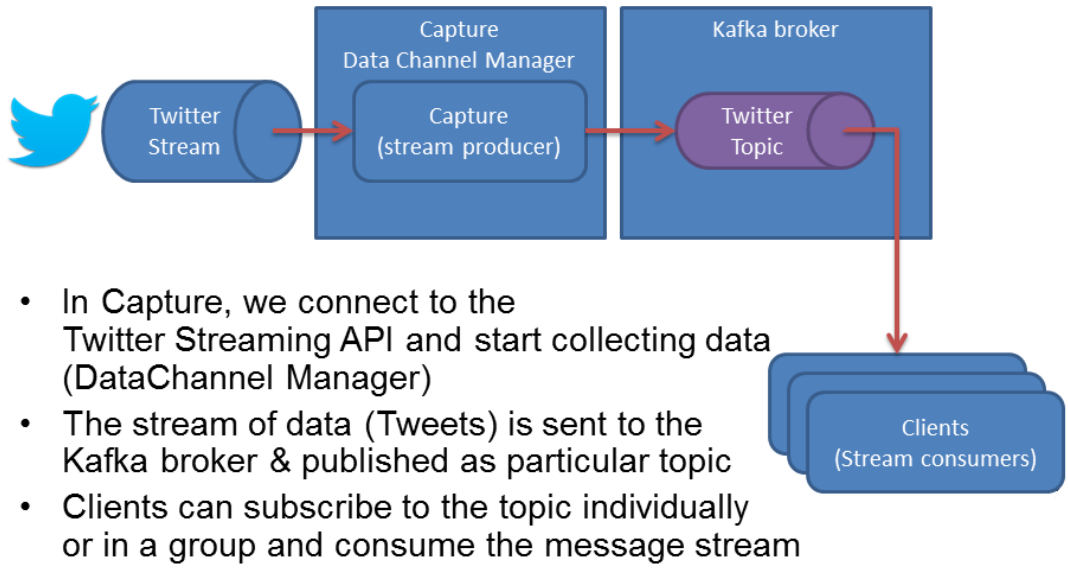


Figure 5. An example of a Kafka consumer reading from Twitter

2.3.2. Kafka message format guidelines

PHEME partners are following a common approach for the format of the Kafka topics. During the second year of the project, we focused on the integration of components based on Twitter data. The idea is to follow a pipeline approach enriching the original Twitter data with extra annotations provided by the different components of the pipeline, thus enabling an incremental addition of annotations of the original Kafka message. The first component in the pipeline (Capture) publishes stream of tweets in a Kafka topic and the next component add new annotations to those tweets (i.e. language of the tweet) and publish them in a new Kafka topic.

This incremental approach allows for a very easy integration, as components just need to publish a new stream with the addition of the new annotation made. New components just need to subscribe to those queues to consume the streams.

This approach has been followed in the design of rumour classification and event detection pipelines (see section 4.2.). In both pipelines, the first component (Capture) sends raw messages to the first Kafka topic. The message is passing through various components, but after every stage of processing it ends up in a new Kafka topic, and ready to be consumed by the next component in the pipeline.

The guidelines for this particular integration pipeline approach involving processing of tweets and the addition of annotations are the following:

- Messages are passed through Kafka topics in JSON format, encoded as simple text strings.
- At the beginning of the pipeline, each JSON object has a set of common baseline properties:
    - As Twitter, our main source, and other sources all follow the convention of placing the document text in a top-level key called "text".
    - The social media message identified should be a top-level "id_str" object, following Twitter convention.
    - To distinguish between different social media document types (e.g. twitter, reddit, etc.) each message has "pheme_source" property that denotes one of supported type. At this moment those are: “twitter”, “rss”, “reddit”. Other types will be included later.
    - In case of Twitter, the raw twitter object is contained in “raw_json” property.
- Changes to the JSON object message are incremental, that is, each component may add new annotations (in a form of JSON properties), but should not remove previous information if it is not necessary. In other words, components re-publish the entire original JSON object and add additional keys.
- New annotations created by the pipeline components are added as top-level properties to the existing JSON object. At this point, the following properties have been declared:
    - Event extraction: "event_cluster", long unsigned int
    - Language: "langid", an array consisting of two elements: [string, float], where string is a two-letter lowercase country code and float the confidence in [0..1]. This is not to be confused with the “lang” field based on the Twitter language identification that comes from the Twitter API and is contained in messages.
    - Tokens: "tokens", an array of tokens in a form of array: [int, int], giving token start/end offsets.
    - Named entities, spatio-temporal entities: "pheme_entities", an array of entities in a form of array: iint, int, string, corresponding to string start offset, end offset, and entity type; e.g. "person", "timex".
    - Support/Deny/Query: "pheme_sdq", array of [string, float] where string is support|query|deny and the float the confidence in [0..1].
    - Spatial grounding: "pheme_location", an object of {key: value}, where the source key is "latlong", "dbpedia", "nuts" or "geonames", and the value is either

a string reference or an array [float, float] for latitude and longitude. All keys are optional.

- If there is a chance of colliding with a social media source's top level key, or a key name is non-descriptive, prefix it with "pheme_"

An example document from the first stage of the pipeline:

{
"**text**": "Obama is doing well",
"**pheme_source**": "twitter",
"**id_str**": "921349812834098012313256586018634",
"**raw_json**": { *raw JSON object, as received from the Twitter API* }
}

After passing through the pipeline components, the message might look like this:

{
"**text**": "Obama is doing well",
"**pheme_source**": "twitter",
"**id_str**": "921349812834098012313256586018634",
"**event_cluster**": 8721364871239,
"**langid**": ["en", 0.879],
"**tokens**": [
[0,5], [6,8],
[9,14],
[15,19]],
"**pheme_entities**": [
[6,8, "person"],
[9, 14, ”event”]],
"**pheme_sdq**": ["support", 0.566],
"**pheme_location**": {
"dbpedia”: "United_States",
"latlong": [35.5, 98.0] },
"**raw_json**": { *raw JSON object, as received from the Twitter API* }
}

Note that other social media document types (coming from social media sources other than Twitter) will follow similar guidelines. In this sense, the core fields remain the same (such as “text”, “pheme_source”, “id_str”), while other specific fields can be added based on particular data type needs.

2.3.3. Other possible integration mechanisms

Integration using Apache Flink

There are several open frameworks and tools allowing real-time computation, such as Apache Storm, Yahoo! S4, Spark and Apache Flink, among others. In a preliminary version, Capture decided to make use of Storm due to its easy integration with other Apache tools (i.e. Kafka or Hadoop) and excellent scalability, tolerance to failures and integration capabilities. However, during 2015 Apache Flink has been gaining traction as one of the main purely stream processing engine with excellent batch processing performance. Therefore, the Capture installation offers currently Flink libraries for data processing.

Integration using REST APIs

It is also possible to use directly APIs from different modules to achieve a more tight integration between components. Examples of these are:

- Capture REST API in order to execute basic and/or faceted queries over the collected data. Components from PHEME will be able to fetch data using the available services.
- Dashboard Document API: API to insert documents in the PHEME Dashboard as discussed in section 7.10.

2.3.4. Frameworks available for integration

As hinted in section 2, the Capture module for data collection provides a set of big data frameworks useful for data and component integration. PHEME will take advantage of these integration features as baseline for data and component integration for the veracity framework.

Figure 6 shows a detailed view of the Capture framework stressing the integration tools:

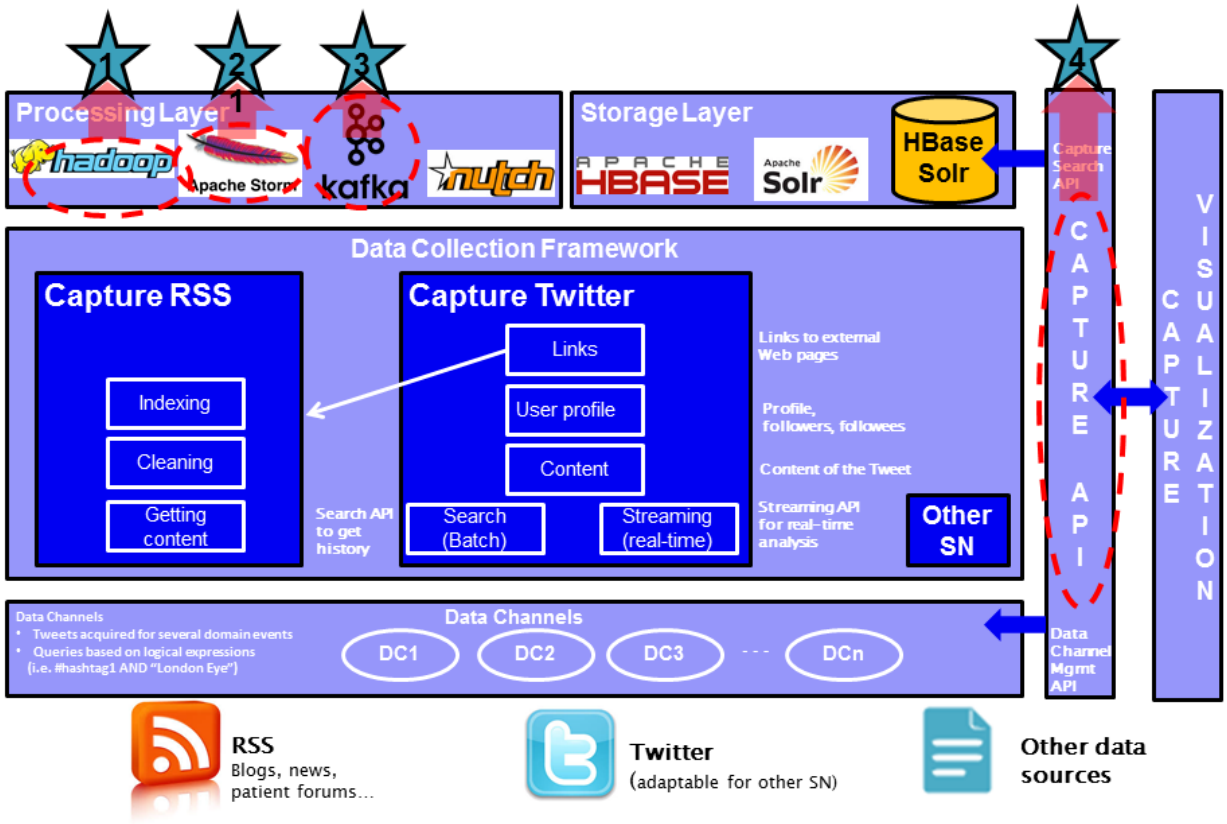


Figure 6. Capture integration points.

Figure 6 shows that Capture is built on top of three Open Source frameworks from the Apache Foundation that provides processing capabilities, namely (1) Apache Hadoop[9], (2) Apache Storm[10] and (3) Apache Kafka[11]. Capture also provides the (4) Capture API allowing an extra integration point.

9 Apache Hadoop: http://hadoop.apache.org/

10 Apache Storm: https://storm.apache.org/

11 Apache Kfka: http://Kafka.apache.org/

3. Content and Knowledge Integration Prototype

3.1. Social media content repository

To support the social media data storage in PHEME, WP6 provides the data collection framework named **Capture** developed by Atos. Capture data is stored in NoSQL databases (Apache HBase and Apache Solr), that provide necessary scalability and data distribution over multiple nodes in order to handle large volumes of data.

The syntactic repository is part of the Capture module for data collection, further explained in section 7.1. Figure 7 shows the high-level architecture of Capture.

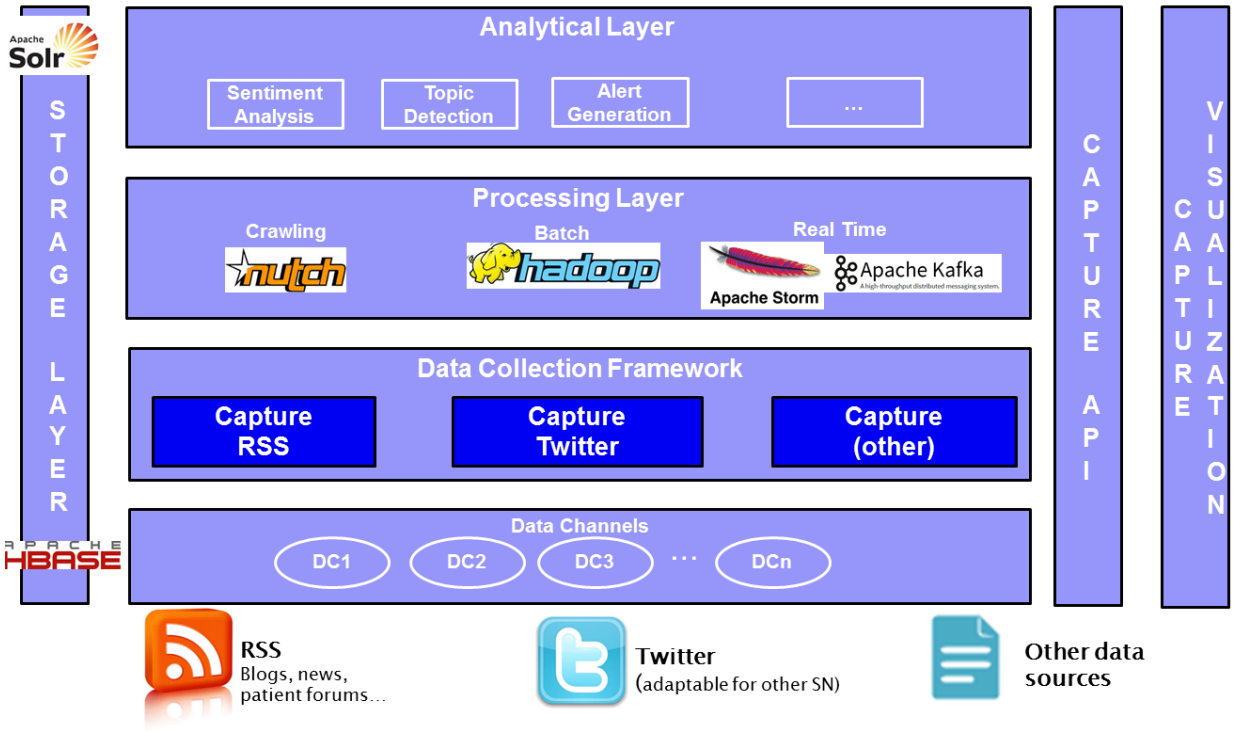


Figure 7. Data collection framework (Capture) high-level overview

From the repository perspective, the main element from the Capture architecture is the storage layer. Capture is using Apache HBase as storage repository combined with a powerful indexing mechanism based on Apache Solr. This way, fast information retrieval for the collected data (i.e. tweets) is ensured. The interaction with the repository is done using a RESTful service layer (the Capture API). This API layer provides methods for the management of the data channels and search,

It is also worth mentioning that Capture enables pipelining different analytical components both batch or real time, giving an extra integration flavour to the data collection tool. Capture provides the possibility of interacting with real time streams of tweets. To that extent, the use by other components developed within the project of the real time processing framework provided by Capture is possible without having to interact directly with the repository. More detailed explanation of the Capture API and how to use it for data integration is provided in section 7.1.

During the first year of the project, the objective was to carry out the data gathering from Twitter. To that extent, Atos provided an initial version of the Capture framework, developed in the scope of commercial proof of concepts with Atos customers. Capture had an initial data model enabling the acquisition of tweets. However, the acquisition tweets was not deemed

enough for the requisites of the PHEME use cases for rumour and veracity detection. Therefore, the Capture model was extended to be able to cope with different data sources (Twitter, RSS and in the future any other social network). In particular, the Twitter model in Capture was also extended to be able to not only retrieve the pure JSON representation of a tweet, but also to store and index more raw info, such as getting user profiles, list of followers and followees, etc. The Capture model for Twitter data is now characterized by the followings attributes:

- TweetID: The representation of the unique identifier for the Tweet.
- CreatedAt: UTC time when this Tweet was created.
- FavouriteCount: Indicates approximately how many times this Tweet has been "favorited" by Twitter users.
- HashTags: Tweet hash tags.
- InReplyToId: If the represented Tweet is a reply, this field will contain the integer representation of the original Tweet's ID. Needed for the conversation chains.
- Latitude: The latitude of the Tweet's location.
- Longitude: The longitude of the Tweet's location.
- OriginalTweetId: Original identifier of the Tweet.
- Place: It is a specific, named location with corresponding geo coordinates.
- RetweetCount: Number of times this Tweet has been retweeted.
- Source: Utility used to post the Tweet, as an HTML-formatted string. Tweets from the Twitter website have a source value of web.
- Text: Tweet text
- UserDescription: The user-defined text describing their account.
- UserFollowers: The number of followers this account currently has. Under certain conditions of duress, this field will temporarily indicate "0."
- UserFollowees: The number of users this account is following. Under certain conditions of duress, this field will temporarily indicate "0.
- UserID: The representation of the unique identifier for the User.
- UserName: User name.
- UserScreenName: The screen name of the user.

Besides Twitter data, PHEME use case partners have a set of requirements with which the social data repository should deal with:

- Collection of data from RSS feeds: Both case studies expressed the need of acquiring data from news and blogs for different purposes. A standard way of acquiring data is getting the content based on querying RSS feeds provided by many sites. In order to create a common acquisition framework, Capture will integrate this RSS feed content gathering using a similar approach. The approach would be to generate a new data source model of type RSS, similar to the case of Twitter mentioned above. This new type of data source will give the possibility of expressing the RSS feeds (the actual URL of the feeds) and filters (keywords or similar) to be acquired. In this way, for instance, Capture will be able to acquire news from the pages retrieved from the feeds

that contains a set of keywords, not all the pages. This is still work in progress at the time of writing this deliverable.

- Specific connectors to some forums: As a requirement from the eHealth case study, some patient forums that are not open or do not provide RSS feeds would need ad-hoc scraping in order to get content. This might require also agreements with the forum's owners and it is still unclear to what extend WP6 will provide support for this type of functionality.

Speaking in terms of enhancements or PHEME-related improvements to the Capture framework, it is planned to work in the following aspects in the coming months:

- Define a future strategy for snapshots of user profiles. Capture allows the possibility of getting and storing Twitter user profiles. However, the question of automating what profiles should be extracted is still unclear and will depend on the needs of the analysis to be done in the scope of the PHEME veracity framework. Therefore, the concrete strategy to get Twitter user profiles is not set yet. Currently, it is a manual procedure (a service getting one or several profiles in a service invocation). A tentative strategy could consist in getting the profiles on demand.
- Show the retweets history.
- Links/Sources download and indexing: Getting the links (URL mentioned in a specific tweet) could be of interest to get more knowledge related to specific rumours. However, retrieving all the links in all tweets could become storage and computationally expensive. Therefore, a strategy to manually select and get the content for some key tweets will be put into place within Capture.
- Try to maximise the amount of tweets gathered by making an intelligent use of the Twitter APIs. Capture is currently managing the limits of queries posed by the different open Twitter APIs. However, there is still room for improvement.

Section 7.1.4. reports the main methods of the Capture API.

3.2. Knowledge repository

From a technical perspective, the Knowledge repository in PHEME project is covered by GraphDB. A detailed description of GraphDB was given in T4.1. (Deliverable D4.1.1, "LOD-based Reasoning about Rumours: Initial Prototype").

Figure 3 illustrates the integration of GraphDB with Kafka and Concept Suggester. In section 7, we describe each component related to GraphDB with the corresponding input and output formats.

The PHEME ontology was developed under T2.2, whose aims were to build ontological models of veracity, misinformation, social and information diffusion networks, rumours, disputed claims, temporal validity of statements, and user online behaviour. The PHEME ontology can be considered as a constellation of a main ontology that models the veracity parameters, and related ontologies that model user behaviour, temporal and special parameters, etc.

The ontology development cycle and results are described in D2.2. The main ontology reflects the rumour annotation scheme, developed by UWAR and presented in D2.1. This ontology has been created in several cycles that closely follow the refinements of the annotation scheme. The basis for the main PHEME ontology is the PROTON ontology.

There were developed datasets, annotated by the proposed annotation scheme and the related ontology model, through a web application developed by the partner UWAR.

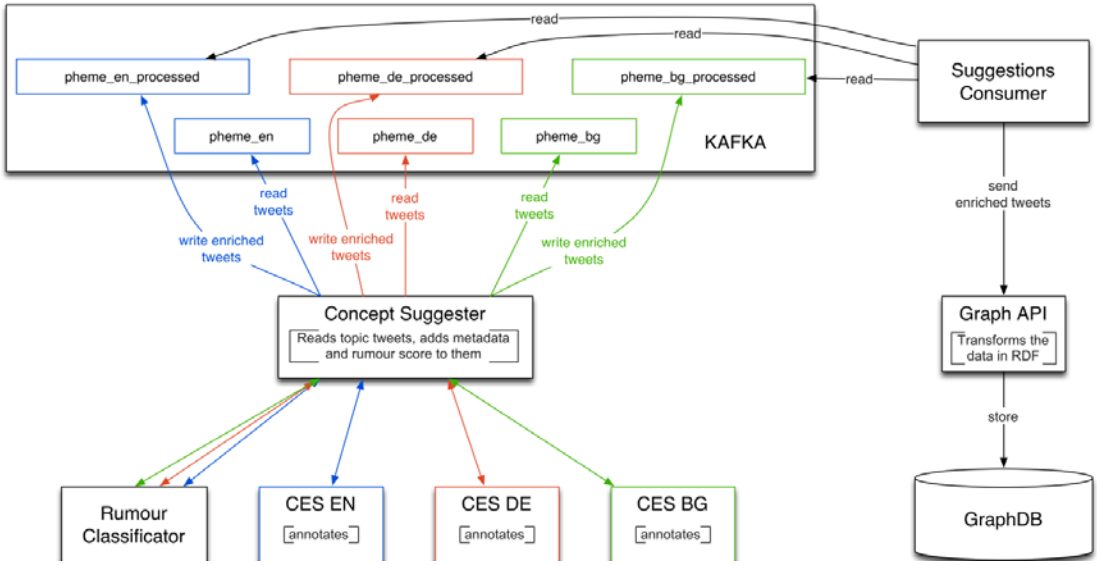


Figure 8. Kafka- GraphDB integration, serving concept extraction and rumour classification

Below there are two instances as examples of a source tweet and a replying tweet with respect to the PHEME ontology in the repository:

pheme:twt498254340310966273

a pheme:SourceTweet ;

pext:contentInformationResource "Michael Brown is the 17 yr old boy who was shot 10x & killed by police in #Ferguson today. Media reports \"police shoot man\". #blackboysonly"@en ;

ptop:hasDate "Sat Aug 09 23:47:46 +0000 2014" .

This instance is the source tweet. Its content is in the property: contentInformationResource. Also, the property hasDate has been specified.

pheme:twt498260814487642112

a pheme:ReplyingTweet ;

pext:contentInformationResource "@AmeenaGK @jaythenerdkid And how long before a conservative pundit finds some pic of him messing around with friends to ID him as a \"man\""@en ;

ptop:hasDate "Sun Aug 10 00:13:30 +0000 2014" ;

ptop:derivedFromSource pheme:twt498254340310966273 .

This instance is a replying tweet. Additionally to the above properties, another property is specified: derivedFromSource.

4. Software Integration Prototype

This section reports on the current version of the integrated prototype released in year 2. It also gives an overview of the main processes, components and data delivered, as shown in Figure 9.

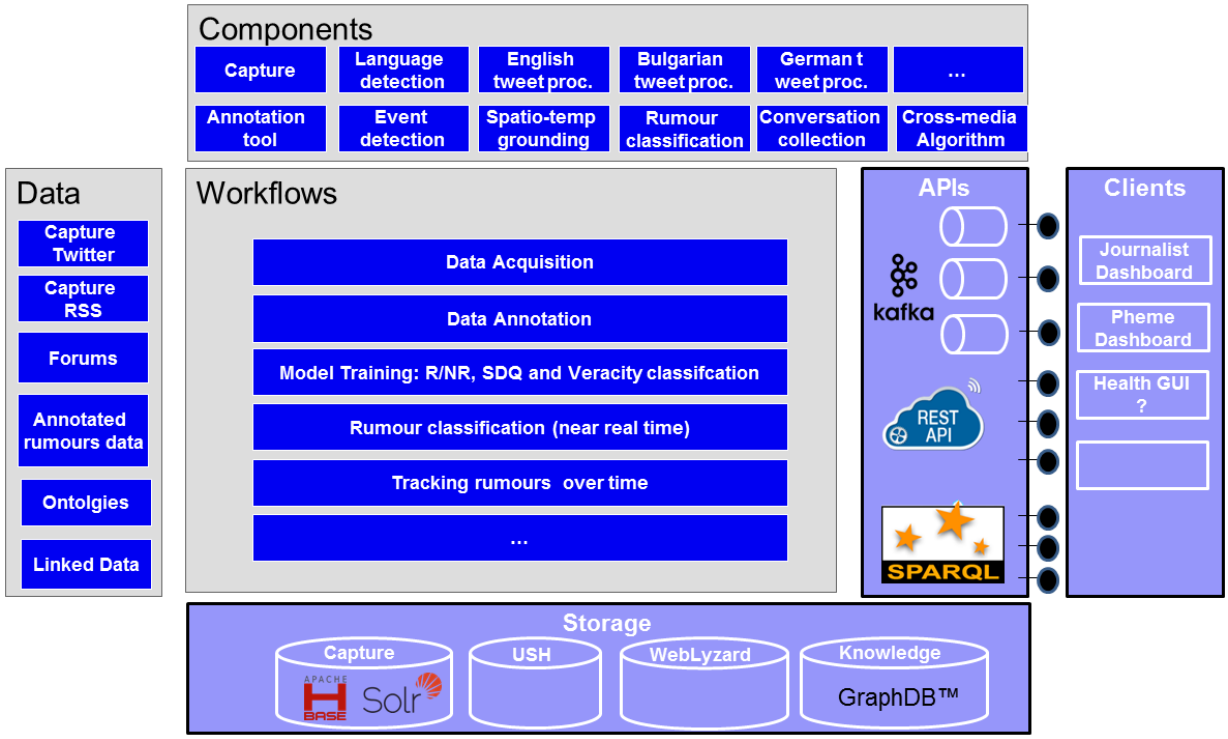


Figure 9. Overview of the main processes, components and data

4.1. Overview of the components available for integration so far

Table 3 shows the components that are under development in PHEME so far. It is important to note that not all components listed are at the same level of maturity at the time of actual integration. Therefore, Table 3 describes in brief the status of each component, its availability and pointers to this document or specific deliverables where more information can be found.

Table 3. PHEME components

Component	Responsible Partner	Availability	Brief Description	Extended Description
Capture	ATOS	Available	Data collection tool developed in WP6	Section 7.1.
Language Detection	USFD	Available	Algorithm to classify incoming messages according to their language (focus is on EN, DE, BG)	Section 7.7.
English Tweets processing components	USFD, ONTO	Partially available	CES (Concept Extraction Service) for English is applied to Tweets for identifying named entities and linking them to LOD.	Deliverable 4.1.1 and current deliv.
Bulgarian Tweets processing components	ONTO	Partially available	CESBulgarian	Deliverable 4.1.1 and current deliv.

Component	Responsible Partner	Availability	Brief Description	Extended Description
German Tweets processing components	USAAR	Partially available	CES German	Deliverable 4.1.1 and current deliv.
Sub-story Detection	USFD	Available	Sub-story detection aims to detect sub-stories circulating in a stream of tweets, around a newsworthy event (e.g. Ferguson unrest). It adds annotation to the tweets indicating the ID of the cluster of events the tweet is assigned to.	Deliverable 3.3.1 and section 7.5.
Spatio-temporal grounding	USFD	Partially available	Annotates spatio-temporal expressions in the text.	Section 7.6.
Cross-Media and Cross-Language linking	USAAR	available	Linking to press-media from Twitter text. Extraction of an extended set of key words/terms	Deliverable 3.1 Section 7.9.
Rumour Classification	ONTO	Partially available	Topic-agnostic, language-agnostic model, based on features like user metadata (friends, followers, activity), evidence cited in the forms of URL domains, text style such as punctuation, length, capitalization, replies, retweets, velocity of replies.	Deliverable D4.3.1 Section 7.12.
Conversation Collection	UWAR	Available (Twitter)	The conversation collection module developed by UWAR is a web service developed as a web interface or API that allows the user to collect (and visualise) the thread of tweets replying to a specific tweet. The user needs to specify a single source tweet, providing its URL or tweet ID, and the tool collects all the replies to that tweet. The tool allows the user to retrieve two different types of outputs: (I) JSON-formatted output including all the tweets in the conversation, which is useful for the integration with the other modules, and (ii) a web-based output that renders the replies in an HTML file in a forum-like visualisation of the thread, which is convenient for end users. NOTE: Since retrieval of replying tweets has been discontinued in Twitter API v1.1 (it used to be available through the	Deliverable 2.4 Section 7.3.

Component	Responsible Partner	Availability	Brief Description	Extended Description
			'related_results/show' endpoint in v1.0), this tool scrapes the replies from the HTML of the source tweet.	
Data annotation & analysis module	UWAR	Available (Twitter)	The data annotation tool is a web application that can be used to visualise streams of Twitter data, and to annotate the tweets as being rumours or not.	Deliverable 2.4
PHEME Dashboard	MOD	Available	PHEME Dashboard and API developed in WP5	Deliverable 5.2.1
GraphDB	ONTO	Available	Knowledge Repository	Deliverable D4.1.1 Section 7.2. .

4.2. Overview of the main processes and pipelines

This section provides a very high-level overview of some of the main processes envisaged for the project, with special attention to the integration between different components and the data repositories.

The main processes discussed so far in the project are shown in Table 4:

Table 4. PHEME main processes

Process / Pipeline	Availability	Responsible Partner	Brief Description
Data Collection	Partially Available	ATOS	Based mainly in Capture. Available for Twitter and RSS feeds. In progress for Reddit Under study for other social media
Data Collection and Annotation	Partially Available	ATOS UWAR	Continuation of the Data Collection process, it provides data for annotation. Available for Twitter, Reddit and RSS feeds. Under study for other social media
Rumour Classification (near-real time)	Partially available	ATOS	This pipeline process tweets in real-time with the aim of detecting events and candidate rumours. This is the main pipeline where we focused the integration efforts in year 2, as it involves integration of functionality from many components and algorithms developed by partners. More info can be found in section 4.2.2.
Data Annotation	Available	UWAR	Annotation tool available for identification of

Process / Pipeline	Availability	Responsible Partner	Brief Description
			rumours and non-rumours Crowdsourcing methodology established for the annotation of tweets within conversations, using the annotation scheme developed at Pheme More info can be found in Deliverable 2.4
Rumour / Non-Rumour Classification model	Partially available	ONTO	This process classifies tweets as rumour or non-rumour, by computing the rumour probability according to a tree classifier trained on the Journalism data. More technical details on the classifier can be found in Deliverable 4.3.1
Support / Denying / Question Classification model	Partially available	USFD	This is a process that involves the training of a model for SDQ classification Under construction. More info in the subsequent deliverables.
Veracity Classification model	Not available	USFD	This is a process that involves the training of a model for veracity classification Under construction. More info in the subsequent deliverables.
Tracking rumours over time	Not available	ATOS	This process is in charge of monitoring and analysing rumours over time. It is in essence very similar to the Rumour classification process, but it is batch in nature. This process may end-up divided in several sub-processes. To be done in year 3

It is worth noticing that other processes may arise in the final year or the project. Only processes that need a certain degree of integration are listed here, meaning that processes that involve several components but are tightly integrated have not been considered.

4.2.1. Data collection and annotation workflow

The first of the workflows is related to the gathering of Twitter data and its annotation for further usage in training models. Figure 10 shows the main components and iterations involved in this workflow:

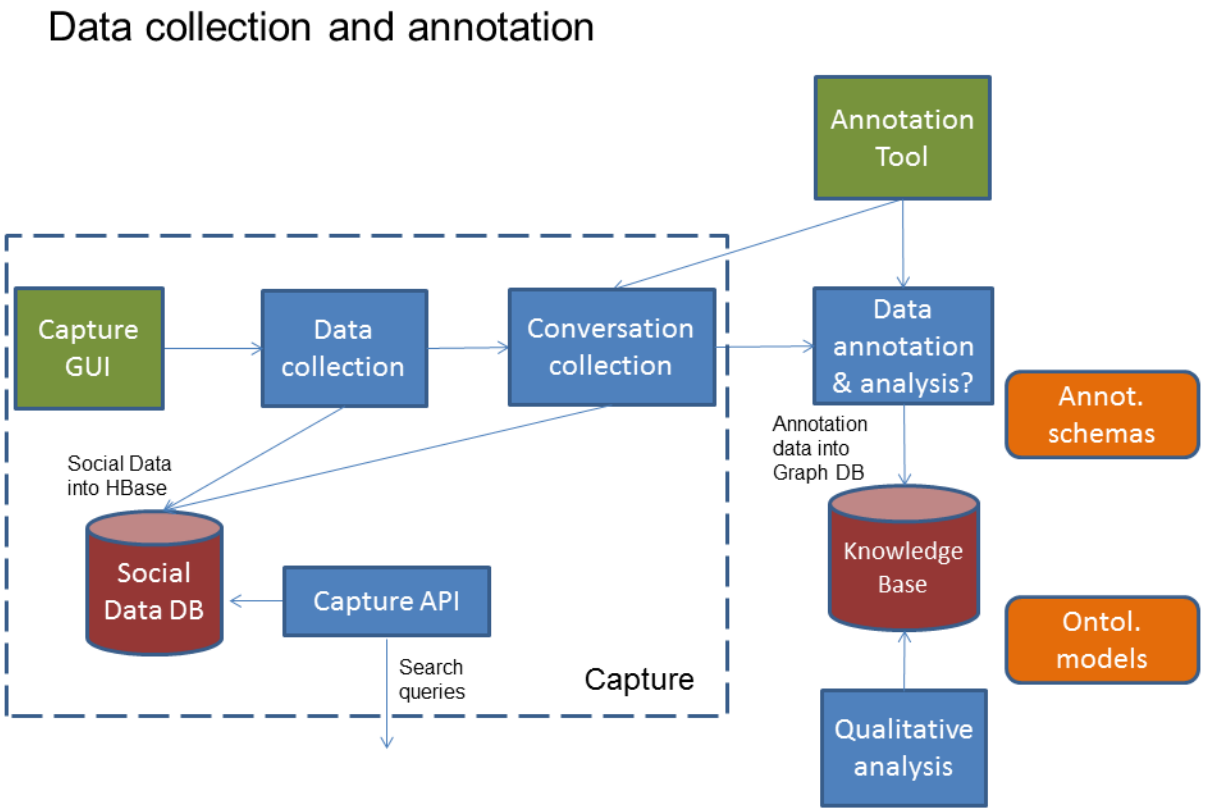


Figure 10. Data collection and annotation workflow.

A user (i.e. a journalist) sets up a particular data collection (i.e. a data channel collecting tweets of a specific event such as the Scottish independence referendum). The figure above shows the Capture module that collects the data for the specific queries that the user wants to launch. Once the user looks at the data stored in the Social Data repository (depicted as Social Data DB in the figure), they might decide to collect several conversations related to specific tweets likely to be giving rise to rumours. These conversations are stored in the social media content repository and then released for annotation. The user then makes use of the data annotation module to annotate the tweets (using the PHEME annotation schema from the Knowledge repository). The annotations, which are manually performed by the user, are stored in the knowledge repository (Knowledge Base in the figure).

As explained in section 7.1. , Capture allows for the creation of data channels and data sources to retrieve and store data collected from the sources specified in the data channel. Secondly, the likely rumourous tweets are sampled, which has been performed so far by selecting the most highly retweeted tweets. Lastly, the Conversation Collection tool collects complete conversations for the sample of likely rumourous tweets; the conversation includes all the tweets that responded to each of the tweets. The conversation collection step is performed by using the tweet ID of each of the selected tweets and, since the Twitter API does not provide an endpoint to achieve this, the web interface is scraped instead. At this point, the integration of both tools lies on the collection of tweets on the Capture site.

The steps to take to achieve full integration of both tools would be the following:

i. Integrate in the Conversation Collection (CC) tool the access to the entire data channel defined in the context of PHEME by means of Capture REST API calling the data channel API (see section 7.1. for service endpoint and invocation details).

ii. Call the Capture REST API from the CC tool to get all the tweets gathered in a concrete data channel (DC).

iii. Select the set of relevant tweets from a DC by means of a filtered search based on a threshold that defines the minimum for number of retweets over the DC calling to http://95.211.84.96:8080/CaptureREST/**{dcId}**/twitter/retweetCount?value=100[12]

iv. Reproduce the conversation with the CC tool and store it in Capture. To do that Capture has to be extended providing a REST method that supports CRUD operations over a concrete conversation.

4.2.2. Rumour classification in near-real time

This workflow depicts one of the main goals of PHEME: real-time detection of rumours. Once the models are trained, a real-time pipeline will classify the incoming flow of tweets and detects associated events. Figure 11 depicts the main components and iterations involved in this workflow:

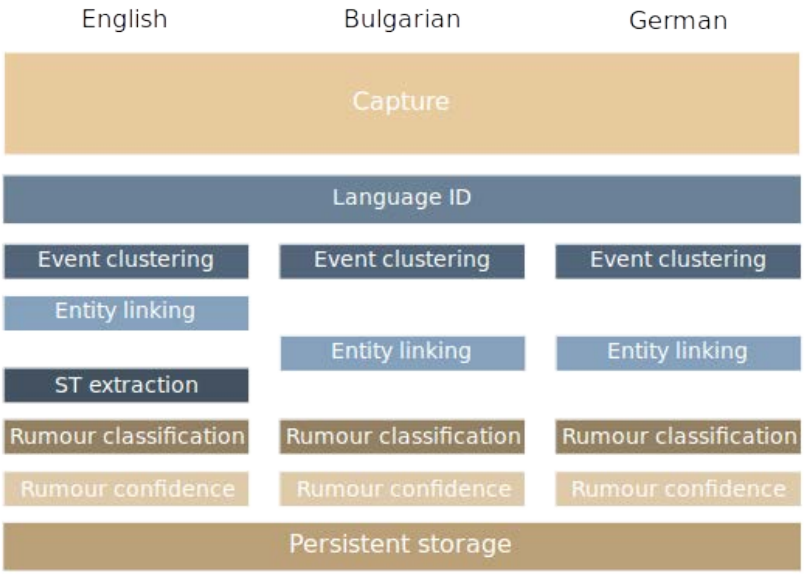


Figure 11. Near real-time rumour classification process.

The Capture data collection module starts gathering tweets in real-time (a stream of tweets) that is fed into the rumour detection pipeline. The figure above depicts how the pipeline starts with the incoming stream of social data. The data is then subject to several analysis, such as language detection, Named Entity Recognition (NER), Rumour classification, Spatio-temporal grounding, etc. Note that not all the steps of the process are implemented so far. Some of the components are not suitable yet for real-time processing, so the pipeline might be adapted as the throughput of the pipeline must be balanced to avoid bottlenecks. The objective of the pipeline is however the classification of the tweets and its visualisation and usage in the different use cases.

Kafka is used to decouple the interactions between the different components of the pipeline.

At the time of writing this document, the project is analysing the scalability of the pipelines to ensure the right performance and solve scalability issues. During year 3 the pipelines might

[12] {dcId} refers to the id of the current data channel

change depending of the results of this analysis and the performance of the different components.

4.2.3. Model training processes

Once annotated, the data can be used to train models (i.e. to train models to classify tweets as rumours). *Figure* 12 depicts the main components and iterations involved in this workflow:

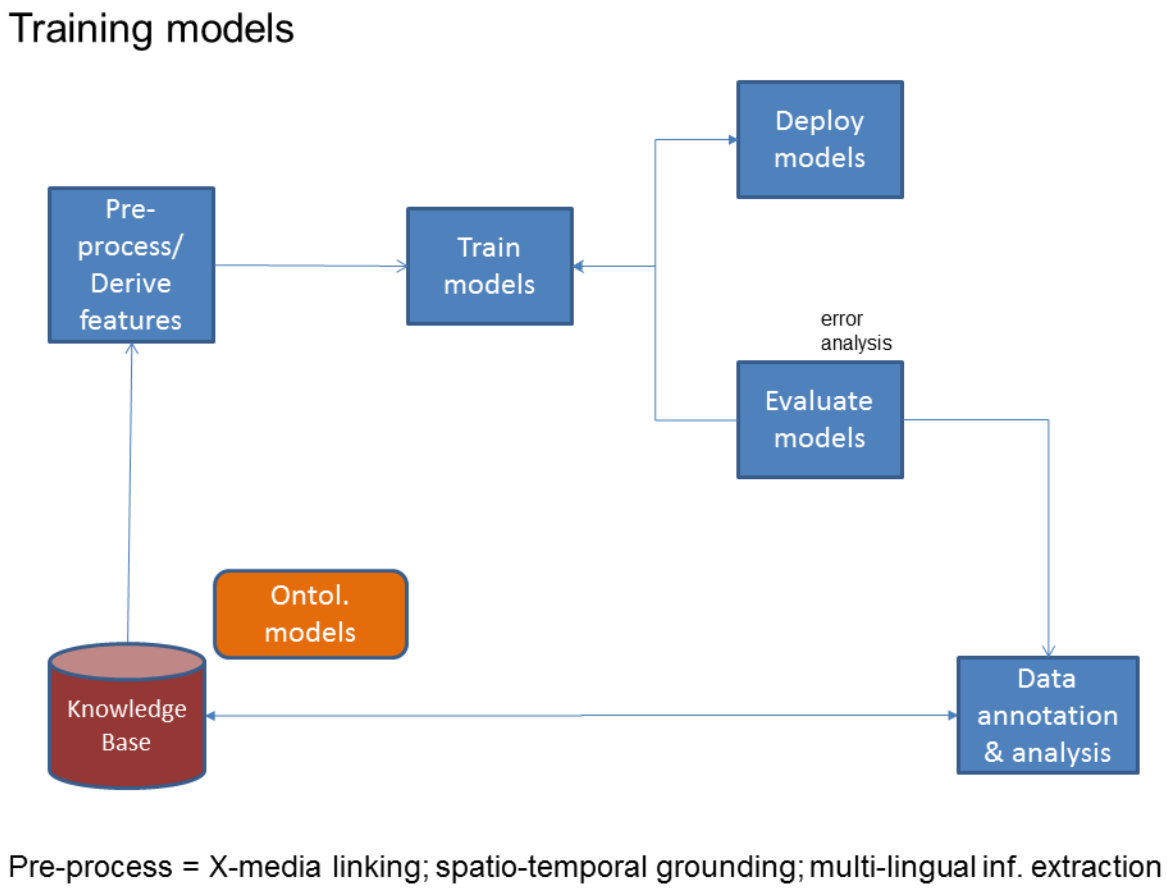


Figure 12*: Model training workflow.*

In this workflow a set of steps to pre-process the data coming from the knowledge repository and to derive features will be done before it starts to train the models. The models will then be deployed and evaluated, creating a feedback loop to improve the annotations and the models. This is a simplified view of a process that might require several iterations and involve the collection and annotation of new data (previous workflow). This workflow is necessary to provide the models for the next workflows.

In this case, the first stage in the integration process is the conversation retrieval by the Data Annotation & Analysis tool.

In a second stage, the integration process goes into several directions:

i. harmonisation between the data annotation scheme and developed ontology which models social and information diffusion networks
ii. their mapping to LOD resources
iii. synchronization of all available knowledge resources for search and extraction of veracity information
iv. integrate curation of content (tweets) with UWAR and GraphDB

Point (i) is ensured by the close cooperation among partners in the joint development of the annotation scheme and the ontology. Point (ii) will be ensured by the role of PROTON as LOD and data aggregator. The synchronisation (iii) will be achieved via the creation of a semantic repository, tuned to the PHEME specific needs. Point (iv) enables UWAR to directly query the SPARQL endpoint of GraphDB for enrichment schemas, tweets, metadata, and to write back to GraphDB manually enriched tweets and replies on tweets.

The data now is preferably in English, since most of the necessary resources are available for English. However, also German and Bulgarian datasets have been created, which will be developed further.

5. Conclusion and next steps

This document reports the work carried out during first two years of the projects in WP6 related to data collection tools, content, knowledge and software integration covering aspects such as the PHEME Architecture, Data Collection framework, Integration approach or Integration achievements.

The document describes the current version of the architecture and integration approach followed. The architecture is aligned with several EU and world-wide standardization initiatives, such as the Reference Architecture proposed by NIST for big data or the one proposed by the MLi project for Language Technologies. The integration approach is based mainly in Apache Kafka, an open source, distributed publish/subscribe tool that has been used widely within the project in year 2 that provides the adequate means to create loosely-coupled pipelines and therefore ideal for data-driven integration. An overview of the main processes, pipelines and components is also described.

The work carried out within integration provides the glue for some of the components and functionalities developed within the project so far. However, it is an ongoing process. Some of the components are still under development or being enhanced, and not all the functionality delivered in the last months is completely integrated. Therefore there is still plenty of work to do in WP6 related to integration. The final third year of the project will be devoted precisely to integrate the main functionalities to enable its use by the PHEME use cases.

6. Bibliography and references

Apache HBase: http://hbase.apache.org/

Apache Hadoop: http://hadoop.apache.org/

Apache Kafka: http://Kafka.apache.org/

Apache Storm: https://storm.apache.org/

Apache Solr: http://lucene.apache.org/solr/

GraphDB: http://www.ontotext.com/products/ontotext-graphdb/

7. Annex 1. Component descriptions

7.1. Data collection framework: Capture

7.1.1. Description

User-generated content and social networks are highly dynamic, thus the PHEME data collection tools will track over time the content created by a given user, exchanges with other users, as well as changes in their profiles and social networks. For that purpose, the partner ATOS provides in the context of PHEME a data collection tool named "*Capture*".

From the conceptual point of view, the main concept in Capture is the ***Data Channel***. A data channel is the way users can group query results below a single umbrella. Data channels allow defining several queries or *Data sources* to social networks. All results are stored and indexed associated to the channel, giving the user the possibility of knowing and querying for their results, rather than having them all in a single storage.

Data Channels have associated metadata that describes them. This metadata is defined by means of the set of attributes:

- channelID: unique identifier of a data channel
- description: data channel description
- name: data channel name or Title
- creationDate: Date/time when the data channel has been created
- updateDate: Date/time when the data channel has been updated last time
- startCaptureDate: Date/time to start the collection of data for the data channel
- endCaptureDate: Date/time to finalize the collection of data for the data channel

The second entity that enriches the definition of Data Channel is the ***Data Source***. A Data Source represents a specific web resource (i.e. Twitter, RSS, Reddit, Facebook, etc.) and the definition of queries or filters that the system will perform. Conceptually, a Data Channel could be composed of 1 to N Data Sources, giving the possibility of making several queries to the same or different resources in the same data channel (i.e two different queries to Twitter and RSS grouped in the same data channel). The Data Source is defined by the following attributes:

- type - The type of the data source, currently supported for twitter and RSS
- sourceID - The ID of the data source
- keywords - The actual query definition.

7.1.2. Technical Perspective

Capture is a solution for social data collection that is based on big data technologies. It relies on the concept of gathering content using dedicated data channels. A data channel listens to data from different data sources (i.e. Twitter), meaning that data channels implement configurable user queries (based on keywords, hashtags, locations, etc.) to gather data (tweets) associated to the channel. The users are therefore able to set up several data channels for different purposes (i.e. to listen to specific events, or search for mentions of legal highs in Twitter), providing that the search limits provided by the APIs of the social networks (i.e. the limits of the public Twitter search and/or streaming APIs) are respected. Therefore, the data can be collected in very flexible ways. Capture also provides a search API to query for the data collected. It is also

worth mentioning that Capture enables pipelining different analytical components both in batch or real time, giving an extra integration flavour to the data collection tool.

Figure 13 depicts the main building blocks of the Capture module along with some of the provided functionalities.

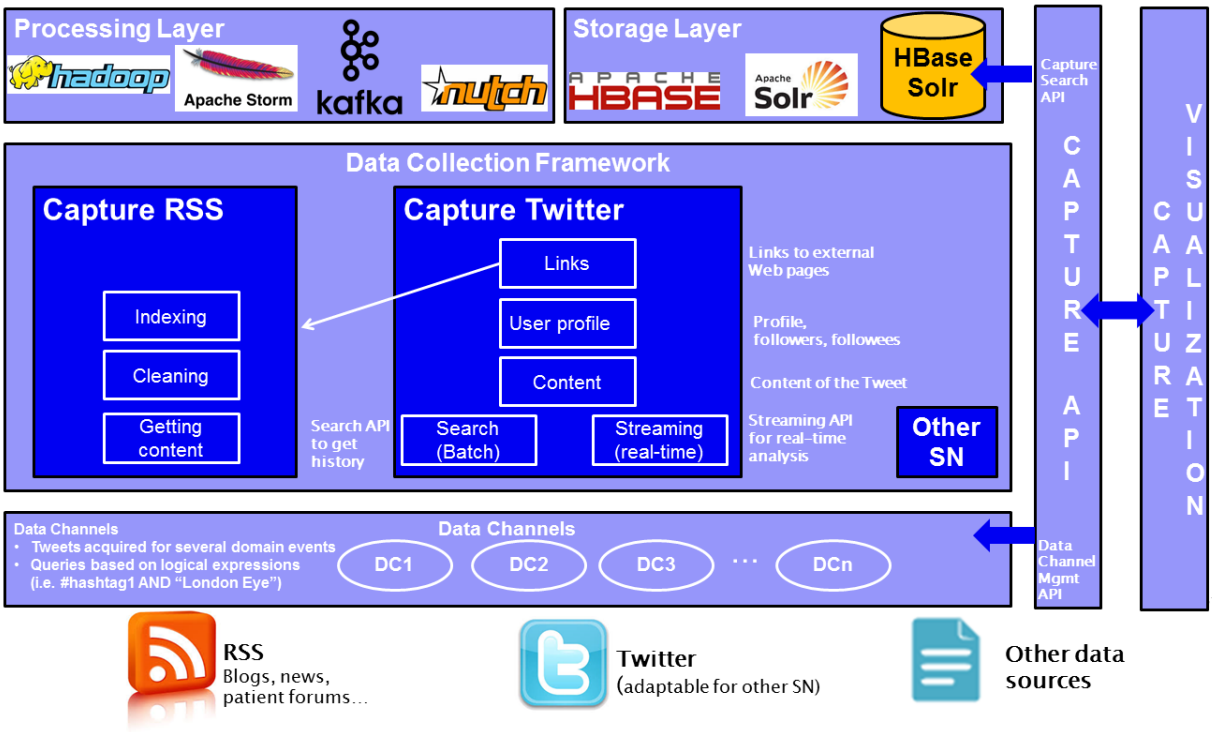


Figure 13. Detailed overview of the Data Collection framework (Capture)

7.1.3. Deployment Environment

Capture is deployable in any UNIX environment with the following characteristics:

- Tomcat 6-7 or Jetty 9
- Java 7
- Apache Kafka 0.8.2.2 or later,
- Apache HBase 1.1.1 or later,
- Hadoop 2,
- Apache Solr 5 or later
- Apache Flink 0.10

The recommended minimal physical architecture is: 4 CPU cores, 8 GB RAM and 500 GB HD (extensible if the storage needs grows in time)

Capture expose two sets of user interfaces: (1) Capture REST API and (2) Capture Web GUI.

7.1.4. Invocation guidelines

The Data Collection tool Capture is deployed as RESTful services. The signature of the service is provided in this section.

The examples below show how to use the Capture REST API for the following actions. Note that the URL is relative. For the Pheme deployment, the root URL is:

http://pheme-capture.gate.ac.uk/CaptureREST

Since the last version of the data retrieval services there have been few subtle modifications, related to the data retrieval from data channels and pagination.

1) Data channel manipulation services

- Create a Data channel (DC) with data sources (DSs):
  - *Step 1: Create DC and DSs*
    - URL: /rest/datachannel/
    - HTTP operation: POST
    - Content:

```
<dataChannel>
<channelID>a59fb3dc-94cd-444c-a730-10c8bb18464c</channelID>
<creationDate>2014-09-05 10:59:05.477</creationDate>
<updateDate>2014-09-05 10:59:05.477</updateDate>
<description>Refined Data channel for legal highs</description>
<startCaptureDate>2014-09-05 10:30:00.000</startCaptureDate>
<endCaptureDate>2014-09-12 15:45:00.000</endCaptureDate>
<name>Data collection for Pheme</name>
<status>active</status>
<type>search</type>
<dataSources>
<twitter>
<type>Twitter</type>
<sourceID>deeddc3e-dc26-4da0-a10e-5eecc6c09fd3</sourceID>
<keywords>
lang:en geocode:51.5085300,-0.1257400,30km "WHITE MAGIC" OR "MIAOW MIAOW" OR "MEOW MEOW" OR MEPH OR "M-SMACK" OR "M-CAT" OR "MEPHEDRONE"
</keywords>
</twitter>
</dataSources>
</dataChannel>
```

  - *Step 2: Querying for the acquired data*
    - To get data using pagination and a specific page size: /rest/datachannel/{dataChannelID}/data?fromId=348173591752810 &numResults=150
      fromId field can be obtained automatically from "lastTweetId" property of the "tweet_list" field of each non-empty result.
    - By default fromId is empty and numResults = 100
  - Get All Data Channels in current Capture instance:/rest/datachannel/
  - HTTP operation: GET
  - Get Data Channel Configuration:/rest/datachannel/{dataChannelID}
  - HTTP operation: GET
  - Update Data Channel:/rest/datachannel/{dataChannelID}
  - HTTP operation: PUT
  - Content:

```
<dataChannel>
<channelID>325be839-6585-4e98-bb26-5018222fe464</channelID>
<creationDate>2014-07-09 11:31:38.271</creationDate>
```

<dataSources>
<twitter>
<type>Twitter</type>
<sourceID>29413386-8351-42e2-a51f-a32f78877d3c</sourceID>
<keywords>Tour 2014</keywords>
</twitter>
</dataSources>
<description>Tour DC Updated</description>
<endCaptureDate>2014-07-09 15:26:00.000</endCaptureDate>
<name>Lab TEST</name>
<startCaptureDate>2014-06-12 17:57:51.401</startCaptureDate>
<status>active</status>
<type>search</type>
<updateDate>2014-07-09 11:31:38.271</updateDate>
</dataChannel>

- Delete Data Channel (logically):/rest/datachannel/{dataChannelID}
- HTTP operation: DELETE
- Get Twitter User Profile:/rest/datachannel/twitter/user/{userId}
- HTTP operation: GET
- NOTE: This method allows to retrieve the user profile (with followers and followees) using a given user handler. It stores the user if it is not in our KB

2) Faceted Query services

- Faceted query services allow to look at the stored data through the following facets:
  - "dcID" – data channel Id a document belongs to
  - "createdAt" – creation time of a document, in ISO format, e.g.: 2015-11-22T06:26:50+00:00
  - "text" – textual content of the document (allowing full text search)
  - "favouriteCount" – twitter specific field, number of times a tweet was starred as favourite
  - "tweetID" – twitter specific field, twitter assigned ID of a message
  - "hashTags" – twitter specific field, list of hashtags in the tweet text
  - "retweetCount" – twitter specific field, number of retweets of an original message
  - "originalTweetId" – twitter specific field, id of an orginal tweet in case that the message is a retweet
  - "inReplyToId" – twitter specific field, id of a parent message, in case that the tweet is a reply to other tweet
  - "sourceUrls" – twitter specific field, list of URLs contained in the tweet text
  - "userFollowers" – twitter specific field, number of followers of the author of the message
  - "userFollowes" – twitter specific field, number of the followees of the author of the message
  - "userID" – twitter specific field, numerical ID of the twitter user

        - "userScreenName" – twitter specific field, textual user name, as presented by twitter
        - "latitude" – twitter specific field, latitude part of the geolocation data (if exists)
        - "longitude" – twitter specific field, longitude part of the geolocation data (if exists)
        - "dcTweeetID" – capture specific field, a composed id (dc id + tweet id)
        - "typeDS" – capture specific field, social networks name ("twitter" for Twitter messages)
- URL endpoint: /rest/datachannel/data
- Parameters:
    - filterExpression – facet(s) to use as a filter. Different facets are separated by "AND" operators. e.g.: (createdAt:[2016-01-10T00:00:00.000Z TO 2016-01-13T00:00:00.000Z] *AND* sentiment_feature:1.0)
        - "text" field:
            - text:vriendinnetjes AND/OR liefste → "vriendinnetjes" and/or "liefste" in any position in the text
            - text:"vriendinnetjes liefste" → "liefste vriendinnetjes" in the text
            - text:Vriendin* → "Vriendin" in any position in the text as a word or as a lema
            - text:Vriendin* AND/OR Gezellig → "Vriendin" in any position and/or the word "Gezellig" in any position.
    - sorter – field to use as a sorting order. Default sorting field: tweetID.
    - mode – sorting mode: asc or desc. Default: desc
    - id – limit results to concrete data channel
    - typeDataSource - limit results to concrete social network type. Default: Twitter.
    - numResults – number of results per page
    - page – number of the page
    - fields – coma separated list of fields to return
- example:
  Retrieve all documents from data channel "1f55451f", with creation date between 2015-04-13 07:00:00 UTC and 2015-04-13 08:00:00 UTC, sort by "retweetCount" field, and include the following fields only: retweetCount, userID, text, tweetID, createdAt:

  /rest/datachannel/data?filterExpression=(createdAt:[2015-04-13T07:00:00.000Z TO 2015-04-13T08:00:00.000Z])&id=1f55451f&sorter=retweetCount&mode=desc&fields=retweetCount,userID,text,tweetID,createdAt

- example:

  Retrieve all documents from data channel “1773632b”, with text containing “esplendido” and “video” words in any place, with positive sentiment, and including all fields:

  rest/datachannel/data?filterExpression=((text:esplendido AND video) AND sentiment_feature:1.0)&id=1773632b

7.2. Knowledge repository: GraphDB

7.2.1. Description

The Knowledge repository has been explained in detail in section 2.2.2. This section expands the GrapDB description with its deployment and invocation guidelines.

7.2.2. Deployment Environment

GraphDB is deployable at any UNIX environment with:

- Tomcat 6-7
- Java 7

The minimal physical architecture required is: 8 cores, 20 GiB RAM, 100 GiB SSD.

GraphDB expose two sets of user interfaces: (1) Openrdf-WorkBench and (2) GraphDB WorkBench and a SPARQL endpoint.

7.2.3. Invocation Guidelines

GraphDB is currently deployed at ONTO with Openrdf-WorkBench, GraphDB WorkBench and SPARQL endpoint, and can be accessed at http://pheme-repo.ontotext.com. The following ontology files are deployed:

- <file://PhemeOntology-v2.ttl>
- <file://Travis-Allen-Twitter-Ontology2.owl>
- <file://atc2en_de_es.ttl>

After a log in with user name “admin” and password “root” one may select a predefined query under SPARQL/Query.

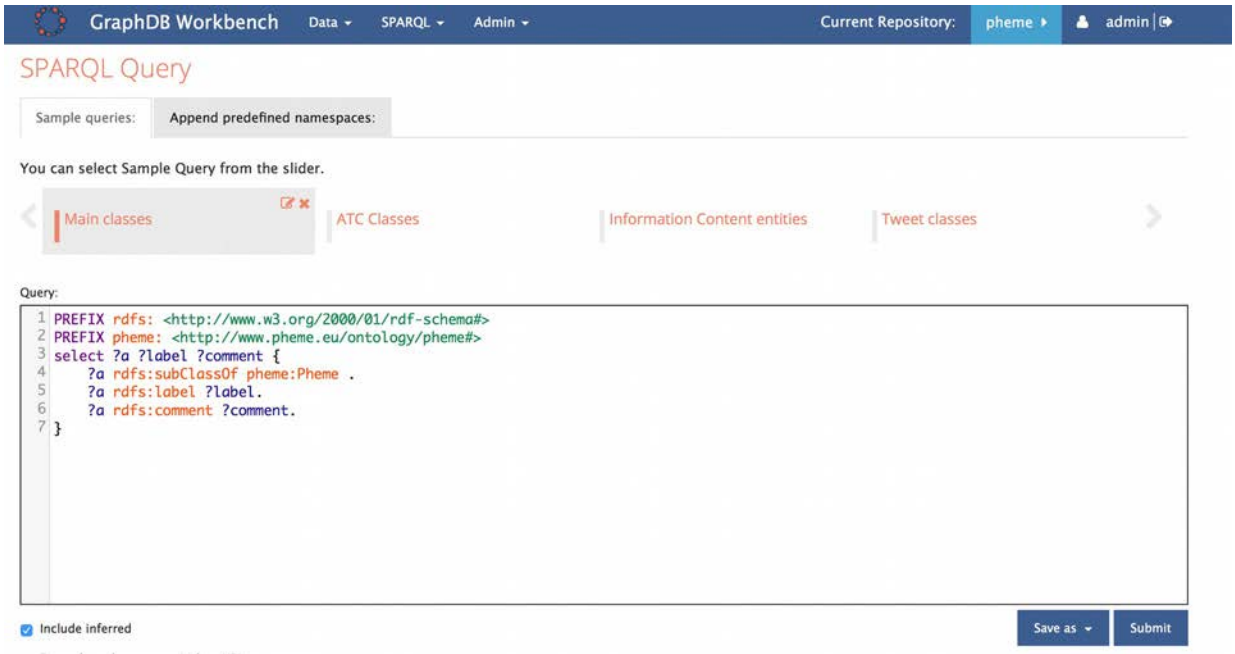


Figure 14. SPARQL query interface.

GraphDB reveals the results in a set of interfaces for RDF navigation and exploration, as shown in Figure 14 and Figure 15 below.

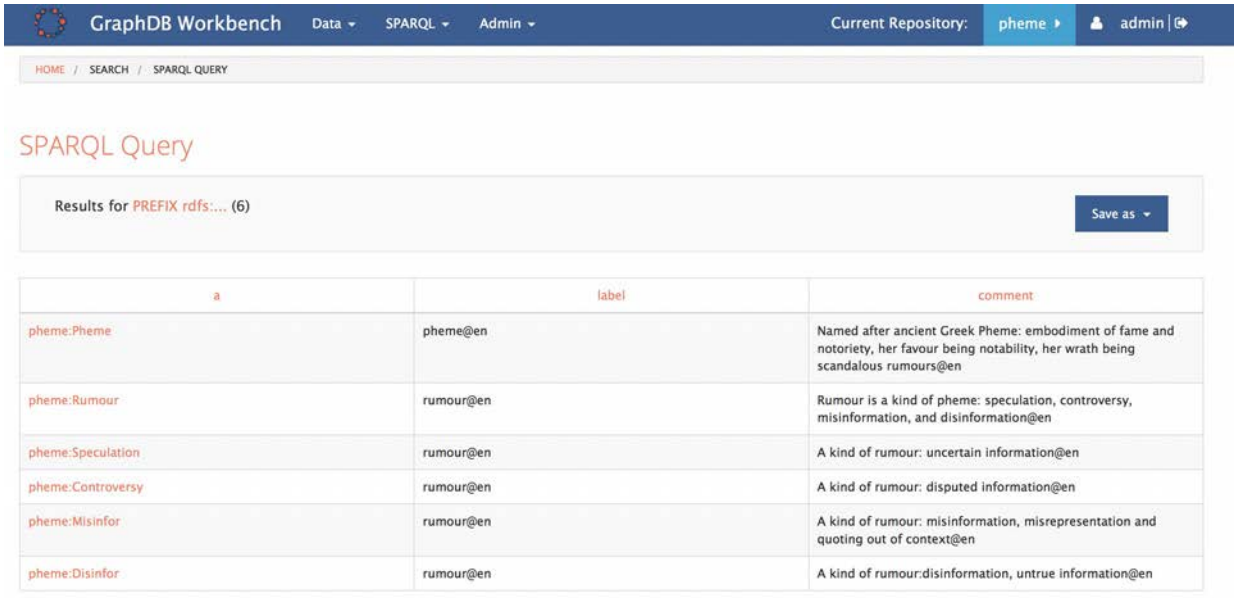


Figure 15. RDF query result set.

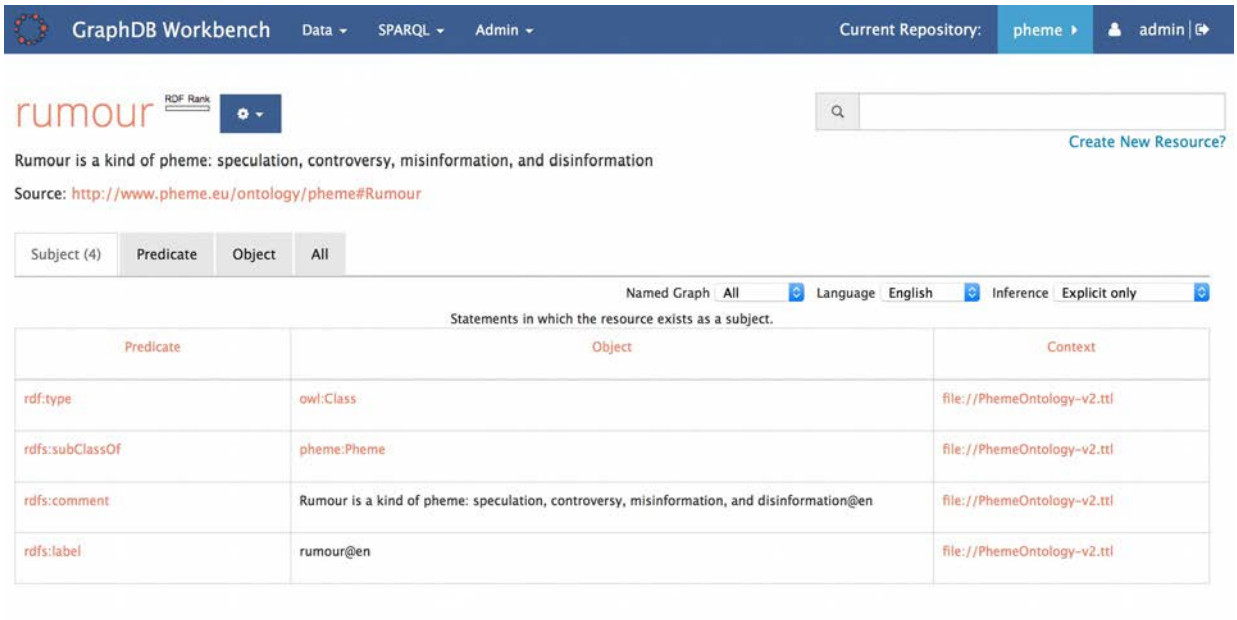


Figure 16. PHEME ontology navigation.

Sample queries that would help to explore further the currently deployed ontologies, including the rumour classification scheme and the tweeter metadata vocabulary, are shown below.

```
###Pheme classes
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pheme: <http://www.pheme.eu/ontology/pheme#>
select ?a ?label ?comment {
?a rdfs:subClassOf pheme:Pheme .
?a rdfs:label ?label.
?a rdfs:comment ?comment.
}

###ATC Classes
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select * {
?a rdfs:subClassOf atc:A01.
?a rdfs:label ?label.

}

###Information Content entities
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select * {
?a rdfs:subClassOf <http://purl.obolibrary.org/obo/IAO_0000030>.
?a rdfs:comment ?comment
}

###All Tweet subclasses
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select * {
?a rdfs:subClassOf
<http://www.semanticweb.org/travis/ontologies/2013/10/ontologicalengineeringtwitter#entity>.
```

```
OPTIONAL {
?a rdfs:comment ?comment.
}
OPTIONAL {
?a rdfs:label ?label.
}
}
```

Example request to the SPARQL endpoint of GraphDB that is equivalent to “Information Content entities” query is narrated bellow.

Sparql request:

http://pheme-repo.ontotext.com/repositories/pheme?query=PREFIX+rdfs%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F2000%2F01%2Frdf-schema%23%3E%0D%0Aselect+*+{%0D%0A++++%3Fa+rdfs%3AsubClassOf++%3Chttp%3A%2F%2Fpurl.obolibrary.org%2Fobo%2FIAO_0000030%3E.%0D%0A++++%3Fa+rdfs%3Acomment+%3Fcomment%0D%0A}&implicit=true

Response (partial):

```
<?xml version='1.0' encoding='UTF-8'?>
<sparql xmlns='http://www.w3.org/2005/sparql-results#'>
  <head>
    <variable name='a'/>
    <variable name='comment'/>
  </head>
  <results>
    <result>
      <binding name='comment'>
        <literal>an information content entity is an entity that is generically dependent on some artifact and stands in relation of aboutness to some entity -IAO</literal>
      </binding>
      <binding name='a'>
<uri>http://purl.obolibrary.org/obo/IAO_0000030</uri>
      </binding>
    </result>
```

Below we show an example of Sparql query that illustrates the value of integrating LOD knowledge with the Pheme ontology. We want to retrieve Tweets that are related to the location New Jersey. Ideally, we expect to retrieve not only Tweets that mention New Jersey directly, but also Tweets that mention cities located in the New Jersey area.

```
PREFIX pubid: <http://ontology.ontotext.com/resource/>
PREFIX pheme: <http://www.pheme.eu/ontology/pheme#>
PREFIX pub: <http://ontology.ontotext.com/taxonomy/>
PREFIX geo-ont: <http://www.geonames.org/ontology#>
PREFIX dlpo: <http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo>
select * {
        #Select New Jursey state
        ?parent pub:preferredLabel "New Jersey"@en.
        ?parent pub:exactMatch ?geoparent.
        ?geoparent a geo-ont:Feature.

    #Now select tweets mentioned with places inside New Jersey.
        ?t pheme:containsMention ?a.
        ?a pheme:inst ?pub.
        ?pub pub:preferredLabel ?pubLabel.
        ?pub pub:exactMatch ?geo.
        ?geo geo-ont:parentADM1 ?geoparent.
        ?t <http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#textualContent> ?text.
} limit 100
```

Below we show a snapshot of the results. They comprise tweets that mention Burlington and Atlantic City.

3	pubid:tsk7qofr0idc	http://sws.geonames.org/5101760/	pheme:tweet-656937079817965568	http://www.pheme.eu/resources/pheme/06149c051e27408698f9809e21f8d597	pubid:tsk4zj4iy1a8	"Burlington County"@en	http://sws.geonames.org/4500994/	@realDonaldTrump @i_am__tyler @MichaelCohen212 going be down in Burlington tonight
4	pubid:tsk7qofr0idc	http://sws.geonames.org/5101760/	pheme:tweet-655048472438018048	http://www.pheme.eu/resources/pheme/f797a85f30544d09898fc6a5731b79e5	pubid:tsk4wgmr5qf4	"Atlantic City"@de	http://sws.geonames.org/4500546/	Wed Nov 25th The B-Street Band pre Thanksgiving Party at the Trump Taj Mahal Casino/Hotel Atlantic City, NJ... http://t.co/qABxLJb8i6
5	pubid:tsk7qofr0idc	http://sws.geonames.org/5101760/	pheme:tweet-655080196270194688	http://www.pheme.eu/resources/pheme/6620da7a119b4addba9263eecf6df610	pubid:tsk4wgmr5qf4	"Atlantic City"@de	http://sws.geonames.org/4500546/	Atlantic City celebrates #Oktoberfest2015 at Trump Taj Mahal Oct 24-25. #JerseyShoreHomes http://t.co/UEtkcunWoY
6	pubid:tsk7qofr0idc	http://sws.geonames.org/5101760/	pheme:tweet-655074504083943424	http://www.pheme.eu/resources/pheme/705cbdf88d2a4222a6420e49aa7ac71c	pubid:tsk4wgmr5qf4	"Atlantic City"@de	http://sws.geonames.org/4500546/	Do you think that you missed your only chance to see 50 Shades of Gay at the Trump Taj Mahal in Atlantic City... http://t.co/UcEqFs3tHk

Figure 17. SPARQL results snapshot

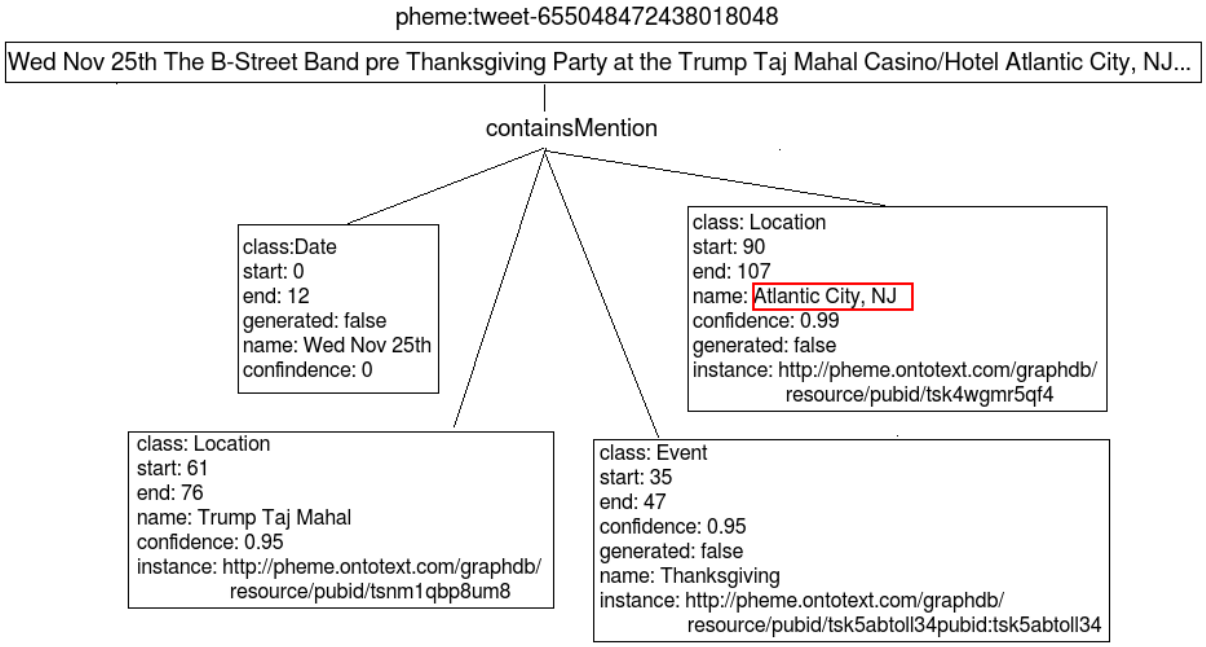


Figure 18. Example of tweet from with Location "Atlantic City", as result of a query on New Jersey-related tweets.

7.3. Conversation Collection Module

The conversation collection module developed by UWAR is a web service developed as a web interface or API that allows the user to collect (and visualise) the thread of tweets replying to a specific tweet. The user needs to specify a single source tweet, providing its URL or tweet ID, and the tool collects all the replies to that tweet. The tool allows the user to retrieve two different types of outputs: (I) JSON-formatted output including all the tweets in the conversation, which is useful for the integration with the other modules, and (ii) a web-based output that renders the replies in an HTML file in a forum-like visualisation of the thread, which is convenient for end users.

NOTE: Since retrieval of replying tweets has been discontinued in Twitter API v1.1 (it used to be available through the 'related_results/show' endpoint in v1.0), this tool scrapes the replies from the HTML of the source tweet.

7.3.1. Technical Perspective

The conversation collection module is composed of two main components. The FrontEnd component that collects the input of the user and shows the output, and the BackEnd component that retrieves the conversations from Twitter.

When a user submits the tweet ID or tweet URL, the FrontEnd component calls the BackEnd component to retrieve the whole conversation for that tweet. The BackEnd component then accesses the URL of that tweet, to scrape the tweet IDs that participated in the conversation. The BackEnd then collects the content of those tweet IDs from the Twitter API. Note that:

- Twitter pages the replies, so the script also checks if there are more pages with replies.
- The script then retrieves, recursively, the replies for all those replying tweets, given that some replies can only be accessed this way occasionally.

Note that this recursive tweet collection can be slow on some occasions, which is why the prior data collection module samples the tweets so as to make the scale of this collection phase feasible.

The content of the whole conversation, including both the source tweet and all the replying tweets is then returned to the user, either visualised in a web page, or in JSON format, as specified by the user. The latter is used for the integration with Capture, which accesses the Conversation Collection Module to retrieve conversations for some of the tweets. Figure 19 shows an example of a conversation produced for a tweet with this module, in this case with the visual output.

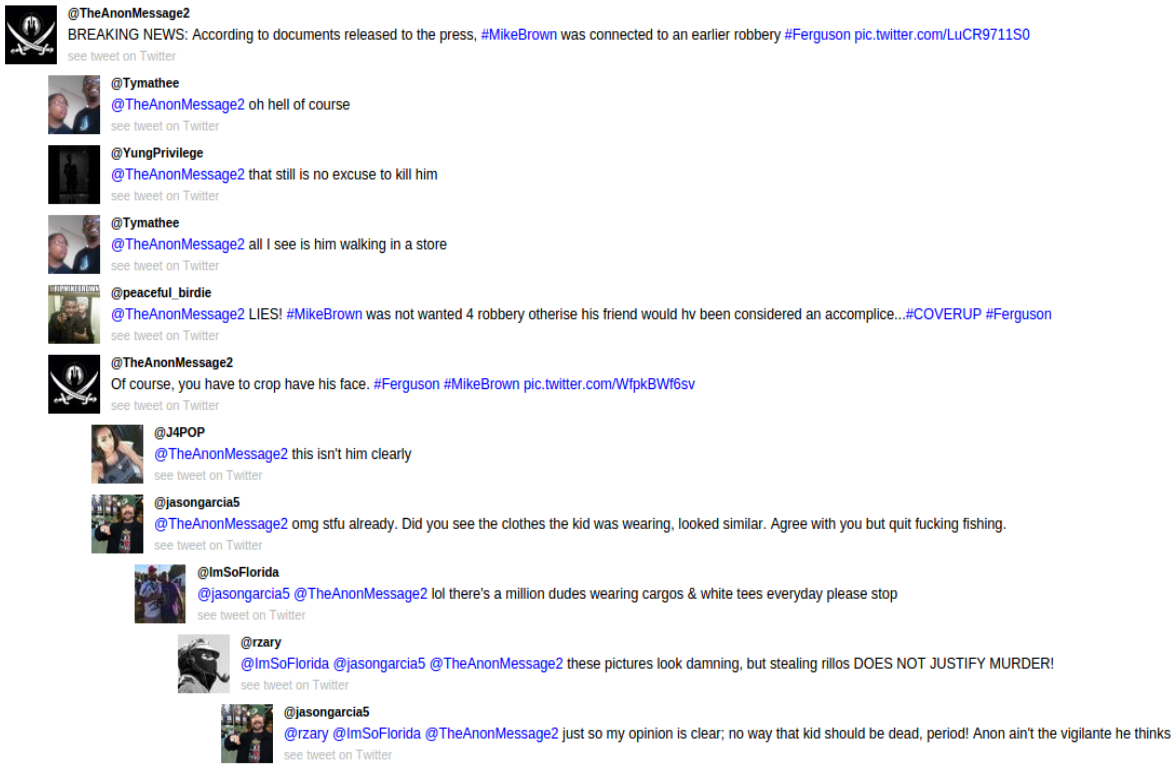


Figure 19. Sample conversation collected with the Conversation Collection Module

In the integration of components at PHEME, the Conversation Collection Module gets a request from Capture, which includes a tweet ID. The Conversation Collection Module then retrieves the whole conversation as specified above, returning it back to Capture.

7.3.2. Deployment Environment

The conversation collection module is implemented to run on an Apache web server, and is written in PHP and Python. PHP is used for the implementation of the FrontEnd, and Python is used for the implementation of the BackEnd. The access to the Twitter API, which is performed in the BackEnd, is done with the Tweepy open source library for Python (the integration with Capture will allow to use the Capture libraries to get the tweets).

7.3.3. Invocation Guidelines

The web application is available at: http://www.dcs.warwick.ac.uk/~arkaitz/threads/

The URL above shows a web interface where the user can specify either the URL of a tweet, or the ID of a tweet. In order to retrieve the conversation, the tweet specified has to be a source tweet, i.e., it cannot be a reply to another tweet.

The user can either use the web interface to submit a tweet ID or URL, or instead it can be integrated with other modules by sending the tweet ID or URL by POST. This is the endpoint that is being used in PHEME for the integration with the other modules.

7.4. Data annotation & analysis module

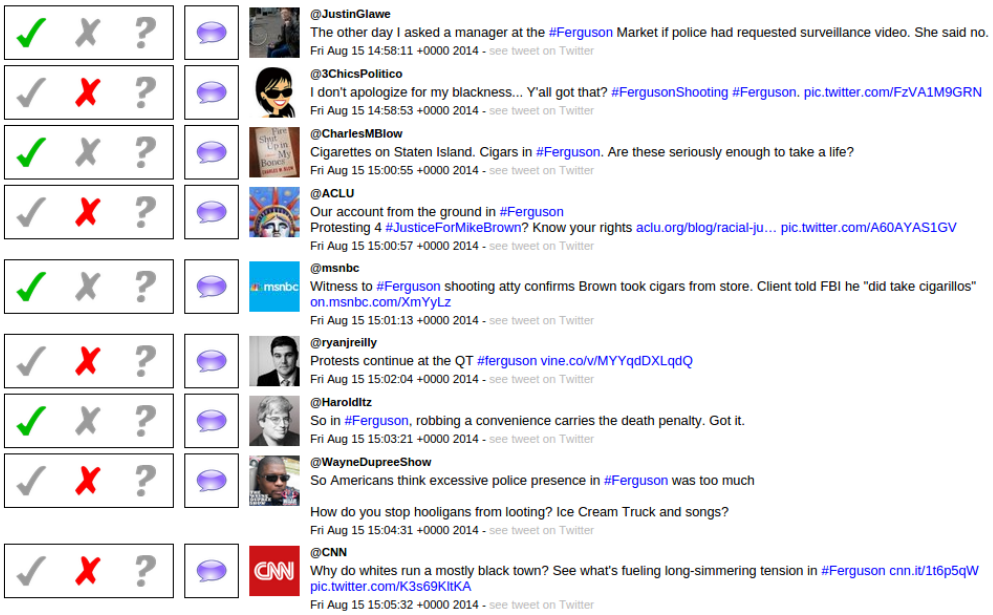
The data annotation tool is a web application that can be used to visualise streams of Twitter data, and to annotate the tweets as being rumours or not.

7.4.1. Technical Perspective

The data annotation tool retrieves all the tweets from Capture, and displays them in a timeline of tweets for a convenient visualisation that enables the user to perform the annotation. This tool requires integration with both Capture, to collect tweets, as well as with the Conversation Collection Module, to retrieve conversations.

The data annotation tool requires two values as input from the user: a Twitter query, and a number of retweets as a threshold. The number of retweets is used as a threshold to pick tweets with more retweets than that, especially given that the number of tweets tends to be so large usually that a human annotator could not manage. Having a query and a threshold as input, the data annotation tool makes a request to Capture in order to retrieve all the tweets that match the query and have a higher number of retweets. The tool then accesses the Conversation Collection Module to retrieve the conversations associated with those tweets.

All the tweets obtained through Capture and complemented through the Conversation Collection Module are then displayed in a web interface (see Figure 20 for an example), where the user can annotate each tweet as being a rumour or not. In the list of tweets to be annotated, the annotator can choose to visualise the associated conversation by clicking on the bubble next to it, mark it as a rumour by clicking on the check mark, mark it as a non-rumour by clicking on



the cross, or leave it for later by clicking on the question mark.

Figure 20. Screenshot of the data annotation tool

The output of the annotation tool is then saved in the GraphDB (Owlim) database.

7.4.2. Deployment Environment

The data annotation module is a web interface that is completely implemented in PHP, and runs on an Apache web server. The application also makes use of the AJAX technology, which enables to refresh the website with the latest annotations performed by the annotators without having to reload the whole page.

7.4.3. Invocation Guidelines

The data annotation tool is a web interface that the annotators can access and visualise. The user needs to access the following URL to visualise the web application:

http://www.dcs.warwick.ac.uk/~arkaitz/rumour-annotation/.

The user can see the list of events they are following there, and click on one when they are interested in annotating it. The events are organised by day, so the user can choose to visualise the tweets from the day they are interested in.

7.5. Event detection module

Event detection aims to detect newsworthy events from a stream of tweets. We develop a fast and efficient algorithm based on hierarchical Dirichlet process (HDP) to detect events from Twitter data in real time. The algorithm is discussed in detail in the deliverable D3.3.1 and was evaluated as the best one for the event detection task in Pheme. The algorithm acts upon a stream of tweets and outputs the event cluster id along with the tweets. The algorithm reads messages from a Kafka topic and assumes the messages to be in the Kafka message format described in Section 2.3.2. It clusters the tweets based on the topics determined by a hierarchical Dirichlet process model and output the cluster id associated with each tweet. The cluster id is available through the "event_cluster" field of the output Kafka The HDP algorithm requires one to specify the input Kafka stream to read from and the output Kafka stream to write. It also considers an optional third argument denoting an upper bound on number of topics to be produced for a batch of at most 25k tweets (uses a default value of 100).

It uses a separate pre-processor for tweets from different languages and the same HDP algorithm works for tweets from different languages. It considers the "in_reply_to_status_id" field of the incoming Twitter message to group together tweets belonging to the same conversational thread.

The algorithm is developed using the Python programming language running in the Linux operating system environment. Python 2.7 or higher is required to run the algorithm. The algorithm is integrated in the Pheme project using the Apache Kafka framework and uses Kafka 0.8.1.1. The HDP algorithm uses the python packages Gensim and NLTK and will require installations of these packages in the system.

The algorithm can be invoked by calling the Python program runHDP_Kafka_stream3.py from the command line and providing the Kafka stream to read and write and an optional upper bound on number of topics as arguments.

Usage:

```
python runHDP_Kafka_stream3.py inputKafkastream outputKafkastream [numberOfTopics]
```

E.g.

```
python runHDP_Kafka_stream3.py pheme_en pheme_en_events 100
```

7.6. Spatio-temporal and named entity extraction module

It's important to know where and when a claim is being made. It's also important to know what time and place are being referred to; knowing that a message was written in New York is not a strong basis to draw conclusions if the content refers to Rome. While spatio-temporal grounding is dealt with partially in message metadata, extracting the times and places mentioned in social media messages is dealt with in this module, desribed in detail in D2.4.

The system finds mentions of times and places by casting this as a sequence labelling problem, trying to find strings of tokens in sentences that are in fact temporal or spatial referents. The overarching schema and definition is derived from ISO-TimeML and ISO-Space. The program used is the general purpose *entity-recognition* package,[13] created as part of Pheme and adapted with specific unsupervised representations and feature extractors to perform spatio-temporal chunking. The training data is also that from D2.4, a mixture of Pheme-originated annotated documents and also existing spatio-temporal corpora.

Named entity, spatial, and temporal extraction are all run as independent instances of the *entity-recognition* class. The NE and spatial variants use non-generalised Brown clusters from English tweets, with c=2500 and T=250 million; the spatial variant uses non-generalised Brown clusters induced over a blend of Reuters newswire and English tweets, with c=6000 and 64 million tokens of each genre (128 million total).

The code runs in Python 3 under any supported environment, and relies on CRFsuite, SciPy and NumPy. These are also ostensibly portable to any C-standard compliant platform; we run them under Linux.

The program, stored in the GATE extras repository (gate-twitter/stream/) is invoked from the command line:

```
./entities_Kafka.py
```

This connects to the Kafka broker, consuming JSON documents one by one and re-publishing them with overlaid *tokens*, mapping tokens to pairs of character offsets, and also a *pheme_entities* field containing a list of entities, for each the entity typesand bounding token offsets.

7.7. Language ID

It's critical to identify the language of content before sending it for linguistic processing. It's also worthwhile ignoring non-project-languages for event clustering, in order to reduce load and ease development. For this, we use state-of-the-art language ID to screen and split incoming content.

Analysis earlier in the project[14] covered the capability of language identification over social media content, where terseness of messages provides a real challenge to traditionally n-gram based approaches. Langid.py, with its *LD* feature selection, reduces the impact of noisy features and identifies language-discriminatory features that are stable across shifts of genre – perfectly

13 https://github.com/leondz/entity_recognition ; also USFD: Twitter NER with Drift Compensation and Linked Data, 2015. L Derczynski, I Augenstein, K Bontcheva. Proceedings of the W-NUT worshop.

14 Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, Kalina Bontcheva. 2015. Analysis of Named Entity Recognition and Linking for Tweets. Information Processing & Management; 51(2):32-49

suited to social media. It performed best overall in our trials. Twitter's recently-upgraded language ID is still not yet powerful enough for the task, working over a balance of existing non-social-media language ID solutions.

Therefore, we wrap langid.py with Kafka input and split-output interfaces, taking in any social media content and providing three language-specific streams for English, German and Bulgarian.

Langid.py is entirely Python, using an internally-encoded model, and so as platform-independent as that language. We have deployed this on Linux systems using the Kafka-python library.

The module is stored in the GATE Extras repository (gate-twitter/stream/). It reads from a given Kafka topic, consuming there and mapping ATOS Capture output into the Pheme/Twitter JSON format agreed and described in D6.2.1. Language specific messages are published on there Kafka topics: pheme_en, pheme_bg and pheme_de. The remainder of content is discarded.

```
./langid_Kafka.py
```

7.8. Rumour classification

Having enriched, event-clustered, language-specific text, the goal is to identify candidate rumours. This is done by placing messages into the class of either support, deny, query or comment (SDQC). We implement this using a Gaussian process model, implemented in Gpy, and using the technology from D2.4 and WP4.[15]

The tool has a number of Python dependencies which require Python version 2.7 and certain platform-specific tweaks; it is deployed on Linux.

Content is kept in the private Bitbucket repository mlukasik/Kafka_pheme and replicated in the GATE SALE Pheme repository.

Run:

```
./Kafka_consumer_misinformation.py TOPIC1 TOPIC2 TRAININGDATAPATH
MODELPATH
```

to read content JSON from TOPIC1 and output resulting content JSON to TOPIC2. Recommended training data and models are
`data/twoPHEME_datasets_as_events_041015.csv`
and
`results/store_models_test/BROWNGPjoinedfeaturesPooledLIN0.pick`.
The output parameter added is `pheme_sdqc` which contains both the predict label and also confidence.

15 Classifying Tweet Level Judgements of Rumours in Social Media. Lukasik, Cohn and Bontcheva, 2015. Proc. EMNLP

7.9. Cross-Media and Cross-Language Linking

The component "Cross-Media and Cross-Language Linking" has been developed in the context of Task 3.1, and has been described in detail in D3.1 "Cross-Media and Cross-Language Linking Algorithm", as well as in two workshop papers[16] [17].

The core of the algorithm is language-independent string alignment between social media texts (in our first prototype, Twitter posts) and user-linked web documents (i.e., classical online media articles). The algorithm can additionally connect to media repositories for further cross-media access. In the currently implemented version, the algorithm accesses a third-party web service, EventRegistry[18], a product of the FP7 Project "Xlike"[19]. Usability evaluation is planned in cooperation with WP8 (Digital Journalism Use Case) partners in Y3.

The German pre-processing pipe-line is now operational, and the steps needed to transform classical text processing software to be applicable to social media documents (in the current version, Twitter text) have been described in D2.2 "Linguistic Pre-processing Tools and Ontological Models of Rumours and Phemes". Recent work has been dedicated to integrating this component in the Pheme-Kafka pipeline, which is described in the current document. The German processing tools also benefit a planned updated version Cross-Media algorithm, via lemmatisation of both the social media and the online media texts, which results in improved string alignment.

7.10. Dashboard API

To foster collaboration and leverage synergies between Pheme and the ASAP FP7 project (www.asap-fp7.eu), the data interchange between the dashboard and other Pheme work packages is based on an extended version of the webLyzard API published as part of the ASAP Deliverable 6.2. From the webLyzard API components, the following are relevant in the context of Pheme:

- Document API – ingests unstructured data; for the initial prototype, this includes English-language news media, to be be extended with social media content from T6.1 The main API objects are Documents, Sentences and Annotations – the latter will be provided by WP2-3, using Pheme-specific API extensions to support the required metadata elements.
- Search API – returns a set of query results in the form of unstructured text documents. The main object is Query.
- Embeddable Visualization API – represents a standardized way to integrate individual visualizations in third-party applications (as compared to using the full dashboard), which will allow us to use dashboard components in the digital journalism showcase (T8.3) as well. The main object is Visualization, which is typically rendered based on the results of a Query.

[16] Piroska Lendvai, Thierry Declerck (2015a). Similarity-Based Cross-Media Retrieval for Events. In: Ralph Bergmann, Sebastian Görg, Gilbert Müller (eds.): Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB, Trier, Germany, CEURS, 10/2015

[17] Piroska Lendvai, Thierry Declerck (2015b). Bootstrapped Extraction of Index Terms from Normalized User-Generated Content. In: Proc. of NLP 4 CMC 2015: 2nd Workshop on Natural Language Processing for Computer-Mediated Communication / Social Media. German Society for Computational Linguistics & Language Technology.

[18] https://github.com/gregorleban/EventRegistry

[19] http://www.xlike.org

The use of the webLyzard API not only avoids redundant efforts, but also supports joint exploitation efforts; specific opportunities include ASAP's Visualization-as-a-Service (VaaS) approach, where Pheme components such as the cluster map and the keyword graph could be offered as part of an integrated framework.

See deliverable D5.2.1 for more detailed and technical description of the component.

7.11. Concept Suggester

This component serves as a mediator between Kafka stream on one side and Rumour classification (RC) and Concept Extraction Service (CES, performing Named Entity Recognition) on the other side. Concept Suggester reads periodically unprocessed tweets from specified Kafka topic (e.g. "pheme_en") and redirects the tweet for processing to CES and Rumour classification. Next, the response from both services is annotated to the source tweet and written back into Kafka in another topic (e.g. "pheme_en_processed).

Input format

1. Source tweet in JSON format:


```
{
  "pheme_source": {
     "createdAt": 1446838154000,
     "favouriteCount": 0,
     "hashTags": "opKKK",
     "inReplyToId": "-1",
     "originalTweetId": "662649983322234883",
     "rawJson": {...},
     "retweetCount": 0,
     "source": "<a href=\"https://commun.it\" rel=\"nofollow\">Commun.it</a>",
     "text": "RT @RawStory: Child molesting KKK chief: \u2018I will always hate the Jew \u2014 and please vote for Trump!\u2019 https://t.co/YkXadavCai #opKKK https://t\u2026",
     "userScreenName": "linksteroh"
  }
}
```


Output format

The Concept Suggestor combines both RC and CES outputs: the rumour probability (see next section) and a list of named entities (Person, Location, Organization). Output –


```
{
  "pheme_source": {
     "isRumour": false,
     "originalTweetId": "",
     "inReplyToId": "",
     "createdAt": "",
     "favouriteCount": "",
     "userScreenName": "",
     "hashTags": "",
     "rawJson": {...},
     "created_at": "",
     "lang": "",
     "text": "",
     "source": ""
  },
  "mentions": [
     {
        "endOffset": 0,
        "startOffset": 0,
        "features": {
```

```
            "isGenerated": "",
            "class": "",
            "relevanceScore": 0,
            "confidence": 0,
            "inst": ""
         },
         "type": "",
         "name": ""
      }
   ],
   "rumour": false,
   "rumourCoefficient": 0,
   "lang": "",
   "text": "",
   "source": ""
}
```

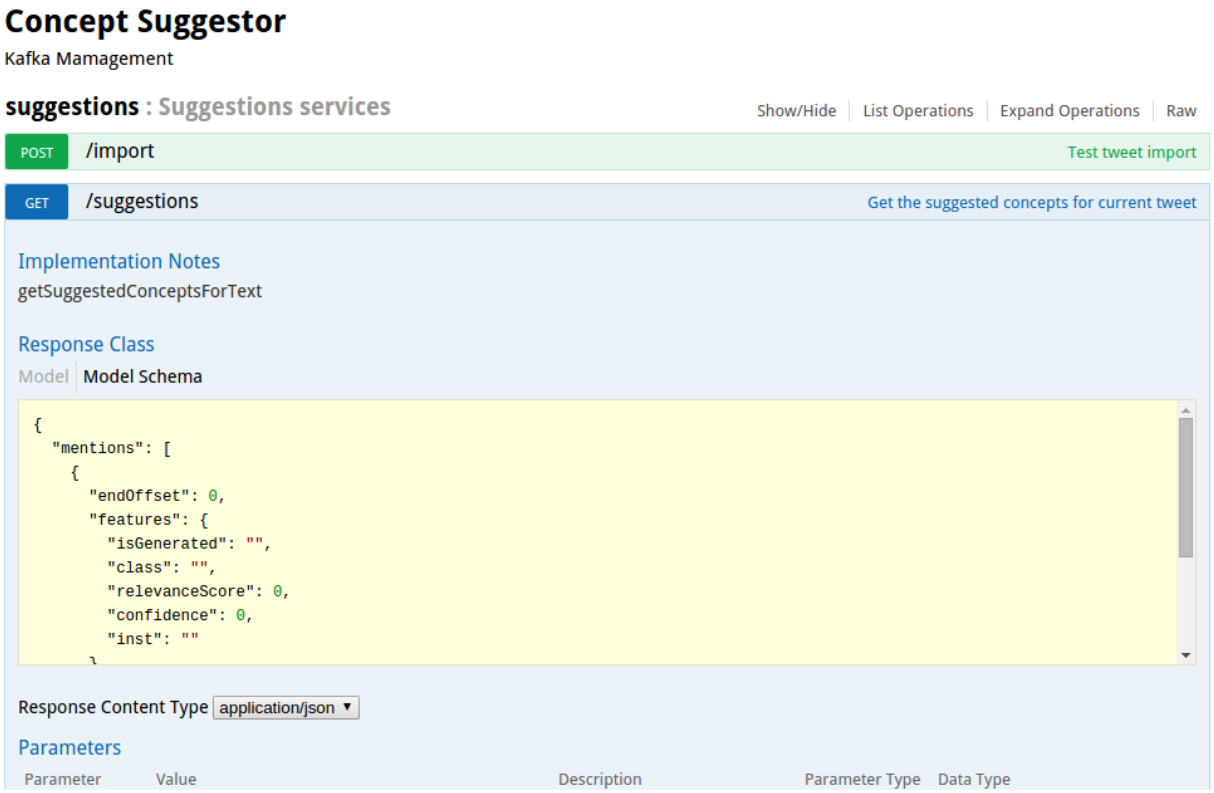



Figure 21. Screenshot of the Concept Suggester API

7.12. Rumour classificator

The Rumour Classificator (RC) is a machine learning model (classification tree) that predicts the probability that a tweet is a rumour. The model is based on the following features: *sourceId*, *number of followers* of the user, *number of followees* of the user, *number of statuses* of the user, *user ID*, the list of *URLs referenced* in the tweet and *the text of the tweet*. As a result the service gives back the rumour coefficient of the specific tweet. The RC returns a probability, which quantifies the likelihood that the tweet is a rumour. Therefore, it is a real value between 0 and 1; a value of 1 means that the tweet is a rumour with certainty and a value of 0 means it is not a rumour, with certainty.

Input format:

Analysed fields from rawJson: **source.id, followers, followees, statuses, user.id, links**

Output format:

The component produces a score which is attached

```
{
 "score": 0.72
}
```

7.13. Concept Extraction Service

This service extracts from the text of the tweet all named entities, with their positions in the text, relevance score, prediction confidence, name and type (Person, Location, Organization). Input format for the service:

- text of a tweet (for example *Obama is in the White house*)
- language of the tweet as abbreviation (for example *en*, *bg* or *de*)

The output format for the service is an array of mentions as follows:

```
{
 "mentions": [
   {
     "name": "Obama",
     "startOffset": 0,
     "endOffset": 5,
     "type": "Keyphrase",
     "features": {
       "class": "http://ontology.ontotext.com/taxonomy/Keyphrase",
       "relevanceScore": 1,
       "confidence": 0.5,
       "inst": "http://data.ontotext.com/publishing/topic/Obama",
       "isGenerated": "true"
     }
   },
   {
     "name": "White house",
     "startOffset": 16,
     "endOffset": 27,
     "type": "Location",
     "features": {
       "class": "http://ontology.ontotext.com/taxonomy/Location",
       "relevanceScore": 0.40740740740740744,
       "confidence": 0.9855316909406875,
       "inst": "http://ontology.ontotext.com/resource/tsk4xg5bncow",
       "isGenerated": "false"
     }
   }
 ]
}
```

7.14. Suggestions consumer

This service reads all the tweets from the Kafka topics with already processed tweets (for example "*pheme_en_processed"*) and send to the *graph-api*. This service is actually a consumer and mediator between Kafka and *graph-api*.

Input format - same JSON format for the tweets as in the other services.

```
{
  "pheme_source": {
    "isRumour": false,
    "originalTweetId": "",
    "inReplyToId": "",
    "createdAt": "",
    "favouriteCount": "",
    "userScreenName": "",
    "hashTags": "",
    "rawJson": {
      "is_quote_status": false,
      "id": 0,
      "contributors": "",
      "coordinates": "",
      "entities": {...},
      "extended_entities": {... },
      "favourite_count": 0,
      "favorited": false,
      "filter_level": "",
      "geo": "",
      "id_str": "",
      "in_reply_to_screen_name": "",
      "in_reply_to_status_id": "",
      "in_reply_to_status_id_str": "",
      "in_reply_to_user_id": "",
      "in_reply_to_user_id_str": "",
      "quoteStatus": false,
      "place": "",
      "possibly_sensitive": false,
      "retweet_count": 0,
      "retweeted": false,
      "retweeted_status": "RawJson",
      "timestamp_ms": "",
      "truncated": false,
      "user": {
        "id": "",
        "screen_name": "",
        "name": "",
        "image_url": "",
        "followers_count": "",
        "friends_count": "",
        "statuses_count": 0,
        "verified": false,
        "location": ""
      },
      "created_at": "",
      "lang": "",
      "text": "",
      "source": ""
    },
    "mentions": [
      {
        "endOffset": 0,
        "startOffset": 0,
        "features": {
          "isGenerated": "",
          "class": "",
          "relevanceScore": 0,
          "confidence": 0,
          "inst": ""
        },
        "type": "",
        "name": ""
      }
```

```
    ],
    "rumour": false,
    "rumourCoefficient": 0,
    "lang": "",
    "text": "",
    "source": ""
  }
}
```


7.15. Graph-api

This service transforms the incoming tweets in JSON format from the *Suggestions controller* into RDF format and imports them into the GraphDB.

Input format - same JSON format for the tweets as in the other services:


```
{
  "pheme_source": {
    "isRumour": false,
    "originalTweetId": "",
    "inReplyToId": "",
    "createdAt": "",
    "favouriteCount": "",
    "userScreenName": "",
    "hashTags": "",
    "rawJson": {
      "is_quote_status": false,
      "id": 0,
      "contributors": "",
      "coordinates": "",
      "entities": {...},
      "extended_entities": {... },
      "favourite_count": 0,
      "favorited": false,
      "filter_level": "",
      "geo": "",
      "id_str": "",
      "in_reply_to_screen_name": "",
      "in_reply_to_status_id": "",
      "in_reply_to_status_id_str": "",
      "in_reply_to_user_id": "",
      "in_reply_to_user_id_str": "",
      "quoteStatus": false,
      "place": "",
      "possibly_sensitive": false,
      "retweet_count": 0,
      "retweeted": false,
      "retweeted_status": "RawJson",
      "timestamp_ms": "",
      "truncated": false,
      "user": {
        "id": "",
        "screen_name": "",
        "name": "",
        "image_url": "",
        "followers_count": "",
        "friends_count": "",
        "statuses_count": 0,
        "verified": false,
        "location": ""
      },
      "created_at": "",
      "lang": "",
      "text": "",
      "source": ""
    },
    "mentions": [
```

```
        {
          "endOffset": 0,
          "startOffset": 0,
          "features": {
            "isGenerated": "",
            "class": "",
            "relevanceScore": 0,
            "confidence": 0,
            "inst": ""
          },
          "type": "",
          "name": ""
        }
      ],
      "rumour": false,
      "rumourCoefficient": 0,
      "lang": "",
      "text": "",
      "source": ""
  }
}
```


The output format is in RDF form. For example:


```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.pheme.eu/ontology/pheme#tweet-673747983727234873">
    <has_container xmlns="http://rdfs.org/sioc/ns#"
rdf:resource="http://www.pheme.eu/ontology/pheme#thread-673747983727234873"/>
    <has_creator xmlns="http://rdfs.org/sioc/ns#"
rdf:resource="http://www.pheme.eu/ontology/pheme#user-18825344"/>
    <containsMention xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:resource="http://www.pheme.eu/resources/pheme/11682999fb6e44ed91726acab475aae2"/>
    <containsMention xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:resource="http://www.pheme.eu/resources/pheme/1bbb3a0ddf654729be8bd9b6ae667c48"/>
    <containsMention xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:resource="http://www.pheme.eu/resources/pheme/534dac7ec0d24c52884b262fdbfabc39"/>
    <containsMention xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:resource="http://www.pheme.eu/resources/pheme/6e05b1e186b94589b56273fda884a22d"/>
    <containsMention xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:resource="http://www.pheme.eu/resources/pheme/8d0040ce6b5f4f8d85786bb21b492dbb"/>
    <containsMention xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:resource="http://www.pheme.eu/resources/pheme/94973d7339fe49d4b69fdc4f4c9466c7"/>
    <createdAt xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2016-06-25T10:02:45.000+03:00</createdAt>
    <hasEvidentiality xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:resource="http://www.pheme.eu/ontology/pheme#url-given"/>
    <rumourCoefficient xmlns="http://www.pheme.eu/ontology/pheme#"
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.49741011460429485</rumourCoefficient>
    <textualContent xmlns="http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#">RT @RawStory:
Child molesting KKK chief: I will always hate the Jew and please vote for Trump! https://t.co/YkXadavCai #opKKK
https://tfdswds</textualContent>
    <rdf:type rdf:resource="http://www.pheme.eu/ontology/pheme#Rumour"/>
    <rdf:type rdf:resource="http://www.pheme.eu/ontology/pheme#SourceTweet"/>
    <rdf:type rdf:resource="http://www.pheme.eu/ontology/pheme#Tweet"/>
  </rdf:Description>
</rdf:RDF>
```