RENDER
FP7-ICT-2009-5
Contract no.: 257790
www.render-project.eu

RENDER

Public Annex to Deliverable D6.4.4

RENDER Deployment and Integration Requirements

Editor:	Javier Caminero, Telefónica
Author(s):	Javier Caminero & Mari Carmen Rodríguez Gancedo, Telefónica; Delia Rusu, JSI; Daniele Pighin, Google; Maurice Grinberg, Ontotext; Angelika Mühlbauer, Wikimedia; Ioan Toma, UIBK; Felix Leif Keppmann, KIT
Deliverable Nature:	Report (R)
Dissemination Level: (Confidentiality)	Public (PU)
Contractual Delivery Date:	30 September 2013
Actual Delivery Date:	30 September 2013
Suggested Readers:	RENDER software modules developers and architects, everyone interested in rebuild the RENDER project scenario.
Version:	1.0
Keywords:	Integration, deployment, exploitation plan, architecture, API, modules

Disclaimer

Full Project Title:	RENDER – Reflecting Knowledge Diversity
Short Project Title:	RENDER
Number and Title of Work package:	WP6 - Dissemination, Exploitation, and Community Building
Document Title:	Public Annex to Deliverable D6.4.4 RENDER Deployment and Integration Requirements
Editor (Name, Affiliation)	Javier Caminero, Telefónica
Work package Leader (Name, affiliation)	Andreas Thalhammer, UIBK

Executive Summary

This document specifies the integration requirements of every software module involved in the RENDER project. The installation and integration APIs, and tools of each module are described in order to rebuild the RENDER scenario once the project ends. The following tools are described in this document: Diversity Mining Toolkit (Enrycher), OWLIM Data Repository, Telefónica Opinion Mining Toolkit, TwiDiViz Tweet Visualization, Diversity Enricher Drupal Extension, Diversity-Aware Ranking Service, Wikimedia Toolkit, DiversiNews and TopicSum Summarizer.

Table of Contents

List of Figures

1 Introduction

Looking ahead to the end of the RENDER Project in 2013, and as part of the exploitation plan, the following documentation is intended to future users, in order to make them capable to deploy all the essential modules and to setup the project scenario.

The project scenario will be divided into different modules according to the architecture. The setup and use of each module must be self-reliant. In order to achieve this, each module should fulfil some specific features according to its functionalities and its goals.

2 Diversity Mining Toolkit (Enrycher)

The Diversity Mining Services (Enrycher) expose the functionality of the Fact and Opinion Mining toolkits via a RESTful API. Enrycher is a service-oriented system, providing shallow as well as deep text processing functionality at the text document level. The shallow text processing functionality includes topic and keyword detection and named entity extraction, while the deep text processing functionality provides named entity resolution with respect to existing Linked datasets, named entity merging, word sense disambiguation into WordNet and assertion extraction, by identifying subject – predicate – object sentence elements together with their modifiers (adjectives, adverbs) and negations, sentiment detection.

Web front-end application
The Diversity Mining Toolkit (Enrycher) has an accompanying Web interface for users to test the results directly: http://enrycher.ijs.si/. Additionally, the topic and sentiment models that are employed by Enrycher can be validated and updated via a dedicated Web interface which exposes the model training functionality developed within the Opinion Mining Toolkit: http://aidemo.ijs.si/render/index.html. Having a dedicated Web interface for model training and validation allows for this package to be optional, as certain production deployments could not require it.

2.1 Installation and Deployment

The Enrycher Java services have been developed under the Windows operating system, but can also be installed under Unix. The system prerequisites are an operating system running Java. To run the system as a service Apache Tomcat should be installed and configured.

The Enrycher Java services source code and API is provided in GitHub, together with installation instructions. One can either call specific Enrycher methods via the available Enrycher API, or build the Enrycher web application WAR (**W**eb application **Ar**chive) file – a sample web.xml file can be found in Annex B.

The deployment specifications on the Enrycher system are depicted in Figure 1. A server running Apache Tomcat hosts the Enrycher application, and optionally the Interactive Modelling Toolkit. The Modelling Toolkit is only required if new sentiment models are required. Otherwise existing sentiment models can be used.

The main deployment modules are the Opinion Mining toolkit for sentiment detection, and the Fact Mining Toolkit for annotating text at different granularity levels.

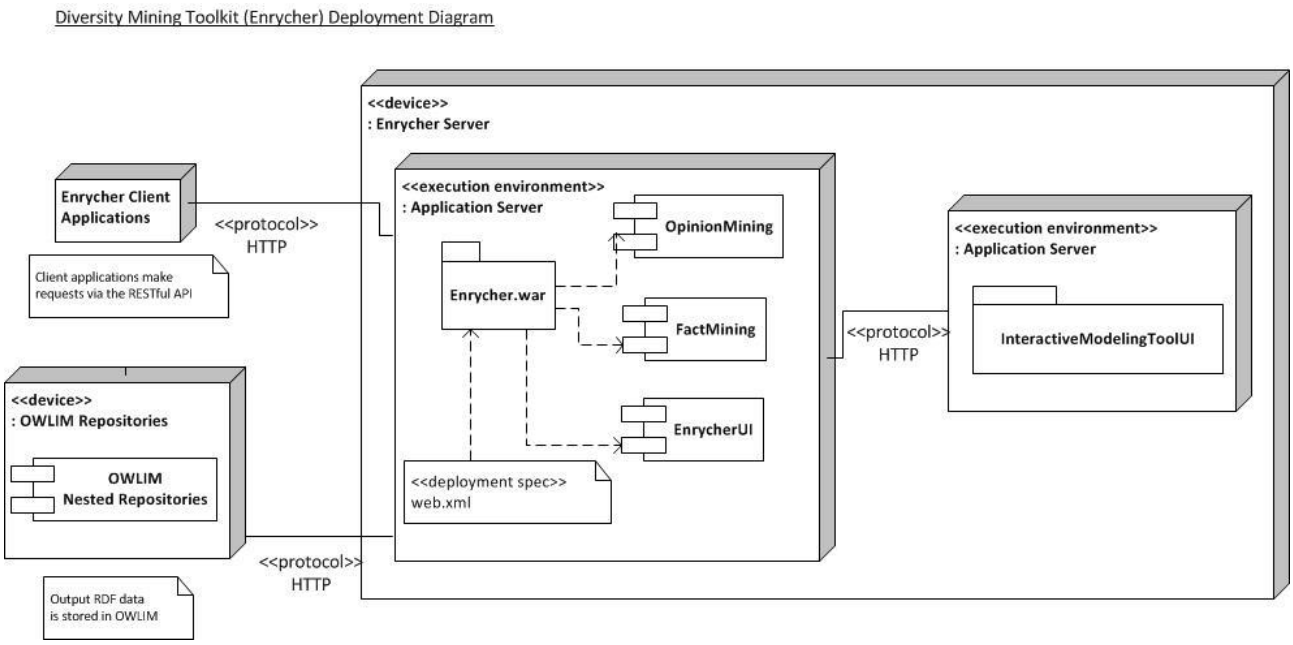


Figure 1: Diversity Mining Toolkit (Enrycher) Deployment Diagram.

2.2 Configuration

The module configuration is done via configuration files. These specify the training models used by the system, as well as other input and output options which refer to the data format and components used in the processing pipeline. The training models are created via the Interactive Modeling Tool which is configured separately from the main Enrycher module.

It is recommended to have a folder containing all resources required by the system:

- Pre-processing dependency files: sentence splitting, tokenization, part-of-speech-tagging, etc.
- Named entity resolution dependency files

Moreover, a config.xml file with the list of dependencies and their location should be created (see an example in Annex A).

To set Enrycher to run as a service, the web.xml file should be configured to include all pipeline components that are to be accessible. An example web.xml for the Enrycher Java services can be found in Annex B.

2.3 Administration

Apache Tomcat[1] should be installed, configured and started for the Enrycher Java services.

2.4 Integration with Other Modules

In order to integrate Enrycher with the rest of the system, Enrycher is deployed as a low-coupled component which provides a RESTful API for managing input and output data. The results are represented using the RDF format and can be retrieved as such or stored in OWLIM repositories.

The system exposes several services, depending on the requested functionality. The service format is:

[1] http://tomcat.apache.org/

/[language]/run-[sentiment-model]-[document-type].rdf

where:

language – is the language of the document

sentiment-model – is the model used by the sentiment classifiers

document-type – is the type of the document: post of news article.

E.g. the service

http://aidemo.ijs.si/enrycher-render/en/run-newssenti-newsarticle.rdf

is configured for English, using a newssenti model for news articles.

Currently, there are three sentiment models available:

- newssenti - sentiment model for news
- socialsenti - sentiment model for social media data
- gensenti - generic sentiment model

2.5 Maintenance Details

Depending on the input documents, some might cause the Enrycher system to unexpectedly stop working. We therefore recommend a website monitoring system that sends an alert when the system is unresponsive, so that it can be restarted.

3 OWLIM Data Repository

RENDER basic data layer is a reason-able view, based on FactForge (http://factforge.net) the public service developed and maintained by Ontotext. It is a compound dataset of some of the most popular datasets from the Linked Open Data Cloud:

- DBPedia 3.8
- Freebase
- CIA Factbook
- New York Times
- Wordnet 3.0
- Lingvoj
- Lexvo
- Geonames
- Musicbrainz

The concept of a Reference Knowledge Stack has been introduced in RENDER project. It refers to defining a unification ontology, which is linked to the ontologies of FactForge datasets, and serves as a common entry point to FactForge's data, making querying, accessing, managing and navigating this wealth of data more efficient and manageable.

RENDER secondary data layer consist of two datasets, loaded in two repositories, nested into RENDER basic data layer RENDER news and tweets.

3.1 Installation and Deployment

A new version of FactForge and RENDER basic data layer, were released in 2013, loaded in OWLIM-SE 5.3. RENDER basic data layer (http://render.ontotext.com) counts 1.7 billion explicit statements; 5.3 billion retrievable statements available after inference and owl:sameAs expansion which include 1.5 billion inferred statements. Full materialization with respect to OWL-Horst optimized [**Fehler! Verweisquelle konnte nicht gefunden werden.**] is performed at the time of data loading. The OWL-Horst optimized inference forward chaining rules are part of the distribution of OWLIM-SE. They allow for defining constraints, defining rules to handle types, transitive relations. The standard OWLIM-SE OWL-Horst optimized rule set has been augmented with additional rules enriching the datasets with instances which cover structural differences in their schemata. The additional rules cover mappings of LOD schemata to the reference layer ontologies of the Reference Knowledge Stack. The datasets combined in FactForge are: DBPedia, Freebase, Geonames, CIA World Factbook, Lingvoj, MusicBrainz, WordNet, New York Times. Only the unification ontology is loaded into the repository. The ontologies of LOD datasets are not, as they would only generate additional redundant statements, and the make the inferencing process heavier, having to account for them as well.

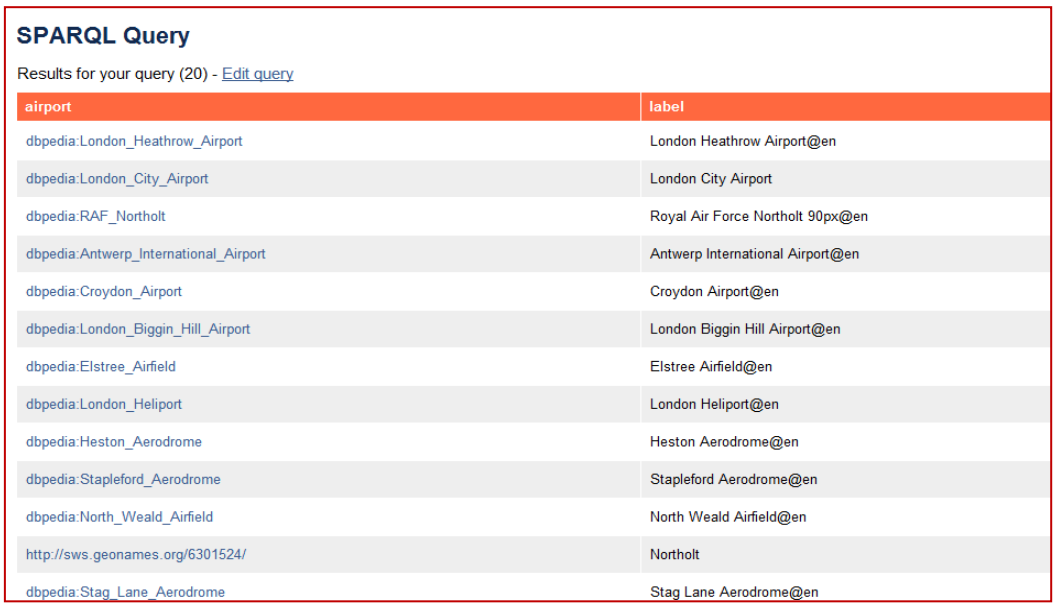
FactForge provides with several methods to explore the combined dataset that exploits some of the advanced features of OWLIM-SE. Firstly, 'RDF Search and Explore' keywords with a real-time auto-suggest feature ordered by 'RDF Rank' (similar to Google's Page Rank). Secondly, a SPARQL page allows users to write their own queries with clickable options to add each of the known namespaces. Lastly, a graphical search facility called 'RelFinder' that discovers paths between selected nodes. This is a computationally intensive activity and the results are displayed and updated dynamically during each iteration. The resulting graph can be reshaped by the user with simple click and drag operations.

The current version of RENDER basic data layer at http://render.ontotext.com/sparql supports SPARQL 1.1, the latest and most powerful edition of SPARQL language. This allows making use of all new SPARQL 1.1 and OWLIM 5.3 features when querying. These are the most prominent feature examples:

- GROUP BY, min
  - Minimal and maximal population counts of European countries
- Literal index over dates
  - World governors in office between 1980 and 2005
- Literal index over digits
  - European countries with population above 20 MLN
- Geospatial index
  - Show the distance from London of airports located at most 50 miles away from it

The exemplary queries are provided at the SPARQL end point and can be executed at any time.

The SPARQL end point allows connecting to visualization services, and produced different kinds of visualizations of the query results, Figure 2 below:



SPARQL Query

Results for your query (20) - Edit query

airport	label
dbpedia:London_Heathrow_Airport	London Heathrow Airport@en
dbpedia:London_City_Airport	London City Airport
dbpedia:RAF_Northolt	Royal Air Force Northolt 90px@en
dbpedia:Antwerp_International_Airport	Antwerp International Airport@en
dbpedia:Croydon_Airport	Croydon Airport@en
dbpedia:London_Biggin_Hill_Airport	London Biggin Hill Airport@en
dbpedia:Elstree_Airfield	Elstree Airfield@en
dbpedia:London_Heliport	London Heliport@en
dbpedia:Heston_Aerodrome	Heston Aerodrome@en
dbpedia:Stapleford_Aerodrome	Stapleford Aerodrome@en
dbpedia:North_Weald_Airfield	North Weald Airfield@en
http://sws.geonames.org/6301524/	Northolt
dbpedia:Stag_Lane_Aerodrome	Stag Lane Aerodrome@en

Figure 2: SPARQL Query Results Visualization

In other to deploy the RENDER repository it is needed a server machine with the Tomcats TCP ports accessible from the Internet:

- http://render.ontotext.com
- http://rendertweets.ontotext.com
- http://rendernewss.ontotext.com

These are actually three Tomcats serving our RDF browsing tool Forest.
Additionally we are deploying forth Tomcat serving our OWLIM v.5.3 RDF repository

3.2 Configuration

The inference strategy in OWLIM, the repository of RENDER basic data layer, is one of materialization based on R-Entailment as defined by the Horst, where Datalog like rules with inequality constraints operate directly on a single ternary relation that represents all triples. In addition, free variables in rule heads are treated as blank nodes. Materialization involves computing all the entailed statements at load time. While

this introduces additional reasoning cost when loading statements into a repository, the desirable consequence is that query evaluation can proceed extremely quickly. Several standard rule sets are included in all editions of OWLIM and these include:

empty – no inference;

rdfs[2] – RDFS semantics using rule entailment, but without data-type reasoning, i.e. without the literal generalization and related rules;

owl-horst – equivalent to pD*, again without data-type reasoning;

owl-max – RDFS and OWL-Lite (that can be captured in rules);

owl2-ql – a fragment of OWL2 Full based on DL-Lite$_R$, a variant of DL-Lite that does not require the unique name assumption;

owl2-rl[3] – the OWL2 RL profile, a fragment of OWL2 Full amenable to implementation on rule-engines, but without data-type reasoning.

In addition to the standard semantics, user-defined rule-sets can be used. In this case the user provides the full pathname to a custom rule file that contains definitions of axiomatic triples, rules and consistency checks. For ease of use, the rule files for the standard rule-sets are included in the distribution and users can modify or extend these for their specific purposes.

Consistency checks are used to ensure that the data model is in a consistent state and are applied whenever an update transaction is committed, for example to ensure that `owl:Nothing` has no members or that no pair of individuals have both `owl:sameAs` and `owl:differentFrom` relationships.

During loading, all inferred statements are materialized, except those generated as a result of the semantics of `owl:sameAs`. OWLIM-SE uses special data structures to maintain equivalence classes and uses the URI of the first asserted resource in each equivalence class in the statement indices. This allows for the correct expansion of results during query-answering while keeping the index sizes manageable. This technique has the further advantage that it can be switched off during query answering in order to limit the number of 'duplicate' results.

Except for the inference rules that serve the reference knowledge stack and allow to cover the structural mismatches in the conceptualizations of different FactForge ontologies, which count about 40, the final data collection of RENDER includes new inference rules, specifically crafted to provide consistency checks based on disjoint class definitions, which allow to capture conceptual contradictions, as for example, capturing that it is impossible for one entity to be both of type City and of type Book.

They are of the following form:

```
X owl:sameAs Y
X rdf:type W
Y rdf:type Z
W owl:disjoint Z
----------------
X ptop:error Y
```

This approach is the beginning of the creation of LOD curation methodology over explicit and implicit statements in an open world reasoning context.

[2] http://www.w3.org/TR/rdf-schema/
[3] http://www.w3.org/TR/owl2-profiles/

3.3 Administration

Except for the datasets, described in the rest of the sections and PROTON, FactForge and RENDER basic data layer includes several popular schemata which are configuring our RDF repository. These are:

- DCMI Metadata Terms (Dublin Core - DC)

Dublin core is a popular metadata initiative providing a model adapted to represent library information. It consists of 15 properties. Dublin core has been very widely used by many library institutions, and . Dublin Core is part of a larger initiative DCMI, which aims at including these 15 properties into a larger system of vocabularies allowing to model richer library environments.

- SKOS (Simple Knowledge Organization System)

SKOS provides a standard way to produce knowledge systems in RDF, by providing the standards for building thesauri, classification schemes, taxonomies, subject heading sections, etc. all these connections to the framework of the Semantic Web.

- RSS

RSS stands for really simple syndication. It is a web content syndication format. It complies with W3C recommendations of XML representation. It includes information about the authorship and the date of publication of a given news feed.

- FOAF

FOAF, Friend of a Friend, is a model allowing to describe person data and relationships in machine readable form, abiding the principles of the Semantic Web. Its last version of 2010 is part of RENDER basic data layer.

- Freebase OWL ontology

Ontotext is developing a Freebase OWL ontology, based on the Freebase schema. This ontology will allow to produce inferred statements out of Freebase entries that are produced as a result of OWL reasoning.

3.4 Integration with Other Modules

The News dataset provided by JSI is a collection of news articles crawled from the Google News web site in a specific period. The collection contains articles which are also clustered into stories. The articles are stripped of HTML markup and chrome (navigation, headers, footers etc.), then enriched with named entity detection and disambiguation algorithms and with full constituency parse trees. All the articles in this collection are in English, although they are gathered from publishers located all over the world.

Google news articles were analyzed according to the following RDF model, c.f. Figure 3 below. It includes diversity information, topic information, reference to an entity from the LOD cloud, document identification.

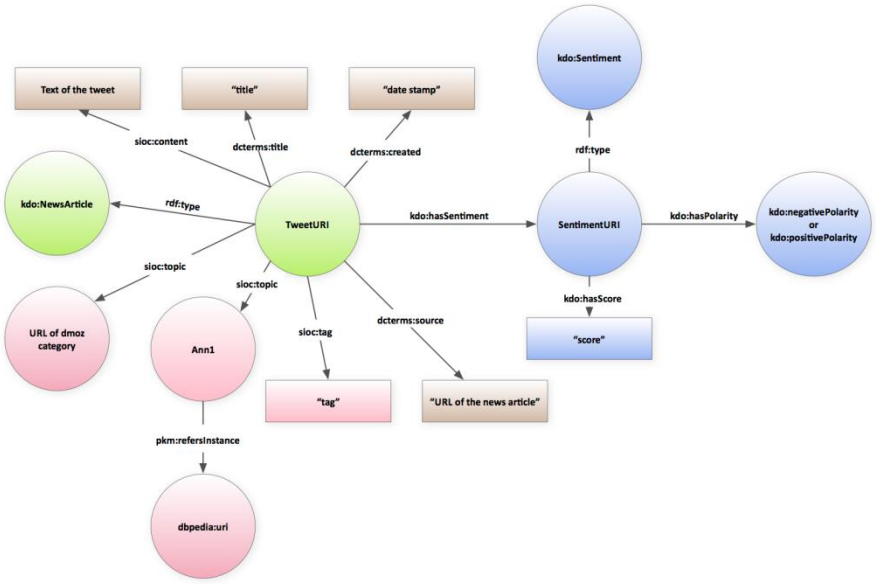


Figure 3: RDF Model

This nested repository is available at http://rendernews.ontotext.com

The Twitter dataset collected by Telefonica is processed using the Diversity Mining Services (Enrycher) and its RDF representation is stored in the OWLIM repositories.
The processed data comprises annotated tweets dated as follows:

- Monthly datasets including May, June, July, August 2012

The RDF model of the tweets is presented in Figure 4 below. It includes diversity information, topic information, reference to an entity from the LOD cloud, author account information, and tweet identification information.

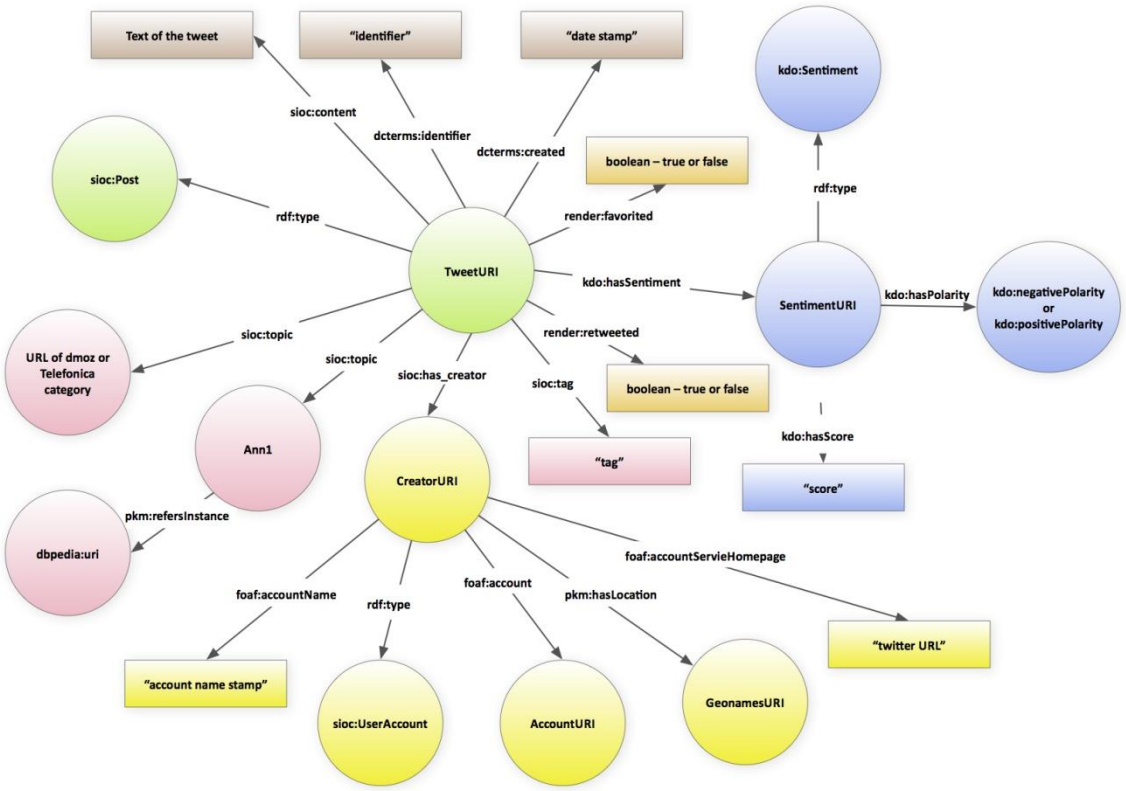


Figure 4: RDF Model for the Tweets

This nested repository is available at: http://rendertweets.ontotext.com

3.5 Maintenance Details

The maintenance of the OWLIM RENDER repository includes its nested children News and Tweets. Their size and RAM usage definitely has to be considered. Together with Forest UI browsers mentioned in the previous section are maintained through special scripts checking their status.

4 Telefónica Opinion Mining Toolkit

The diversity Information Extensions for Telefónica Tools or referred in a simpler way as Telefónica Opinion Mining Tool (T-OMT), provides different means to search and visualize certain topics and track their evolution, having as final goal the assistance in corporate decision making.

4.1 Installation and Deployment

The Telefónica's Opinion Mining Tool component is a Java Web application. It should run on a dedicated machine, which will be called "server machine". There is no mandatory Operating System to be installed on the server machine, although a Unix distribution like Ubuntu is preferable. Details of how to install the Operating System is out of the scope of this deliverable. Java framework, Tomcat servlet container, Mongo DB and T-OMT itself must be installed in the server machine and are described in following sub-sections. Clients and users of T-OMT must access the application using a Web Browser installed in their machines. This tool was tested on Mozilla Firefox 21.0 and Google Chrome 27.0. In addition, if clients want to make use of Pivot Viewer, Silverlight technology (http://www.microsoft.com/silverlight/) must be installed in their browsers.

4.1.1 Java

T-OMT is developed in Java (http://www.java.com), so Java framework must be installed in the server machine. The installation process of Java is described in Java home page, and binaries for different Operating Systems are available. The minimum Java version to install is Java 7.

4.1.2 Tomcat

A servlet container must be installed in the server machine. T-OMT runs over Tomcat (http://tomcat.apache.org/), so Tomcat must be installed in the server machine. The minimum Tomcat version to install is Tomcat 7. There are two ways to install Tomcat (described in Tomcat home page): install Tomcat as a service/deamon or download a zipped file. Both methods are valid for T-OMT. Please follow installation instructions of Tomcat in Tomcat home page.

4.1.3 Mongo DB

T-OMT uses Mongo DB (http://www.mongodb.org/) as database to persist and retrieve the reports. Mongo DB should be installed as a service/deamon in the server machine. Version 2.4.4 of mongo DB is used in T-OMT. Installation instructions of how to install Mongo DB are available in Mongo home page. In production environments the 64 bits version of Mongo DB is preferable.

4.1.4 T-OMT

The Telefónica's Opinion Mining Tool is distributed as a WAR file. In order to install T-OMT, simply put the WAR file in Tomcat's 'webapps' folder.

In other to deploy the Telefonica's Opinion Mining Tool it is needed a server machine with the Tomcat's TCP port accessible from the Internet. So, if Tomcat's TCP port is 8080 (default), clients and users must point their browsers to http://ip-of-server-machine:8080/TOMT/ to see and interact with T-OMT.

4.2 Configuration

There are several things that can be configured in the server machine, regarding the Operating System, Java, Tomcat and Mongo DB. These configurations are out of the scope of this deliverable and we will assume that all of the configuration is the default configuration.
The main configuration parameter of Tomcat is the TCP port in which it is listening for incoming connections. We will assume that this port is 8080 (default).

The main configuration parameter of Mongo DB is the port, in which it is listening for incoming connections. We will assume that this port is 27017 (default).
If the default configuration of Tomcat and Mongo DB is maintained, there is nothing to configure in T-OMT.
Clients and users of T-OMT must enable Javascript in their browsers. Normally, Javascript is enabled by default. If not, follow the instructions of the web browser in order to enable it.

4.3 Administration

The main administration aspects of T-OMT are to start and stop the application. To start the application, Tomcat must be started. As there were two ways of installing Tomcat, there are two ways of starting it.
If Tomcat was installed as a service/deamon, that service or deamon must be started in order to start Tomcat. Please follow instructions of the Operating System to see how to start a service or deamon.
If Tomcat was downloaded as a zipped file, simply execute 'startup.sh' or 'startup.bat' (depending of the Operating System) to start Tomcat.
To stop the application, Tomcat must be stopped. Again, if Tomcat was installed as a service/deamon, that service/deamon must be stopped following the Operating System instructions. If Tomcat was downloaded as a zipped file, simply execute 'shutdown.sh' or 'shutdown.bat'.

4.4 Integration with Other Modules

T-OMT makes use of Ontotext data repository. HTTP connection between the T-OMT server machine and Ontotext server machine must be available. T-OMT executes HTTP GET queries against Ontotext data repository in order to retrieve information to compose the reports presented to the user.
In addition, Pivot Viewer UI is integrated with T-OMT UI as a separate Web page. Pivot Viewer makes use of Silverlight technology of the client's browser.

4.5 Maintenance Details

The maintenance of T-OMT is very simple. Just check that the application is up and available through Internet (if not, check that Tomcat is running) and that Mongo DB has free space to keep persisting reports (the 32 bits version of Mongo DB is limited to 2GB of data. In production environments, the 64 bits version is preferable).

5 TwiDiViz Tweet Visualization

TwiDiViz is a web-based application, developed by UIBK, enabling the analysis and visualization of diversity aspects in Twitter data. This tool uses core RENDER technologies i.e. the Knowledge Diversity Ontology (KDO) ontology, the Enrycher service, and the enriched Tweet data store[4] implemented with OWLIM.

5.1 Installation and Deployment

In order to display the visualization of Tweets with TwiDiViz, the Microsoft Silverlight plugin needs to be installed for the respective browser. For Linux, there exists the open source Moonlight[5] solution.

In order to run a own deployment of TwiDiViz one has to download the source code from https://github.com/athalhammer/TwiDiViz, import it in Microsoft Visual Studio 2012 Ultimate Edition, create a XAP file, and deploy it on a Web Server (e.g. Apache).

Currently an example deployment is available at:

http://sti2.at/~simonh/RENDER/KDOVisualization/KDOVisualizationTestPage.html

It visualizes more than 1000 Tweets of the RENDER Twitter dataset at once.

5.2 Configuration

In the source code, several parameters such as the respective SPARQL endpoint, username and passwords, can be adapted. For this, the file “KDOVisualization/MainPage.xaml.cs” has to be edited at the respective sections.

5.3 Administration

The main administration aspects of TwiDiViz are to start and stop the application. To start or stop the application, the web server (e.g. Apache) must be started or stopped. There are no further administrative tasks to perform, while running TwiDiViz.

5.4 Integration with Other Modules

TwiDiViz is integrated with the RENDER Twitter data store implemented with OLWIM. Moreover, it is integrated with the T-OMT UI as a separate Web page. The filter rules can be exchanged with a fixed JSON file format:

```
{
“source” : “...”,
“countries” : [“...”,”...”,...],
 “topics” : [“...”,”...”,...],
“from_date” : “...”,
“to_date” : ”...”,
“query_date” : “...”
}
```

Figure 5: T-OMT Filter Exchange Format

[4] http://rendertweets.ontotext.com/
[5] http://mono-project.com/Moonlight

5.5 Maintenance Details

The maintenance of TwiDiViz is very simple. A simple check whether the application is on-line and available through its Web interface is sufficient.

6 Diversity Enricher Drupal Extension

The Diversity Enricher Drupal extension has been developed as a show case for diversity-enabling technologies. It supports diversity-aware navigation, organization, and presentation of Drupal articles. The show case can be found at http://render-project.eu/drupal.

6.1 Installation and Deployment

The code of Diversity Enricher is available at https://drupal.org/sandbox/sti-innsbruck/1991696. After download, the following steps have to be taken to install the Diverstiy Enricher on a Drupal 7 installation:

The following libraries are required and have to be stored in the “lib” folder.

- rdfapi-php (http://sourceforge.net/projects/rdfapi-php/)
- semsol-arc2 (https://github.com/semsol/arc2)
- phpSesame mod (https://github.com/athalhammer/phpSesame)

The “files[]” paths in the “diversity_enricher.info” file might have to be adapted after all libraries are installed.

The following prerequisites exist for the Diversity Enricher Drupal module:

- Installed Drupal 7
- Installed and running instance of SESAME (http://www.openrdf.org/)
- Running instance of the RENDER Ranking Service (https://github.com/athalhammer/RENDER-ranking-service)
- Running instance of Enrycher (http://enrycher.ijs.si/)
- Installed Pear library (http://pear.php.net/package/HTTP_Request2/redirected). It has to be installed on your server

As like with any other Drupal module, the folder diversity_enricher has to be copied into the “./sites/all/modules” folder of a Drupal installation. For further configuration and administration, please see Sections 6.2 and 6.3.

6.2 Configuration

The following steps have to be undertaken to configure the Diversity Enricher module:

- Copy the Diversity Enricher module folder to your extension folder (e.g. $YOUR_DRUPAL_PATH/sites/all/modules/)
- Activate the Diversity Enricher module (e.g. click on "Modules" in the Admin interface, search "Diversity Enricher" and activate it)
- Configure Diversity Enricher (only possible with Admin or User rights)
- Go to Configuration/DiversityEnricher: Admin in the admin interface
- Fill out the requested fields (all of them are needed to ensure that Diversity Enricher is working properly)
- Go to Structure/Blocks in the admin interface
- Search for Diversity Enricher and place it at "Sidebar Second" (or anywhere else you want to have it displayed)

6.3 Administration

In the administration interface, the following points can be altered:

- Sesame / OWLIM Server URL
- Sesame / OWLIM Repository Name
- Use RENDER Ranking Service (enable and disable)
- Ranking Service URL
- Enrycher Service URL

Moreover, it is possible for administrators to import data from a Sesame / OWLIM store. For this, the Configuration panel has the option: Diversity Enricher: Import.

6.4 Integration with Other Modules

The following other modules of the RENDER project are or may be used with the Diversity Enricher module:

- OWLIM triple store: may be used for storing the diversity data. Can also be used as a data source for importing.
- Enrycher service: used to process the articles written by the Drupal users.
- Diversity-Aware Ranking service: may be used to display related articles at the side panel
- Knowledge Diversity Ontology: used to represent diversity-aware data

6.5 Maintenance Details

As the hosting Drupal 7 installation, the only maintenance required is to keep the Apache2 server with the PHP5 module running.

7 Diversity-Aware Ranking Service

The Diversity-Aware Ranking Service is a system that leverages two clustering algorithms on diversity-enriched datasets. It further comprises a visualization component that exemplifies the combination of different diversity aspects. The deployment of the RENDER ranking service is available at http://ranking.render-project.eu.

7.1 Installation and Deployment

A reference deployment can be found at http://ranking.render-project.eu. For an own deployment, the following steps are necessary:

- Requirements: Java EE 6, GlassFish Server v3, Eclipse
- Setup development environment: checkout latest version of the master branch, in Eclipse install git extension, in Eclipse install GlassFish Server adapter, import project in Eclipse (Projects from Git)
- configure the "endpoint" parameter in WebContent/vis.html and WebContent/tweetVis.html, enter credentials in WebContent/WEB-INF/twitter4j.properties
- Build instructions: export as ranking.war
- Deployment instructions: asadmin> deploy --name ranking ./path-to/ranking.war

7.2 Configuration

The service accepts the following main parameters:

- SPARQL endpoint: the SPARQL endpoint with KDO-compliant information stored
- SPARQL Restrictions: restrictions, such as topics, sentiment, or date on the sioc:Post ?s
- SPARQL Order By: referring to the restrictions a predefined ordering
- Context: selecting only content from a specific named graph
- Rank focus: sentiment or topic

Also, the following special parameters are helpful:

- Force algorithm: prefer a specific algorithm
- Epsilon: manual setting of epsilon
- Gamma: manual setting of gamma
- Sentiment score normalization: normalize sentiment scores

For testing and evaluation purposes also the random element of the MAXIMUM algorithm can be fixed. Also the debug mode can be enabled.

7.3 Administration

The main administration aspects of the Diversity-Aware Ranking Service are to start and stop the application. To start or stop the application, the web server (i.e. Glassfish) must be started or stopped. There are no further administrative tasks to perform, while running the Diversity-Aware Ranking Service.

7.4 Integration with Other Modules

The Diversity Enricher Drupal extension is using the Diversity-Aware Ranking Service for selecting related articles (same cluster). Moreover, the Diversity-Aware Ranking Service might be combined with any OWLIM deployment.

7.5 Maintenance Details

The maintenance of the Diversity-Aware Ranking Service is very simple. A simple check whether the application is on-line and available through its Web interface is sufficient.

8 Wikimedia Toolkit

8.1 Article Monitor

The Article Monitor aims to give an overall overview of a Wikipedia article's status. It summarizes statistical measures as well as analysis reports or links to analysis tools for on-demand computation. The general statistics include the timestamp and user name of the first and recent edit made to the article, the number of unique authors and anonymous edits, the number of references that are used to prove the article's reliability, the number of media files used in the article and a page visitor count for the previous day and the previous month.

The analysis part of the output window contains a link to the Link Extractor to enable a user to compare link distribution throughout different language versions of an article. If available for that specific article, the KIT's WikiGini score to determine the authorship distribution, a Change Detector notice, and a news article count provided by JSI's Newsfeed will also be displayed. In the German Wikipedia there is also a link to the assessment service wikibu.ch which measures the overall article quality.

8.1.1 Installation

To provide an API for the Article Monitor the Supporting Tools need to be installed on a web server. The web server needs to be configured to pass any request to the index.php file, since this is the only valid entry point for the application. The web server needs to be able to interpret PHP code, a version of 5.3 or higher is recommended. Since the Supporting Tools rely on several tools using a database to store precomputed analysis results, a database management system also needs to be available. The applications have been tested using MySQL and MariaDB. The application is strongly relying on Wikipedia database access with several language versions. Hence it is highly recommended to have the applications running on the Wikimedia Toolserver, Wikimedia Labs or a similar environment bound to Wikimedia projects.

To enable Wikipedia's users to use custom code the MediaWiki software provides user subpages where the code can be inserted. To actually use the Article Monitor a javascript gadget code needs to be installed on the user's custom script page. The tool was tested using the interface skins "Vector" and "Monobook" which is used by a vast majority of the users. It runs on any browser and any operating system.

8.1.2 Configuration

The applications can be configured by a global configuration file that needs to be deployed to the application's root folder. A sample configuration is provided in the code repository. Also all incoming requests need to be passed to the index.php file. This can be achieved by setting up the rewrite directives in the web server configuration. The manual of the web server software should be consulted to do this.

8.1.3 Administration

Assuming that the tools have been installed on an environment such as Wikimedia Toolserver or Wikimedia Labs, there is no administration needed.

8.1.4 Integration with Other Modules

Most of the analysis are carried out either by the Supporting Tools themselves or by tools included in the toolkit showcase which is installed on the same server. All other tools provide an API that can be requested by passing the ID, the language version or the URL of the respective Wikipedia article in the querystring. The external tools respond by sending a JSON object that is parsed by the Article Monitor's backend.

8.1.5 Maintenance Details

As for every web application there are access logs written by default. The log files should automatically be archived to prevent the hard disk from running out of storage space.

8.2 Article List Generator

The Article List Generator enables users to search for articles in given categories. The results can be narrowed down using several filters provided by its backend. Its aim is to provide an easy and fast way to crawl an author's preferred categories for articles that may need to be improved or revised.

8.2.1 Installation and Deployment

The Article List Generator is part of the Supporting Tools, hence these need to be installed on a web server. The web server needs to be configured to pass any request to the index.php file, since this is the only valid entry point for the application. The web server needs to be able to interpret PHP code, a version of 5.3 or higher is recommended. Since the Supporting Tools rely on several tools using a database to store precomputed analysis results, a database management system also needs to be available. The applications have been tested using MySQL and MariaDB. The application is strongly relying on Wikipedia database access with several language versions. Hence it is highly recommended to have the applications running on the Wikimedia Toolserver, Wikimedia Labs or a similar environment bound to Wikimedia projects.

The application consists of a frontend that the user is interacting with and a backend the requests are sent to. The backend needs a Python interpreter of version 2.7. Apart from standard libraries the backend uses the additional modules beaker.cache and wikitools which need to be installed. To find the article pages contained in the given category it makes use of CatGraph, which provides a fast low-level search technology.

8.2.2 Configuration

The Article List Generator as a part of the Supporting Tools can be configured by a global configuration file that needs to be deployed to the application's root folder. A sample configuration is provided in the code repository. Also all incoming requests need to be passed to the index.php file. This can be achieved by setting up the rewrite directives in the web server configuration. The manual of the web server software should be consulted to do this.

Its backend also needs to be configured by a configuration file to enable it to access CatGraph.

8.2.3 Administration

Assuming that the tools have been installed on an environment such as Wikimedia Toolserver or Wikimedia Labs, there is no administration needed.

8.2.4 Integration with Other Modules

The Article List Generator provides a filter to put out only articles that were detected as possibly outdated by the Change Detector.

8.2.5 Maintenance Details

As for every web application there are access logs written by default. The log files should automatically be archived to prevent the hard disk from running out of storage space.

8.3 Wikimedia Toolkit

The toolkit is originally created as a showcase for several tools developed by Wikimedia Deutschland e. V. and the Karlsruhe Institute of Technology (KIT) and the Josef-Stefan-Institute (JSI). It contains Link Extractor, Change Detector, WikiGini, News Finder, Wikipedia Maps, and Corpex. While Link Extractor, Change Detector, WikiGini and News Finder access live data from either Wikipedia's databases or the API provided by JSI and KIT, Wikipedia Maps and Corpex use precomputed data that needs to be stored on the web server the toolkit is running on.

8.3.1 Installation and Deployment

The needed source code for the toolkit can be downloaded from its publicly readable repository. Since Link Extractor and Change Detector rely on accessing the Wikipedia databases it is recommended to install it on the Wikimedia Toolserver, Wikimedia Labs or a similar environment.

The toolkit is written in PHP, thus a PHP interpreter is necessary to use it. It also fetches needed data from a database management system such as MySQL or MariaDB. Some of the tools are fetching data from an API running on remote servers. In order to use these tools, a connection to the internet is necessary.

8.3.2 Configuration

To use the toolkit and the tools that reside in it, the access to Wikipedia databases and the request URI of the APIs need to be configured.

8.3.3 Administration

Assuming that the tools have been installed on an environment such as Wikimedia Toolserver or Wikimedia Labs, there is no administration needed.

8.3.4 Integration with Other Modules

The integration with other modules that have been developed by other project partners participating in the RENDER project is carried out by accessing the respective API on remote servers or by using precomputed data stored on the machine the toolkit is installed on.

8.3.5 Maintenance Details

As for every web application there are access logs written by default. The log files should automatically be archived to prevent the hard disk from running out of storage space.

9 DiversiNews

DiversiNews is an interactive tool, which allows users to browse and summarize news articles from different perspectives. Given a certain topic of interest, the tool provides a succinct summary of the latest news, as well as the individual news items that have been summarized, sorted by relevance. The user can further specify, which perspective on the news should be emphasized: articles containing certain predominant keywords, articles that belong to a certain geographical region, or articles with a positive or negative outlook. Depending on the user preferences, the summary is updated accordingly and the articles get reordered.

9.1 Installation and Deployment

The server runs on Windows NT. The client runs in any modern web browser (tested with recent versions of Firefox and Chrome). The typical deployment requires an internet connection for the server, although a local, single-machine deployment of the server is possible.

DiversiNews is a predominantly monolithic application as illustrated in
Figure 6. Both components appearing in the left part of the figure are not of interest here – the thin web client uses Javascript to relay user requests to the server and render its responses, while the OWLIM semantic database provided by Ontotext is documented separately.
The two important components comprising the application server are the database, containing all the news articles as well as various indices and providing a way to perform fast searches over the data, and the summarizer (developed by Google and documented in detail separately). The server also contains a configuration component and a lightweight web server. All the components are tied together by application logic code which performs the retrieval and ranking of the results in accordance with the user query.

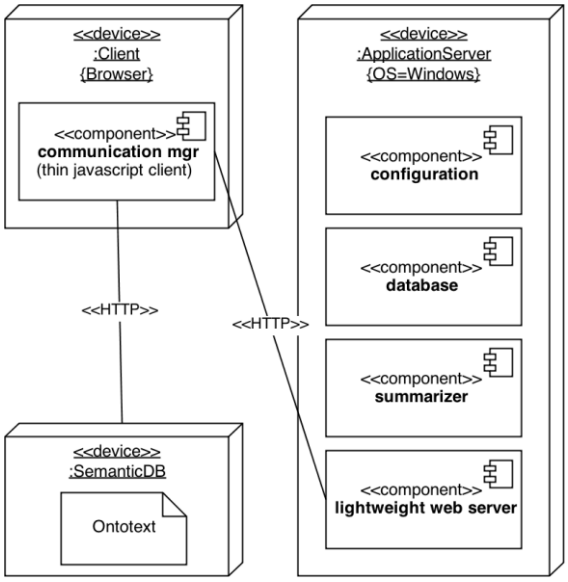


Figure 6: DiversiNews Deployment Diagram

9.2 Configuration

9.2.1 User Interface Configuration

The user interface is not configurable beyond resizing the client window. Clients will be able to select the summarization algorithm in the near future.

9.2.2 Configuration of the Integration Between Modules

The application server and the client communicate over HTTP. The server accepts only GET requests and interprets the following parameters:

- **q** – a string denoting the query. The query can be either a phrase (in which case all articles from the DB containing the phrase will comprise the result) or a article cluster ID (clusters from Google News are used in this demo along with their original IDs).
- **sum –** a numerical value denoting whether the server response would include a summary. 0 denotes "no summary", 1 denotes "Google summarizer" and other numbers are reserved for future summarization algorithms. Because summarization is a time-consuming process, each page visit actually triggers two requests: one with sum=0 to keep the response fast, and one with sum!=0 which provides the summary with a few seconds' delay.
- **reqId** – request ID, returned verbatim in the response (used for tracking and temporal ordering of responses)

The response to the client is a JSON dict with the following keys:

- **reqId** – echoed from the query
- **items** – an array of articles; each item is a dict with the following keys:
  - title
  - description – pruned content
  - URL – URL of the original article
  - lat, lon – location on the map
- **keyEntities** – an array of dbpedia URIs of the main named entities mentioned in this article collection.
- **summary**
- **clusters** – description of the semantic map in the upper right corner of the page.

9.3 Administration

Not applicable.

9.4 Integration with Other Modules

DiversiNews integrates the Google multi-document summarizer, described in Section 10.

9.5 Maintenance Details

Building. The source code is packaged in a ready-to-build Visual Studio solution. If the solution fails, refer to Appendix B for detailed instruction on building DiversiNews; special emphasis is given to integrating the summarizer into the remainder of the application.
Environment. When running the application server, the current working directory has to be the DiversiNews home directory which contains the configuration and various binary dependencies. See Appendix A for a short description. The home directory is already provided and should work without further modifications.

Running. The main entry point for the application is `NewsServer.exe`. Run it without arguments to get the list of command line options together with explanations. Assuming your database resides in `./index/` (or you do not have a database of articles yet), you may want to run the program (starting from the home directory; see previous paragraph) as

..\aiutils\NewsMiner\cpp\x64\Release\NewsServer.exe -def:./news.def -index:./index_2weeks4/ -port:9991 -add:T

Database maintenance. To add articles to the database, use the separate `NewsParser.exe` entry point. The assumption is your articles have been processed by Enrycher and you have the in xml format, one or more articles per file. If the files are in the folder `./xml_in/`, you may want to run NewsParser.exe as

..\aiutils\NewsMiner\cpp\x64\Release\NewsParser.exe parse online -inraw:./xml_in/ -outraw:./xml_done/ -forward:http://127.0.0.1:9991/add -sleep:1 -source:ijs-enrycher

Again, run the executable without arguments for a full list of possible arguments with explanations.
Exiting. The server should always be shut down by sending a HTTP GET request to
http://127.0.0.1:9991/exit
Doing otherwise (in particular, forcibly terminating the executable puts you at risk of corrupting the database and losing all existing articles

Annex C contains a README for the DiversiNews home directory.

10 TopicSum Summarizer

The TopicSum summarizer is a C++ library that the components of RENDER architecture can invoke to generate on-the-fly summaries of documents and document collections. It is one of the building blocks of the DiversiNews interface, where it is used to generate interactive summaries of news collections tailored on the user specification.

10.1 Installation and Deployment

The bundle with the summarizer source code is available for download from: https://code.google.com/p/summarizer/source/checkout. The software is released under the terms of the Apache Software License 2.0.

The only requirement for the installation is a working CNU/C++ development environment. It has been thoroughly tested in a GNU/Linux environment, but the code should be ported to other platforms with little or no modifications. The code is not architecture dependent, and can run on all the platforms for with the GNU C compiler is available.

The suggested installation path requires a working GNU - *automake* pipeline, which ships with most Linux/FreeBSD flavors default installations. Ports of the GNU build tools are available for Windows and OSX environments.

If *automake* is available, to install the software it is sufficient to run the following commands:

1. `./autogen.sh`
2. `./configure --prefix=</path/to/install/dir>`
3. `make && make install`

The README file that ships with the package provides further installation instructions.

10.2 Configuration and Administration

TopicSum API allows clients to specify all required parameters at runtime, and therefore it does not need any special setup, configuration or administration.

10.3 Integration with Other Modules

To compile a binary that uses TopicSum, several flags need to be passed to the compiler and linker. TopicSum integrates with *pkg-config* to make this stage as simple as possible. For systems on which *pkg-config* is installed, it can be invoked to get a list of the required flags in the following way:

- To print compiler flags:

  ```
  pkg-config --cflags summarizer flags
  ```

- To print linker flags:

  ```
  pkg-config --libs summarizer
  ```

- To print both:

  ```
  pkg-config --cflags --libs summarizer
  ```

The outout of *pkg*-config can be directly used to build and link binaries, as in the following example:

g++ my_program.cc `pkg-config --cflags --libs summarizer`

To write an *autoconf*-based package that uses TopicSum, the suggested path is to set the `PKG_CHECK_MODULES` macro in the `configure` script, as shown here:

PKG_CHECK_MODULES([summarizer], [summarizer])

See *pkg-config* man page for more info.

The interface of TopicSum is extremely simple and self-explanatory:

```
class TopicSummarizer {
 public:
  TopicSummarizer() {}
  // Initializes the summarizer with given options.
  // Returns false if the summarizer could not be initialized.
  bool Init(const SummarizerOptions& options);
  // Summarize a collection of documents and stores the result in xml form in
  // summary.
  // Returns false if the collection could not be summarized.
  bool Summarize(const SummaryOptions& options,
                 string* summary);
  // Return the last error message generated by the summarizer
  // during initialization or summarization.
  const string& last_error_message() const { return last_error_message_; }
 protected:
  TopicSum summarizer;
  string last_error_message_;
};
```

The files *example.cc* and *example_simple.cc* in the *src* directory of the package contain fully documented examples of TopicSum inclusion and invocation in C++ programs. An example of TopicSum invocation from a C++ binary is also shown in Annex E.

10.4 Maintenance Details

TopicSum does not require any special maintenance, provided that the stack of GNU libraries, on which it depends continues to provide a stable interface.

References

[1] http://www.render-project.eu

[2] http://enrycher.ijs.si/

[3] http://aidemo.ijs.si/render/index.html

[4] http://195.235.93.35:8080/TOMT/

Annex A Example Enrycher config.xml File for Dependencies

```
<config>
        <en-ner-date>OpenNLP_Java/models/en-ner-date.bin</en-ner-date>
        <en-ner-location>OpenNLP_Java/models/en-ner-location.bin</en-ner-
location>
        <en-ner-money>OpenNLP_Java/models/en-ner-money.bin</en-ner-money>
        <en-ner-organization>OpenNLP_Java/models/en-ner-organization.bin</en-
ner-organization>
        <en-ner-percentage>OpenNLP_Java/models/en-ner-percentage.bin</en-ner-
percentage>
        <en-ner-person>OpenNLP_Java/models/en-ner-person.bin</en-ner-person>
        <sentence-splitter-model>OpenNLP_Java/models/en-sent.bin</sentence-
splitter-model>
        <tokenizer-model>OpenNLP_Java/models/en-token.bin</tokenizer-model>
        <postagger-model>OpenNLP_Java/models/en-pos-maxent.bin</postagger-model>
        <chunker-model>OpenNLP_Java/models/en-chunker.bin</chunker-model>
        <stop-words-folder>Coreference_Java/</stop-words-folder>
        <gate-home>GATE_Java/</gate-home>
        <gate-instances>1</gate-instances>
        <gate-size>100</gate-size>
        <stanford-instances>2</stanford-instances>
        <stanford-lang-model>StanfordParser_Java/englishPCFG.ser.gz</stanford-
lang-model>
        <test-folder>test/</test-folder>
        <log4j-configuration>Log4j/</log4j-configuration>
</config>
```

Annex B Example Section of Enrycher web.xml Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>EnrycherWeb</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>EnrycherServletPre</servlet-name>
    <servlet-class>si.ijs.enrycher.web.EnrycherServlet</servlet-class>
    <init-param>
      <param-name>pipeline-in</param-name>
      <param-value>si.ijs.enrycher.service.EnrycherPreprocessService</param-
value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>EnrycherServletPre</servlet-name>
    <url-pattern>/run-pre</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>EnrycherServletAnn</servlet-name>
    <servlet-class>si.ijs.enrycher.web.EnrycherServlet</servlet-class>
    <init-param>
      <param-name>pipeline-in</param-name>
      <param-value>si.ijs.enrycher.service.EnrycherPreprocessService</param-
value>
    </init-param>
    <init-param>
      <param-name>pipeline</param-name>
      <param-
value>si.ijs.enrycher.service.EnrycherPOSService;si.ijs.enrycher.service.Enryche
rNERGateService;si.ijs.enrycher.service.EnrycherNERService;si.ijs.enrycher.servi
ce.EnrycherEMService;si.ijs.enrycher.service.EnrycherWrapService</param-value>
    </init-param>
    <init-param>
      <param-name>si.ijs.enrycher.service.EnrycherWrapService</param-name>
      <param-value>http://...</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>EnrycherServletAnn</servlet-name>
    <url-pattern>/run-annotation</url-pattern>
  </servlet-mapping>
</web-app>
```

Annex C README for DiversiNews Home Directory

```
This is the default starting directory for the NewsServer MSVC project
running SearchPoint for DiversiNews (= RENDER Google Use case demo including
Google's summarizer).

It contains
  - all of glib's binary dependencies (unicode etc)
  - all of summarizer's binary dependencies (dlls)
  - the qminer index
```

Annex D README for Building the Summarizer under Visual Studio

```
Building instructions for Windows
=================================
(Visual Studio 2010, statically linked. x86 and x64 both supported.)


Note 1: The instructions below use "SUM" to refer to the directory into which the
summarizer is checked-out.

Note 2: The instructions below are for building the whole thing with /MD (i.e.
dynamically linking the C runtime library). If you want /MT:
       * you have to rebuild iconv, libxml and zlib from source
       * you should replace all occurrences of "gtest-md" in the instructions
         below with "gtest"

Note 3: If things don't work for you, try editing src/config.h. This file is normally
automatically created during the (autoconf-guided) build process.


Building the dependencies:
       * iconv, libxml, zlib (all of them present in SUM/dependencies-win; for
         linux, use your packet manager):
            * No compilation needed.
              The win32 binaries were taken from www.zlatkovic.com/libxml.en.html
              The x64 binaries were taken from
http://sourceforge.net/projects/sbml/files/libsbml/win-dependencies/libsbml-win64-
dependencies.zip/download
              except zlib which was compiled from http://zlib-
win64.googlecode.com/files/zlib-1.2.7.f-win64-v1.7z
       * protobuf:
            * [already done] Replace gtest with a newer version (1.6.0 works - tested);
get it from the internet.
              As of 2012-08-30, protobuf includes and older version which does not
compile with VS2010.
            * Open SUM/protobuf-2.4.1/vsprojects/protobuf.sln
            * [already done] For all projects, go to properties -> C/C++ -> Code
Generation, set Runtime Library to "Multi-threaded [Debug] DLL (/MD[d])"
            * Build the solution (probably both in Release and Debug mode). In case of
errors, try compiling multiple times;
              the projects in the solution depend on one another and the compilation
order does not reflect these dependencies.
            * Run SUM/protobuf-2.4.1/vsprojects/extract_includes.bat to copy all the
public headers into a separate "include" directory
       * gtest:
            * Open SUM/protobuf-2.4.1/gtest/msvc/gtest-md.sln, build it (probably both
in Release and Debug mode)
            * Note: The autoconf setup for the summarizer downloads its own copy of
gtest; the MSVC solution is configured to use the one that ships with protobuf. This
should be inconsequential.

Building the summarizer itself:
       * Things already configured in the project (for future reference only):
            * Go to "View -> Property Manager" (or "View -> Other Windows -> Property
Manager"), right-click on Debug in project summarizer,
              and add a propery sheet called e.g. build_macros.
              Right-click build-macros, choose properties, navigate to User Macros, add
a macro "SummarizerRoot" with value "../../".
              Close the properties. Right-click on Release, choose "Add existing
propery sheet", choose "build_macros.props".
            * Add all .h and .cc files to the solution
            * Open project properties. In the "C/C++ -> Preprocessor" category, add the
following to "Preprocessor Definitions":
                  DATADIR="$(SummarizerRoot)/src/";HAVE_CONFIG_H;
            * Open project properties. In the "C/C++ -> General" category, add the
following to "Additional Include Directories" (for x64, replace include32->include64):
                  $(SummarizerRoot)\src;$(SummarizerRoot)\dependencies-
win\include32;$(SummarizerRoot)\protobuf-
2.4.1\vsprojects\include;$(SummarizerRoot)\protobuf-2.4.1\gtest\include;
```

```
            * Open project properties. In the "Linker -> Input" category, add the
following to "Additional Dependencies" (for Release, replace main-mdd with main-md and
gtestd with gtest):

      wsock32.lib;iconv_a.lib;zlib.lib;libxml2_a.lib;libprotobuf.lib;libprotoc.lib;gtest
d.lib;gtest_main-mdd.lib;
            * Open project properties. In the "Linker -> General" category, add the
following to "Additional Library Directories" (for x64, replace lib32->lib64):
                  $(SummarizerRoot)\dependencies-win\lib64;$(SummarizerRoot)\protobuf-
2.4.1\vsprojects\$(Configuration);$(SummarizerRoot)\protobuf-2.4.1\gtest\msvc\gtest-
md\$(Configuration)
      * Things to do:
            * Build the dependencies (see above)
            * Change SummarizerRoot (see above) if needed; it should point to the
directory into which you checked out from git.
            * Compile and run. The solution, as-is, excludes all the tests, so the only
file with a main() is example.cc. To run it,
              change the configuration type (Project Properties -> General) from .lib
to .exe.

Using the summarizer as a library:
      * Your project should include summarizer.lib and all the include paths from the
summarizer project
      * If you built for x64, place the dlls from SUM/dependencies-win/dll64 on the
final executable's path
```


Annex E Example of TopicSum Invocation from a C++ Binary

```
// Copyright 2012 Google Inc. All Rights Reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
// Include summarizer header.
#include "summarizer/summarizer.h"
#include <fstream>
#include <iostream>
#include <streambuf>
#include <string>
#include <vector>
using namespace std;
using namespace topicsum;
int main(int argc, char** argv) {
  // Read article.
  ifstream article_file("testdata/article_14319162.xml");
  if (!article_file) {
    cerr << "File not found\n";
    return 1;
  }
  // Read file content (STL magic).
  string article((istreambuf_iterator<char>(article_file)),
                 istreambuf_iterator<char>());
  // Create options for summarizer.
  SummarizerOptions summarizer_options;
  // Create KLSum options.
  KLSumOptions* klsumoptions = summarizer_options.mutable_klsum_options();
  // (Optional) Weight of a sentence score from a summarizer.
  // Default: 1.0
  klsumoptions->set_summary_weight(0.5);
  // (Optional) Weight of a prior score of a sentence.
  // Default: 0.0
  klsumoptions->set_prior_weight(0.5);
  // Add one document to collection.
  Article* document = summarizer_options.add_article();
  // Set the content of the article.
  document->set_content(article);
  // Add scores for sentences. Scores must be in same order as sentences in
  // the document.
  const int kNumberOfSentences = 17;
  for (int i = 0; i < kNumberOfSentences; i++) {
    // Prior score of a sentence.
    document->add_score(0.5);
  }
  // Create summary options.
  SummaryOptions summary_options;
  // Set the expected length of the summary.
  SummaryLength* sum_length = summary_options.mutable_length();
  // Set desired unit (SENTENCE or TOKEN).
  sum_length->set_unit(SummaryLength::SENTENCE);
  // Set the expected length of the summary.
  sum_length->set_length(2);
  // Initialize summarizer with collection and options. Return true on success.
  TopicSummarizer summarizer;
  if(!summarizer.Init(summarizer_options)) {
    // Print last error.
    cerr << summarizer.last_error_message();
    return 1;
```

```
  }
  // Sumarize the collection with desired options. Return true on success.
  string summary;
  if (!summarizer.Summarize(summary_options, &summary)) {
    // Print last error.
    cerr << summarizer.last_error_message();
    return 1;
  }
  // Print the summary.
  cout << summary << endl;
  return 0;
}
```