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

RENDER

Deliverable 2.2.1

Prototype of the Fact Mining Toolkit

Editor:	Delia Rusu, Jožef Stefan Institute
Author(s):	Blaž Fortuna, Jožef Stefan Institute; Delia Rusu, Jožef Stefan Institute; Mitja Trampuš, , Jožef Stefan Institute; Lorand Dali, Jožef Stefan Institute; Tadej Štajner, Jožef Stefan Institute; Marko Grobelnik, Jožef Stefan Institute
Deliverable Nature:	Prototype (P)
Dissemination Level: (Confidentiality)[1]	Public (PU)
Contractual Delivery Date:	March 2011
Actual Delivery Date:	March 2011
Suggested Readers:	developers working on WP4 – Diversity Toolkit, developers creating case study prototypes (WP5)
Version:	1.4
Keywords:	Fact mining, annotation extraction, disambiguation, named entity identifycation, named entity co-reference and anaphora resolution, assertion extraction, RDF

[1] Please indicate the dissemination level using one of the following codes:
• **PU =** Public • **PP =** Restricted to other programme participants (including the Commission Services) • **RE =** Restricted to a group specified by the consortium (including the Commission Services) • **CO =** Confidential, only for members of the consortium (including the Commission Services) • **Restreint UE =** Classified with the classification level "Restreint UE" according to Commission Decision 2001/844 and amendments • **Confidentiel UE =** Classified with the mention of the classification level "Confidentiel UE" according to Commission Decision 2001/844 and amendments • **Secret UE =** Classified with the mention of the classification level "Secret UE" according to Commission Decision 2001/844 and amendments

Disclaimer

Full Project Title:	RENDER – Reflecting Knowledge Diversity
Short Project Title:	RENDER
Number and Title of Work package:	WP2 Diversity mining
Document Title:	D2.2.1 - Prototype of the Fact Mining Toolkit
Editor (Name, Affiliation)	Delia Rusu, Jožef Stefan Institute
Work package Leader (Name, affiliation)	Delia Rusu, Jožef Stefan Institute
Estimation of PM spent on the deliverable:	8

Executive Summary

This deliverable describes a prototype of the Fact Mining Toolkit, which provides fact extraction functionality and is designed as a Web Service with nine components: one for pre-processing plain-text, six core components providing annotations, assertions and categories and two components for rendering the output – either in RDF or as a graphical visualization. As an example application we developed a News Fact Extraction Service, which applies fact extraction to a stream of news articles. Finally, we present on-going research work in the lines of improving fact extraction by identifying fact templates across multiple documents.

The Fact Extraction Service (also referred to as Enrycher) provides shallow as well as deep text processing functionalities at the text document level. Shallow text processing regards topic and keyword detection and named entity extraction: names of people, locations and organizations, dates, percentages and money amounts occurring in text. Deep text processing implies named entity resolution and merging, word sense disambiguation and assertion extraction. Named entity resolution is performed with respect to existing knowledge bases: DBpedia, YAGO, OpenCyc with the goal of using existing knowledge to enrich the set of features associated to named entities. Entity merging involves co-reference and anaphora resolution for named entities while assertion extraction takes the form of identifying subject – predicate – object sentence elements together with their modifiers (adjectives, adverbs) and negations.

Enrycher has been evaluated on a component basis, and compared to other state-of-the-art fact extraction and, more general, information extraction systems.

The News Fact Extraction Service was developed as an infrastructure for connecting to a real-time stream of news articles. A subset of articles from relevant mainstream media and blogs was sent through Enrycher, and the extracted facts were exposed using a publish-subscribe interface.

We also describe on-going research work in the direction of semantically capturing the macro-level aspects of text. It focuses on constructing domain *templates*, a generic “summary” that fits many pieces of text on a specific theme (e.g. news stories about bombings) at the same time.

As an end result of this deliverable, we provide our project partners with a web service API for interacting with Enrycher, our Fact Extraction Service. Moreover, we release a news corpus composed of selected items processed with Enrycher, which the partners can subscribe to.

Table of Contents

Executive Summary 3
Table of Contents 4
List of Figures 5
List of Tables 6
Abbreviations 7
Definitions 8
1 Introduction 9
1.1 Illustrative Example 10
2 Fact Extraction Service 12
2.1 Architecture 12
2.2 Object Model 13
2.3 Interface 14
2.4 Services 15
2.4.1 Introduction 15
2.4.2 Text Pre-processing 17
2.4.3 Named Entity Extraction 18
2.4.4 Entity Merging 18
2.4.5 Co-reference Resolution 18
2.4.6 Anaphora Resolution 19
2.4.7 Entity Resolution 20
2.4.8 Architecture and implementation 20
2.4.9 Underlying methods 21
2.4.10 Disambiguation 22
2.4.11 Categorization 23
2.4.12 Assertion Extraction 23
2.4.13 Export to RDF 25
2.4.14 Visualization 25
2.5 Related Work 26
3 News Fact Extraction Service 29
3.1 News Stream 29
3.2 Architecture and Interface 30
4 Article Template Discovery 32
4.1 Method Overview 33
4.1.1 Semantic Graph Construction 33
4.2 Approximate Pattern Detection 34
4.3 Preliminary Experiments 35
4.4 Discussion and Future Work 36
5 Conclusions 37
References 38
Annex A Enrycher Object Model 40
A.1 Example of XML format 40

List of Figures

Figure 1 A news article used for exemplifying fact extraction. 11
Figure 2 Enrycher service interfaces 12
Figure 3 Fact extraction pipeline example 12
Figure 4 The Fact Extraction Service Architecture 13
Figure 5 Reference implementation of Enrycher Object Model 14
Figure 6 Java API for calling Enrycher through Web Service API 14
Figure 7 Example call to Enrycher service 15
Figure 8 The Fact Extraction Service hierarchy. 15
Figure 9 The Fact Extraction Service dependencies. 16
Figure 10 An example of text pre-processing 17
Figure 11 The anaphora resolution algorithm 19
Figure 12 An example entity resolution scenario. 21
Figure 13 The *ContextSimilarity* algorithm. 22
Figure 14 Example of categories and keywords assigned to a document by the Categorization service 23
Figure 15 The assertion extraction algorithm 25
Figure 16 Sample from Export to RDF module applied to example from annex A.1 25
Figure 17 Semantic graph visualization 26
Figure 18 Pseudo code detailing the connection to the Spinn3r stream 29
Figure 19 The Architecture of the News Fact Extraction Service 30
Figure 20 Generalization of input graphs and re-specialization of the pattern 34
Figure 21 A selection of detected patterns for each of the domains 35

List of Tables

Table 1 The list of services within Enrycher. 16
Table 2 Performance figures reported by OpenNLP for text-pre-processing 17
Table 3 Performance reported by OpenNLP for extracting the “person” named entity 18
Table 4 WordNet evaluation results (F measure, in %) for the SemEval 2007 Task 7 corpus..... 23
Table 5 A comparative view on five systems..... 27

Abbreviations

HTTP	Hypertext Transfer Protocol
XML	Extensible Markup Language
HTML	Hypertext Markup Language
REST	Representational State Transfer
RDF	Resource Description Framework
NLP	Natural Language Processing
GATE	General Architecture for Text Engineering
DMOZ	The Open Directory Project
NELL	Never-Ending Language Learner
NER	Named Entity Recognition
URL	Uniform Resource Locator

Definitions

Fact at the text document level, facts are represented by meta-data information attached to the document (e.g. the source of the document, its URL, author, editor, publishing date, topics, etc.), and by information extracted from text: topics, keywords, locations, people, organizations, assertions. In this deliverable, we do not assign degrees of truthfulness to the extracted facts.

Fact mining the task of applying data mining and text mining techniques for identifying facts within textual datasets.

Named entities are atomic elements in text representing names of people, locations, organizations, monetary values, percentages, etc. In this deliverable, we consider only the named entities representing people, locations, organizations, dates, percentages and money amounts.

Annotations in this deliverable, we discuss about two types of text annotations: named entity annotations and word/n-gram annotations.

Assertions represent subject – predicate – object sentence elements together with their modifiers (adjectives, adverbs) and negations.

Co-reference (in linguistics) represents a relation between two words/expressions that have a common referent.

Anaphora (in linguistics) defines an instance of an expression that refers to another expression. In this deliverable, we consider a subset of pronominal anaphors for which we determine the antecedents.

Antecedent (in linguistics) the noun replaced by the pronoun. In this deliverable, we consider as candidate antecedents for pronouns the extracted named entities.

1 Introduction

RENDER aims at discovering, managing and representing information on the Web through diversity-aware algorithms. Diversity mining is one of the core enablers in discovering diversity from structured and unstructured datasets, such as mainstream news and social media. To this end, we employ data mining and text mining techniques to discover patterns indicating content diversity (e.g. different vocabulary when describing the same event) or usage diversity (e.g. different references when talking about the same event).

Content diversity can be detected by analysing the text provided by articles published in mainstream news, blogs, forums, twitter, etc. Given a collection of articles, we identify the following dimensions across which diversity can be mined:

- **Topics** – spread of topics across items.
- **Social** – publisher and author of the article and their social influence; people and organizations mentioned and/or connected in text.
- **Geographical** – location of the author; geography addressed in the article.
- **Opinion** expressed by entities in text.
- **Sentiment** of the article or particular entity.
- **Reporting bias** – what is the difference in sentiment, vocabulary and mentioned entities over articles talking about the same event
- **Knowledge** – difference in fact coverage, facts themselves and expressed entity relations.
- **Cross-lingual and Multi-lingual** – diversity of the above-mentioned dimensions across languages.
- **Context** – temporal trends, background knowledge and contrast between sources (mainstream vs. blogs vs. Twitter).

Usage diversity is reflected in the activities carried out by various users, such as passive readers and prosumers (users who actively modify the content by writing in their blog, on Twitter, etc.). Similar to content diversity, we can identify the following dimensions across which usage diversity can be mined:

- **Demography** – descriptors of users reading the content (e.g. age, gender, job, income).
- **Topic** -- typical interests of user reading the content.
- **Geography** – geographic location of the reader
- **Access method** – means of accessing the content (web browser, mobile, forum, phone, email).
- **Social context** – social graph describing the reader community, what possible influences exist within this community.
- **Time** when the users are accessing the content (absolute, day of week, hour of day); changes of their activities through time (trends).

This deliverable presents a prototype of a Fact Extraction Service based on the service named Enrycher [26]. The service extracts factual information from a given document across the **topic**, **social**, **geographic**, **knowledge** and **context** dimensions. The extracted information is represented as a set of facts about the document, which can be further exported as a set of RDF statements. The Enrycher service works on a single document at the time and does not, in its current form, connect information across documents. As such, it is easy to scale-out by design.

Enrycher is an existing system, which was substantially upgraded for the purpose of this deliverable. Firstly, the system architecture was consolidated by limiting core services to Java and C++ and providing native implementation of the Enrycher object model for these two languages. Secondly, new services such as

word sense disambiguation were developed and several existing services were upgraded and/or ported to the new platform. Finally, a user friendly API in both Java and C++ was developed for the end users.

As an example application of Enrycher we developed a News Fact Extraction Service, which applies Enrycher to a stream of news articles. The processed stream is stored, indexed locally and exposed using the publish/subscribe model. The main idea of this application is to establish an environment for a thorough test of the Enrycher service and to provide a sandbox for testing new methods both by JSI and the rest of RENDER partners.

Finally, we present on-going research work which is not currently included in the prototype but will be available in the next release. The core idea is to use a view across more than a single document to discover and extract fact templates and use these templates for better knowledge fact extraction when dealing with a single document.

1.1 Illustrative Example

As an illustrative example (see Figure 1), we consider an online news article provided by Bloomberg.

From this article, we extract facts at three different levels:

1. The **meta-data level**, where we identify:
    a. **News-source**: www.bloomberg.com
    b. **Article URL**: http://www.bloomberg.com/news/2011-01-17/video-gamers-prolonged-play-raises-risk-of-depression-anxiety-phobias.html
    c. **Author**: Elizabeth Lopatto
    d. **Produced at location**: New York
    e. **Editor**: Reg Gale
    f. **Publish Date**: Jan 17, 2011 6:00 AM
    g. **Topics**: U.S., Health Care, Media, Technology, Science
2. The **shallow text processing** level:
    a. **Topics** (e.g. DMoz):

        Top/Health/Mental_Health/Disorders/Mood/Depression

        Top/Health/Mental_Health/Disorders/Mood

        Top/Games/Game_Studies

    b. **Keywords** (e.g. Dmoz):

        Health, Mental Health, Disorders, Mood, Games, Video Games, Depression, Recreation, Browser Based, Game Studies, Anxiety, Women, Society, Recreation and Sports

    c. **Locations**: Singapore, Ames
    d. **People**: Duglas A. Gentile
    e. **Organizations**: Iowa State University, Pediatrics
3. **The deep text processing** level:
    a. **Named entity resolution:**

        Singapore http://sws.geonames.org/1880252/

        Ames http://www.geonames.org/3037869/ames.html

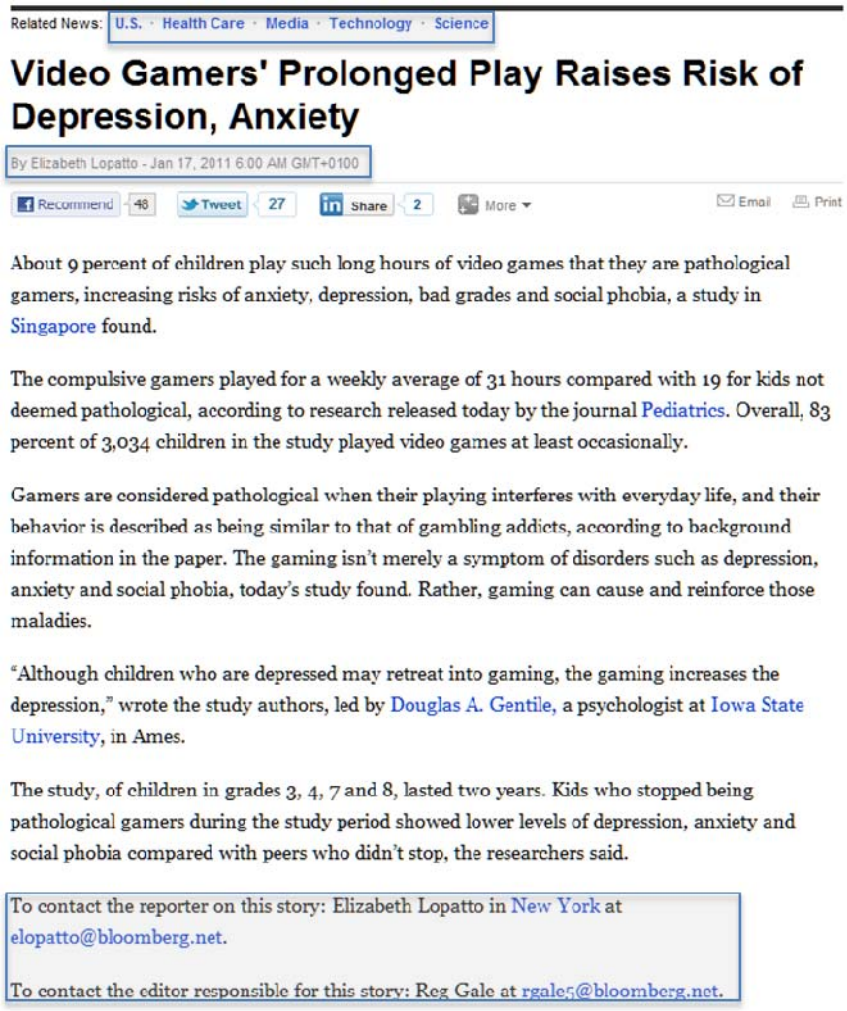
        Iowa State University http://dbpedia.org/resource/Iowa_State_University

    b. **Co-reference** and **anaphora** resolution:

c. **Assertions**:

Children – *play* – [such long] *hours* [of video games]

Words marked between [] represent the modifiers of the object.



Related News: U.S. · Health Care · Media · Technology · Science

Video Gamers' Prolonged Play Raises Risk of Depression, Anxiety

By Elizabeth Lopatto - Jan 17, 2011 6:00 AM GMT+0100

Recommend 48 Tweet 27 Share 2 More Email Print

About 9 percent of children play such long hours of video games that they are pathological gamers, increasing risks of anxiety, depression, bad grades and social phobia, a study in Singapore found.

The compulsive gamers played for a weekly average of 31 hours compared with 19 for kids not deemed pathological, according to research released today by the journal Pediatrics. Overall, 83 percent of 3,034 children in the study played video games at least occasionally.

Gamers are considered pathological when their playing interferes with everyday life, and their behavior is described as being similar to that of gambling addicts, according to background information in the paper. The gaming isn't merely a symptom of disorders such as depression, anxiety and social phobia, today's study found. Rather, gaming can cause and reinforce those maladies.

"Although children who are depressed may retreat into gaming, the gaming increases the depression," wrote the study authors, led by Douglas A. Gentile, a psychologist at Iowa State University, in Ames.

The study, of children in grades 3, 4, 7 and 8, lasted two years. Kids who stopped being pathological gamers during the study period showed lower levels of depression, anxiety and social phobia compared with peers who didn't stop, the researchers said.

To contact the reporter on this story: Elizabeth Lopatto in New York at elopatto@bloomberg.net.

To contact the editor responsible for this story: Reg Gale at rgale5@bloomberg.net.

Figure 1 A news article used for exemplifying fact extraction. The marked sections provide the article meta-data

The meta-data level facts are obtained from a crawler system based on Spinn3r, while the shallow and deep text processing levels are obtained from the Fact Extraction Service (Enrycher).

The extracted facts represent a collection of diverse pieces of information obtained at the document level. Subsequently, some extracted facts can readily fit the Diversity Model[2]: the meta-data information, the content (which we deliver in a semi-structured way via pre-processing services), the agents – which in our case are named entities (people, organizations) or authors (provided in the document meta-data). Other Diversity Model components need to be inferred based on the extracted facts: a quoted named entity can be interpreted as an agent holding an opinion (e.g. "Although children who are depressed may retreat into gaming, the gaming increases the depression," expressed by Duglas A. Gentile).

[2] Diversity Model proposed within the project plenary meeting in Zürich: http://km.aifb.kit.edu/projects/render/index.php/Diversity_model

2 Fact Extraction Service

The Fact Extraction Service (also referred to as Enrycher) provides shallow as well as deep text processing functionalities at the text document level. Shallow text processing regards topic and keyword detection and named entity extraction: names of people, locations and organizations, dates, percentages and money amounts. Deep text processing implies named entity resolution, co-reference as well as anaphora resolution, word sense disambiguation and assertion extraction in the form of subject – predicate – object sentence elements together with their modifiers (adjectives, adverbs) and negations.

The following sections describe the service architecture, the object model, interfaces and finally detail each service component in depth. Section 2 concludes with a related work part and example calls.

2.1 Architecture

The Enrycher service is designed as a distributed set of state-less services connected into a pipeline. Fact Extraction is performed by sending a document through the pipeline, where each service either extracts additional facts, or modifies existing ones.

```
public abstract class EnrycherAbstractService {
    /** Enrycher Service public */
    public abstract String getName();
    /** Enrycher Service optional attributes */
    public Map<String, String> getAttributes()
}

/** Enrycher Service */
public abstract class EnrycherService extends EnrycherAbstractService {
    /** Implementation of Enrycher service functionality */
    public abstract void Enrych(Document doc) throws EnrycherException;
}

/** Enrycher Pre-processing Service */
public abstract class EnrycherInService extends EnrycherAbstractService {
    /** Implementation of Enrycher service functionality */
    public abstract Document Preprocess(InputStream inputStream) throws EnrycherException;
}

/** Enrycher Transformation Service */
public abstract class EnrycherOutService extends EnrycherAbstractService {
    /** Implementation of Enrycher service functionality */
    public abstract void Transform(Document doc, OutputStream outputStream)
        throws EnrycherException;
}
```

Figure 2 Enrycher service interfaces

There are three types of services. The pipeline begins with a **pre-processing service**, which analyses input document and converts it into an Enrycher format. This is followed by a sequence of **extraction services**, which perform the fact extraction. The pipeline can either end with an extraction service, delivering results in Enrycher XML format, or with a **transformation service**, which transforms the enriched document to some other format such as RDF. Figure 2 shows Java interfaces for all three service types. Figure 3 shows an example pipeline composed of 6 services.

Figure 3 Fact extraction pipeline example

There are three supported means for the services to communicate within the pipeline and with the outside world. The most basic one is an XML format, which is also used to communicate with the Enrycher service

as a whole. The preferred way of interaction with the Enrycher services is an object model implemented in Java and C++. Section 2.2 provides details on the Enrycher XML format and its object model.

The pipelines are composed and executed by the Enrycher server. The server can launch several instances of the same service for load-balancing. The functionality of each pipeline is exposed as a RESTful web-service and can be accessed directly, through Java and C++ APIs or viewed on the demo website. Figure 4 depicts the server architecture.

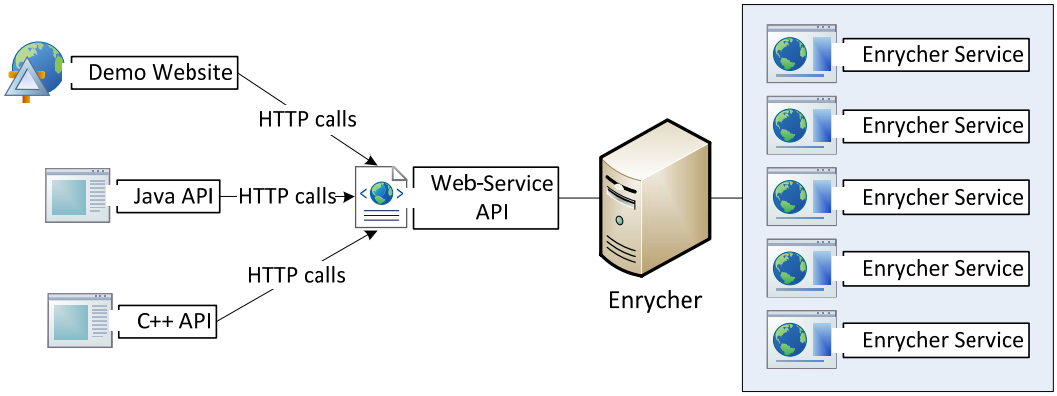


Figure 4 The Fact Extraction Service Architecture

2.2 Object Model

The Enrycher Object Model is primarily based on the Enrycher XML format. This section will first cover the XML format, followed by the description of Object Model in Java and C++.

The top level element in the Enrycher XML format corresponds to one document (`<item>`) and contains the following sub-elements: meta-data (`<metadata>`), text (`<text>`), Annotations (`<annotations>`) and Assertions (`<assertions>`). An example of a document encoded in the Enrycher XML format can be seen in annex A.1.

Document meta-data contains document level annotations (`<semantics>`) and pipeline processing details (`<pipeline>`). Annotations are provided as a list of attributes (`<attribute>`), which have a type and can be either a literal (element's content) or a resource (`resource` attribute). Each attribute forms a triple, with the document as a subject, type as a relation and the literal or resource as an object. Pipeline is described by a list of services (`<tool>`), each service being described by a name, time of execution and duration of execution.

Document text (`<text>`) contains the text of the document broken down into paragraphs (represented by user define tags, such as `<p>` or `<title>`), sentences (`<sentence>`) and tokens (`<token>`). Each sentence and token has a unique ID. There is a possibility of assigning additional attributes to tokens, such as part-of-speech tags (the `pos` attribute in the example).

Annotations (`<annotations>`) consist of annotations grounded in text. Each annotation (`<annotation>`) is described by a unique ID, a set of instances (`<instances>`) and a set of semantic attributes (same format as the document level attributes). Instance is one occurrence of the annotation in text and is described by a unique ID and a list of tokens composing the instance.

Assertions (`<assertions>`) consist of assertions extracted directly from text. Each assertion is described by a unique ID and three elements: subject (`<subject>`), predicate (`<verb>`) and object (`<object>`). Each of the three elements has zero or more modifiers, also given as annotations. Each annotation is given by unique ID, corresponding to one of the existing annotations, and an instance ID, if one exists.

There is a reference implementation of the object model, corresponding to above XML. Figure 5 depicts the diagram of the object model and detailed Java and C++ interfaces will be available at the Enrycher website (http://enrycher.ijs.si). The main class in the model is Document, which can be loaded from and serialized to Enrycher XML format.

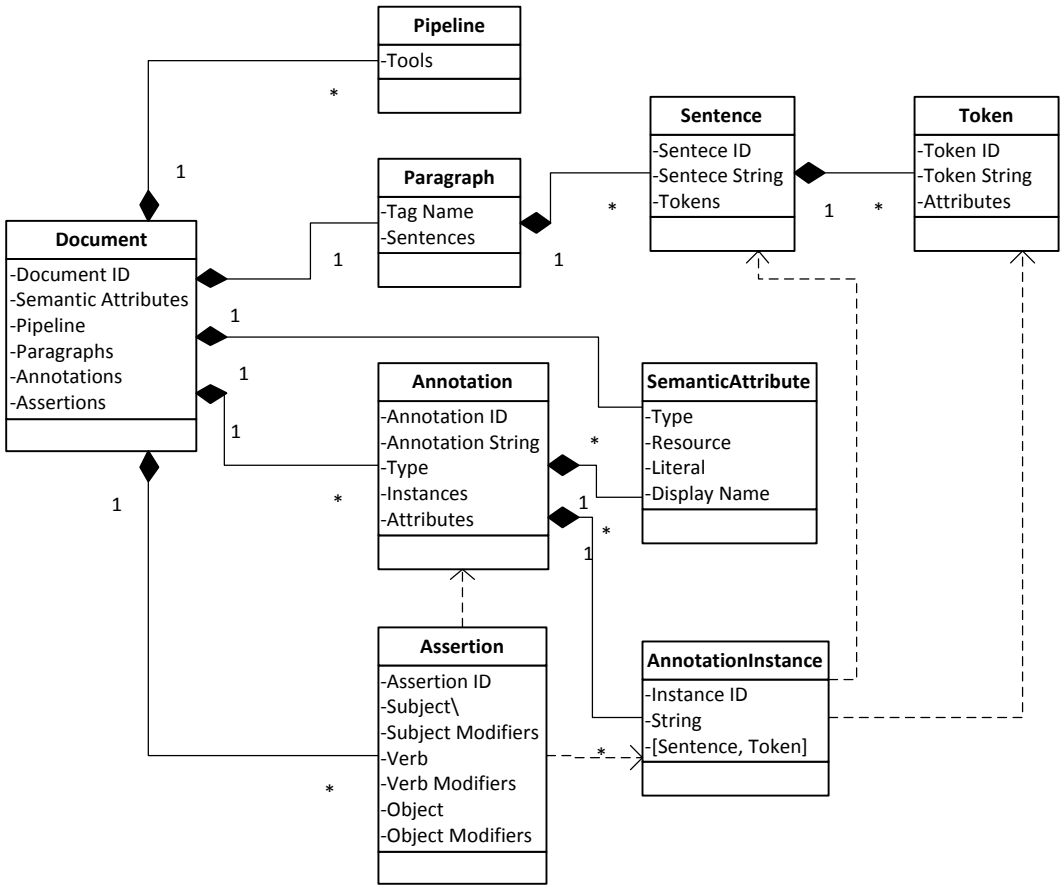


Figure 5 Reference implementation of Enrycher Object Model

2.3 Interface

The main form of interaction with Enrycher service is through a web-service API. Each pipeline is exposed on a particular URL and can be executed by sending an HTTP POST request, with the raw text in the body. The input text can be broken down into paragraphs using standard HTML tags (e.g. `<title>` or `<p>`).

```
public abstract class EnrycherWebExecuter {
     /** Creates an executer for a given pipeline */
     public EnrycherWebExecuter(URL pipelineUrl) { … }
     /** Executes pipeline for a given document */
     public int Process(InputStream docStream) { … }

     /** Called on successful finish of a pipeline without transformation service */
     public abstract void OnEnrycherFinish(int taskId, Document doc);
     /** Called on successful finish of a pipeline with transformation service */
     public abstract void OnEnrycherFinish(int taskId, InputStream inputStream);
     /** Called when there is an error execution a task */
     public abstract void OnEnrycherError(int taskId, EnrycherException e);

     /** Synchronous pipeline execution without transformation service */
     public static Document ProcessSync(URL pipelineUrl, InputStream docStream) { … }
     /** Synchronous pipeline execution with transformation service */
     public static InputStream ProcessTransSync(URL pipelineUrl, InputStream docStream) { … }
}
```

Figure 6 Java API for calling Enrycher through Web Service API

```
// input document
String docString = "This is my document ...";
// URL of Enrycher web service
URL pipelineUrl = new URL("http://example.com/enrycher");
// convert input document to input stream
InputStream docStream = new ByteArrayInputStream(docString.getBytes());
// call Enrycher web service
Document doc = EnrycherWebExecuter.processSync(pipelineUrl, docStream);
// iterate over all the annotations
for (Annotation ann : doc.getAnnotations()) {
    // do something with the annotation
    // ...
}
```

Figure 7 Example call to Enrycher service

The reference implementations in Java and C++ provide wrappers for making web-service calls. The calls can be executed asynchronously, by implementing `EnrycherWebExecuter` in Java or `TEnrycherWebExe` in C++, or synchronously by calling `ProcessSync` or `ProcessTransSync` functions on these two classes. Figure 6 shows the API in case of Java. The wrappers are the preferred way of calling Enrycher services. Figure 7 shows an example call to the Enrycher service.

2.4 Services

2.4.1 Introduction

The Fact Extraction Service is composed of 9 services, depicted in a hierarchy in Figure 8. The services are grouped in 3 types: **Enrycher Pre-processing Services**, marked in green, **Enrycher Extraction Services** marked in blue and **Enrycher Transformation Services** marked in red.

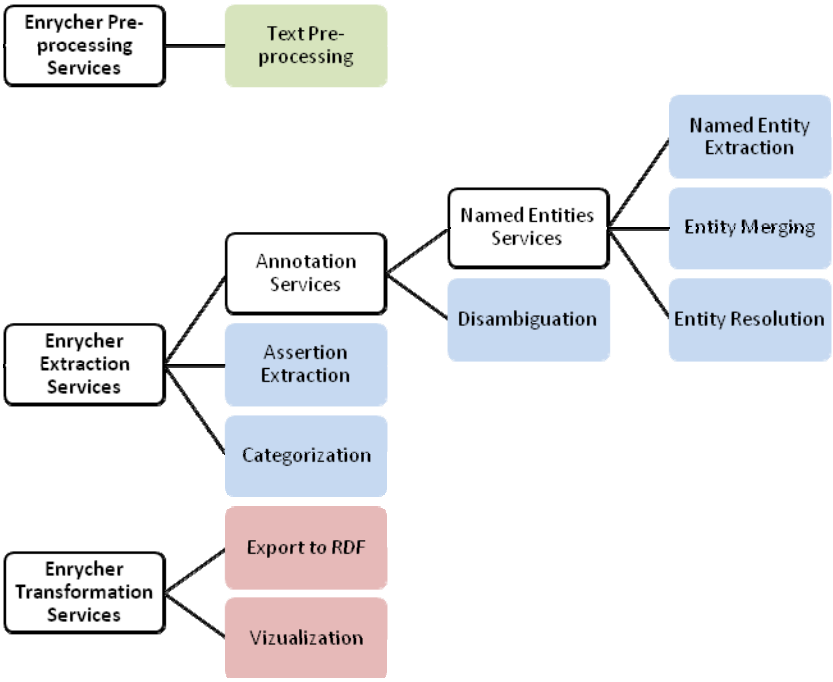


Figure 8 The Fact Extraction Service hierarchy.
The coloured rectangles represent the actual services

The Enrycher Extraction Services provide annotations, assertions and categories. Annotations are further divided based on their type in named entities referring to people, places and organizations, dates, percentages and money amounts or annotations produced by the Disambiguation Service. This service associates WordNet (VUA) resources to words/n-grams in text. The services that process named entities

are Named Entity Extraction, Entity Merging and Entity Resolution. The Named Entity Extraction service identifies names of people, places, organizations, dates, percentages and money amounts in text, while the Entity Merging service provides co-reference and anaphora resolution for the extracted named entities. The Entity Resolution service provides DBpedia, Yago and OpenCyc resources for named entities. The Assertion Extraction Service identifies subject – predicate – object assertions in text, while the Categorization Service provides DMOZ categories.

In Figure 9 we show the dependencies between services, some of them mandatory (depicted with a filled line), some optional (depicted with a dashed line). All services rely on a text pre-processing step handled within the Text Pre-Processing Service, and consisting of sentence splitting and tokenization.

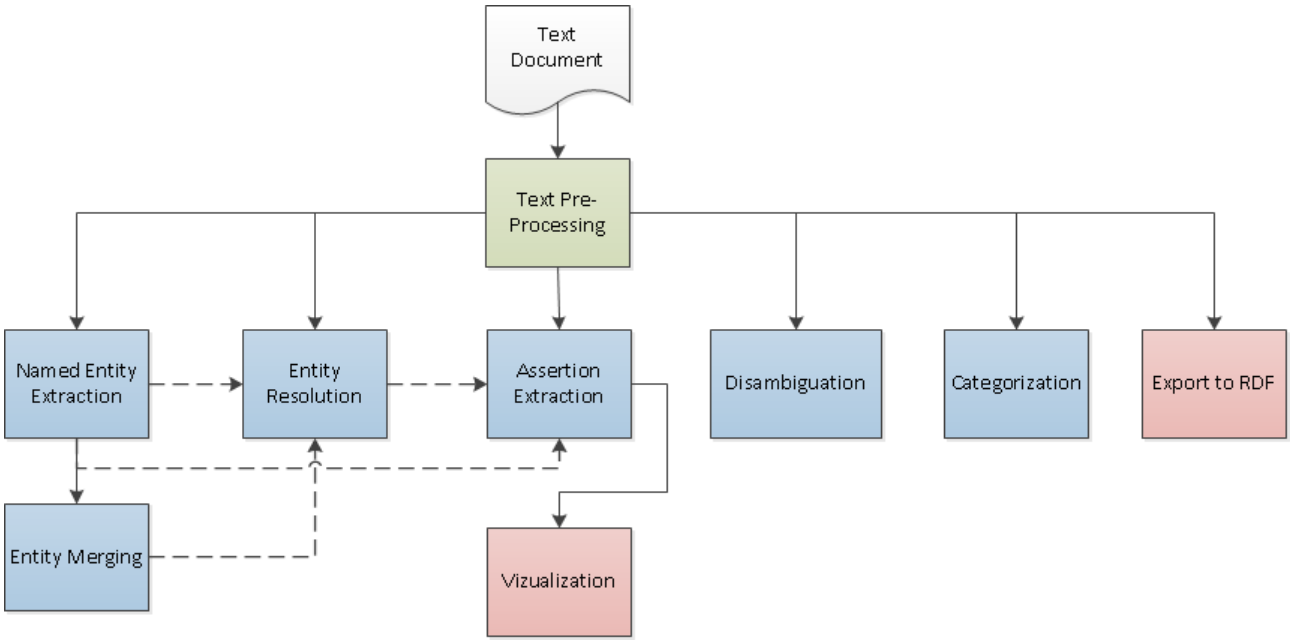


Figure 9 The Fact Extraction Service dependencies.
The dashed line marks dependencies between components that are optional, whereas the filled lines mark required dependencies

Finally, Table 1 lists the full set of services provided, noting for each service its type and requirements. The requirements represent the services that need to be called before the current service can be called. Note that some of the requirements are optional.

Table 1 The list of services within Enrycher.
For each service, the service name, type and requirements of that service (in terms of which other services are a prerequisite) are listed.

	Service Name	Service Type	Requirements
1	Text Pre-processing	Enrycher Pre-processing Service	-
2	Named Entity Extraction	Enrycher Extraction Service	Text Pre-processing
3	Entity Merging	Enrycher Extraction Service	Text Pre-processing Named Entities
4	Entity Resolution	Enrycher Extraction Service	Text Pre-processing Optional: Named Entities, Entity Merging

5	Disambiguation	Enrycher Extraction Service	Text Pre-processing
6	Categorization	Enrycher Extraction Service	Text Pre-processing
7	Assertion Extraction	Enrycher Extraction Service	Text Pre-processing Optional: Named Entities, Entity Resolution
8	Export to RDF	Enrycher Transformation Service	Text Pre-processing
9	Visualization	Enrycher Transformation Service	Assertions

2.4.2 Text Pre-processing

Text pre-processing consists of detecting sentence boundaries, splitting sentences into tokens, and assigning a part of speech tag to each token. For example (see Figure 10):

Input raw text:

```
Manchester United striker Wayne Rooney will face no action from the Football Association
after he appeared to elbow Wigan midfielder James McCarthy. The incident occurred during
Saturday's Premier League match at the DW Stadium in which Rooney scored one of the goals
in United's 4-0 victory.
```

Sentences:

```
Manchester United striker Wayne Rooney will face no action from the Football Association
after he appeared to elbow Wigan midfielder James McCarthy.

The incident occurred during Saturday's Premier League match at the DW Stadium in which
Rooney scored one of the goals in United's 4-0 victory.
```

Tagged tokens for the first sentence (<TOKEN>/<TAG>):

```
Manchester/NNP United/NNP striker/NN Wayne/NNP Rooney/NNP will/MD face/VB no/DT action/NN
from/IN the/DT Football/NNP Association/NNP after/IN he/PRP appeared/VBD to/TO elbow/NN
Wigan/NNP midfielder/NN James/NNP McCarthy/NNP ./.
```

Figure 10 An example of text pre-processing sentence splitting and tokenization

The tag set used for part of speech tagging is the Penn Treebank Tag Set[3].

The OpenNLP library was used for each of the pre-processing operations. The models are maximum entropy models, and the performance figures reported by OpenNLP are (see Table 2):

Table 2 Performance figures reported by OpenNLP for text-pre-processing

	Precision	Recall	F-measure
Sentence Detection	0.9466	0.9096	0.9277
Tokenizing	0.9961	0.9953	0.9957

[3] http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

The accuracy of the POS tagger is measured as the ratio between correctly tagged tokens and the total number of tokens. The accuracy figure reported by OpenNLP is 0.9658.

2.4.3 Named Entity Extraction

In this prototype version of the Fact Extraction Service, named entity extraction is performed based on OpenNLP[4], an existing natural language processing toolkit. As named entities we consider names of people, locations and organizations, dates, percentages and money amounts occurring in the text.

The named entity extraction models are maximum entropy models specific to the language and to the named entity type to be extracted. The performance reported in the OpenNLP documentation for extraction of persons is (see Table 3):

Table 3 Performance reported by OpenNLP for extracting the "person" named entity

Precision	Recall	F-measure
0.8005	0.7451	0.7718

Additionally, we are preparing two more Named Entity Extraction services: one based on GATE (General Architecture for Text Engineering) [8] and another one based on the Stanford Named Entity Recognizer [9].

2.4.4 Entity Merging

The Entity Merging Service has two components: the Co-reference Resolution component and the Anaphora Resolution one. The Co-reference Resolution component is used to merge named entities with different representations. The Anaphora Resolution component identifies corresponding entities for a set of predefined pronouns.

Entity Merging is an Enrycher Extraction Service. The parameters are a list of stop words and the type of resolution needed – Co-reference or Anaphora or both.

2.4.5 Co-reference Resolution

The Co-reference Resolution component is an implementation of a co-reference resolution algorithm that we describe in what follows.

Co-reference resolution is defined as the task of identifying the various representations of a named entity in a given document. The named entity representations are, in this case, the surface forms that define the entity, and represent persons, locations and organizations. Our approach to co-reference resolution is based on [2] and further described in [3]. As a pre-processing step, we eliminate the set of English stop words (e.g. Mr., Inc., etc) for named entities described by more than one word. In order to reduce the search space, for named entities that represent persons we establish their gender, if possible. We identify several co-reference cases:

1. One named entity representation is completely included in the other:

e.g. *Barack*, *Barack Obama*, *(Mr.)*[5] *Obama*, *(Mr.) Barack Obama*, all refer to the same annotation – *Barack Obama*.

2. The named entity is an abbreviation, which is also co-referenced:

e.g. *U.S.*, *U.S.A.* are abbreviations for *United States* and *United States of America*, respectively

The co-reference resolution algorithm was compared to GATE's (version 4.0) co-reference resolver, on a set of 783 named entities that were extracted using GATE from 15 random documents taken from the Reuters

[4] http://incubator.apache.org/opennlp/

[5] Mr. appears in parenthesis because it is a stop-word and will be eliminated in the pre-processing step

RCV1 [1] data set. Our algorithm achieved 96% accuracy (750 out of 783) compared to GATE's 83% accuracy (646 out of 783).

The Co-reference Resolution component requires a stop words list to be loaded beforehand. The component has one parameter, representing the list of annotations of type named entity to be co-referenced, and it returns a list of co-referenced annotations. The list of co-referenced annotations is smaller in size (or equal to, in case no co-references were found) than the initial list of annotations received as a parameter.

2.4.6 Anaphora Resolution

The Anaphora Resolution component is an implementation of an anaphora resolution algorithm, presented in the following.

```
function ANAPHORA-RESOLUTION (pronoun, number_of_sentences) returns a solution, or failure
        candidates ←
        BACKWARD-SEARCH-INSIDE-SENTENCE (pronoun) ∪  BACKWARD-SEARCH (pronoun, number_of_sentences)
        if candidates ≠ ∅ then
        APPLY-ANTECEDENT-INDICATORS (candidates)
        else
        candidates ←
        FORWARD-SEARCH-INSIDE-SENTENCE (pronoun) ∪    FORWARD-SEARCH (pronoun, number_of_sentences)
        if candidates ≠ ∅ then
        APPLY-ANTECEDENT-INDICATORS (candidates)

        result ← MAX-SCORE-CANDIDATE (candidates)
        if result ≠failure then return result
        else return failure

function APPLY-ANTECEDENT-INDICATORS (candidates) returns a solution, or failure
        result ← APPLY-GIVENNESS (candidates) ∪
        APPLY-LEXICAL-REITERATION (candidates) ∪
        APPLY-REFERENTIAL-DISTANCE (candidates) ∪
        APPLY-INDICATING-VERBS (candidates) ∪
        APPLY-COLLOCATION-PATTERN-PREFERENCE          (candidates)
        if result ≠failure then return result
        else return failure
```

Figure 11 The anaphora resolution algorithm

In linguistics, anaphora defines an instance of an expression that refers to another expression; pronouns are often regarded as anaphors. We consider anaphora resolution as a particular case of co-reference resolution, where the aim is to identify the antecedents for a set of predefined pronouns. The antecedents are, in this case, named entities: people, locations, organizations. The pronoun subset we considered for anaphora resolution is formed of: *{I, he, she, it, they}*, and their objective, reflexive and possessive forms, as well as the relative pronoun *who* [3].

Figure 11 describes the anaphora resolution algorithm. The algorithm performs a sequential bi-directional search, first backward within the document, and then forward, starting from the position where the pronoun was found. The goal is to find replacement candidates for a given pronoun, among the named entities.

1. Backward search

   Firstly, we search backwards inside the sentence where we found the pronoun. We select candidates that agree in gender with the pronominal anaphor. Next, we look for possible candidates in the sentences preceding the one where the pronoun is located.

2. Forward search

   If we have found no candidates so far, we search forward within the pronoun sentence, and then forward in the next sentences.

Once the candidates have been selected, we apply antecedent indicators to each of them, and assign scores. The antecedent indicators we have taken into account are: givenness, lexical reiteration, referential distance, indicating verbs and collocation pattern preference. After assigning scores to the candidates found, we select the candidate with the highest overall score as the best replacement for the pronoun. If two candidates have the same overall score, we prefer the one with a higher collocation pattern score. If we cannot make a decision based on this score, we choose the candidate with a greater indicating verbs score. In case of a tie, we select the most recent candidate (the one closest to the pronoun).

For evaluating the anaphora resolution algorithm, we considered the same dataset described in 2.4.5. We compared with two baselines; both take the closest named entity as a pronoun replacement, one takes gender information into account, whereas the other does not. The best result was obtained for the pronoun *he* – 83% accuracy (35 out of 42 pronouns), while the pronouns *they* and *it* were the hardest to resolve, due to multiple candidates and to the impersonal form of *it* [3].

It is recommended that before calling this component, one performs co-reference resolution. In this way, all annotations that match are merged beforehand. The component has two parameters: the list of annotations of the type named entity, and the list of tokens, and it returns a list of annotations, where for each annotation that has a corresponding pronoun a new annotation instance is added.

2.4.7 Entity Resolution

The purpose of this service is to provide integration with existing knowledge bases with the goal of using existing knowledge to enrich the set of features that we are able to extract from text.

2.4.8 Architecture and implementation

This service was designed with separation of concerns in mind. The service already expects that named entity extraction and text pre-processing has been performed. This has the effect of having the entity resolution component relatively language-independent in its implementation. The multilingualism problem can then be solved simply by changing the underlying knowledge base. In terms of input and output, it receives a pre-processed document with a set of named entity annotations, to which it then tries to assign unique URIs.

With regard to using background knowledge data sets, our assumption is that the ontology consists of knowledge database that contains enough data to be able to perform the following tasks.

- First, it should be able to refer to each entity with multiple aliases to facilitate candidate retrieval,
- Second, it should be able to provide enough additional entity features, which we can use to compare those entities to each other and to article anchors that we attempt to link to.

Following these requirements, we chose to use a part of DBpedia, as described in [23] for the facts that it provides both description and attribute data from Wikipedia, as well as references to other ontologies that describe other aspects of the same real-world objects. For the purpose of having rich heterogeneous relational data, we also used the Yago ontology, defined [24], which maps Wikipedia concepts to corresponding WordNet classes. Since a direct mapping from Yago to DBpedia exists, merging the two together is trivial. However, both ontologies are much broader than what our approach requires – we currently only use information on aliases, textual descriptions, *rdf:type* attributes and Yago categories of entities.

2.4.9 Underlying methods

A lot of knowledge is present in plain text rather than a more explicit format. An interesting subset of this challenge is integrating texts with structured and semi-structured resources, such as ontologies. This is especially interesting in the context of Linked Open Data, where the main motivation is to have cross-dataset mappings across as many datasets as possible. In this case, we wish that the subjects and objects in the provided assertions are well-defined within existing ontologies. However, when dealing with entities, mentioned in natural language text, we need to resolve ambiguities in their names. We formulate this as an entity resolution problem, where we are trying to choose the correct corresponding entities from the ontology for the entities mentioned in text.

There are several related topics for this type of problem - reducing a many-to-many relationship to a single link between an in-text entity and an ontology entity: link annotation or ontology alignment, known in the Semantic Web community, word sense disambiguation from the natural language processing community, record linkage and de-duplication from the data management community and entity resolution, as it is known in graph mining.

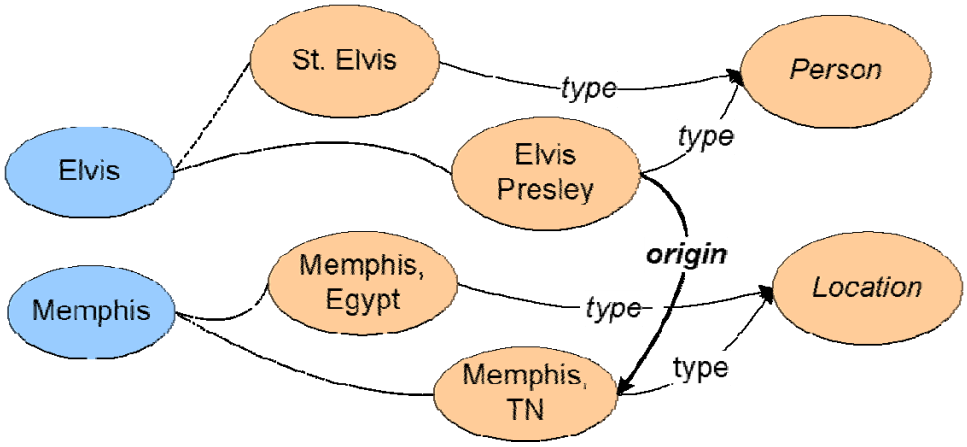


Figure 12 An example entity resolution scenario.
Blue nodes are in-text entities, orange nodes are ontology entities

The process for identifying entities uses several information sources to improve in-text entity resolution quality. Since entities, which are related, tend to appear together in documents more often, we use the entity graph topology as an indicator of this relatedness. For example (see
Figure **12**), in the case when we have a document where there are two unknown entities referred to by the names *"Elvis"* and *"Memphis"*. The first is a common personal name and the second one the name of several locations. We would like to use this relatedness information between those two entities to help in resolving *"Elvis"* as a well-known singer and *"Memphis"* as a city in Tennessee, where the identified singer lived. However, when looking at the whole picture of relationships between two entities, not all of those relationships have the same significance. This paper proposes one such possible approach to heterogeneous relational entity resolution which bases relational significance on the frequency of the relation appearing in the ontology with regard to entity types.

Experiments, conducted in [25] show that resolving entities collectively yields higher quality of named entity resolution than resolving entities independently of one another by using only textual similarity as a heuristic. This confirms the hypothesis that the entities which appear in the same document are somehow related. This means that at any given resolution decision, we consider not only the textual similarity of the content and entity description, but also the relatedness of the entity to other entities that have already been resolved. This approach is known as using relational similarity for collective entity resolution.

2.4.10 Disambiguation

The Disambiguation Service identifies for each word/n-gram in text a corresponding WordNet (VUA) resource, if available.

WordNet [4] is a lexical database of English, comprising nouns, verbs, adjectives and adverbs grouped into synsets. There are two RDF/OWL representations of WordNet in the LOD: WordNet (W3C) models WordNet version 2.0, while WordNet (VUA) models the latest version, 3.0.

The service implements two knowledge-based disambiguation algorithms: a structure based one – *PageRank* and a content-based one – *ContextSimilarity*. By structure we mean the relationships defined between the resources, while by content we mean the textual definitions attached to the resources.

WordNet (VUA) exhibits a graphical structure based on the relationships between resources, for e.g., between an instance and a class described by the *hyponym*, between a class and its superclass described by *hypernym*, and other specific relations such as, *antonym, meronym* and *holonym*. In order to apply the PageRank algorithm [5] in the case of an LOD dataset such as WordNet (VUA), we first build a graph of the dataset *G(V,E)* where the vertices represent all the dataset resources and the edges are the relationships between these resources. As a next step, we identify all the candidate resources for all the words belonging to the text fragment, which are to be annotated. The main difference is the initialization step, which consists of setting the graph vertices to either of the values 0, if the vertex does not represent a candidate resource, or *1/R*, with *R* being the total number of candidate resources. The PageRank value for each vertex *i* ($PR[V_i]$) is computed using the formula:

$$PR[V_i] = \frac{1-D}{N} + D \cdot \sum_{V_j \in InEdges(V_i)} \frac{PR[V_j]}{OutEdges(V_j)}$$

where *N* stands for the total number of vertices in the graph and the damping factor *D* is set to 0.85. The algorithm converges when the difference between the previous and current PageRank values for a vertex is below 10^{-15} (the numerical error for double precision). Finally we select the candidate resource with the highest PageRank score for each word/n-gram to be disambiguated.

```
ContextSimilarity(resource, w) returns Similarity
  Similarity = 0
  NR = GetNeighborhoodResources(resource)
  CW = GetContext(w)
  for i = 1 to Size(NR) do
    CS = sim_cos(NR[i], CW)
    Similarity = Similarity + CS
  end for
return Similarity
```

Figure 13 The *ContextSimilarity* algorithm.
Given a candidate resource and a word/n-gram (*w*), the algorithm computes the similarity between the resource and word's context

In the case of the *ContextSimilarity* algorithm, each candidate resource is scored based on the word overlap between the context around the word to be disambiguated and the human-readable descriptions for a candidate resource, together with its neighbouring resources. A neighbouring resource is determined by retrieving all *hypernyms, hyponyms, meronyms, holonyms* and *attribute* relations. The context of *a* word is represented by the surrounding words in the text fragment, for e.g. all words from the same sentence or paragraph. The overlap between a candidate resource and the word context is computed using the measure of cosine similarity. The candidate resource with the highest score for each word/n-gram will be selected as the annotation. The algorithm is summarized in
Figure 13**.**

The cosine similarity is defined between two bag-of-words vectors (vectors containing all the words in the document) as follows:

$$sim_{cos}(A, B) = \frac{A \cdot B}{\|A\| \|B\|}.$$

The algorithms were evaluated using the SemEval 2007 Task 7: Course Grained English All Words dataset [6]. The SemEval series of evaluation workshops are regarded as a framework for comparing state-of-the-art WSD systems. In Task 7 of SemEval 2007, the participating systems are provided with a corpus consisting of 5 texts annotated with WordNet 2.1 senses from sources like the Wall Street Journal (D_1 to D_3), a Wikipedia entry on computer programming (D_4) and an excerpt from a fiction book (D_5). Out of the 2269 annotated words, 1591 are polysemous (have more than one meaning). The best supervised system obtained an F measure of 82.50%, while the best unsupervised system recorded 77.04%. Table 4 shows the results obtained by our algorithms.

The state-of-the-art supervised disambiguation systems depend on annotated corpora, which is expensive and currently available for a limited number of datasets (mainly WordNet). Therefore we opted for an unsupervised approach which was developed with the aim to generalize to other datasets as well [29].

Table 4 WordNet evaluation results (F measure, in %) for the SemEval 2007 Task 7 corpus. CS is the Context Similarity algorithm, while PR is the PageRank algorithm

	D_1	D_2	D_3	D_4	D_5	All
CS	75.27	77.84	72.80	77.84	72.46	75.50
PR	74.19	74.67	73.60	73.71	71.01	73.51

The service parameters are a WordNet representation and the disambiguation algorithm – PageRank or ContextSimilarity.

2.4.11 Categorization

Categorization is an Enrycher Extraction Service, which provides document level attributes (see Figure 14). The service classifies the document into two taxonomies: Open Directory[6], a large human-edited dictionary with over a million categories, and the category schema from Wikipedia[7]. Additionally, the service uses these categorizations to extract a set of relevant keywords, describing the content of the document. The current implementation is based on a hierarchical centroid-based classifier described in [19], which is designed to scale well with respect the number of categories.

```
<attribute type="dmoz:topic" displayName="Topic on dmoz.org">Top/Sports/Soccer/…/World_Cup</attribute>
<attribute type="dmoz:topic" displayName="Topic on dmoz.org">Top/Shopping/Sports/Soccer</attribute>
<attribute type="rdfs:label" displayName="Keyword">Sports</attribute>
<attribute type="rdfs:label" displayName="Keyword">Soccer</attribute>
```

Figure 14 Example of categories and keywords assigned to a document by the Categorization service

2.4.12 Assertion Extraction

We define an assertion in a sentence as a relation between subject and object, the relation being the predicate. The aim is to extract sets of the form {subject, predicate, object} out of syntactically parsed sentences. The parsing of a sentence is done using the OpenNLP library, which has a parser based on a maximum entropy model, and outputs a parse tree in the Penn Treebank format.

[6] http://www.dmoz.org/
[7] http://en.wikipedia.org/wiki/Wikipedia:Categorization

We apply the algorithm for extracting subject – predicate – object sentence elements described in [27] and outlined in Figure 15.

To extract assertions, firstly we find the subject of the sentence. In order to find it, we are going to search in the NP sub-tree. The subject will be found by performing breadth first search and selecting the first descendent of NP that is a noun. Secondly, for determining the predicate of the sentence, a search will be performed in the VP sub-tree. The deepest verb descendent of the verb phrase will give the second element of the assertion. Thirdly, we look for objects. These can be found in three different sub-trees, all siblings of the VP sub-tree containing the predicate. The sub-trees are: PP (prepositional phrase), NP and ADJP (adjective phrase). In NP and PP we search for the first noun, while in ADJP we find the first adjective.

Moreover, for each element composing the triplet, we find its attributes (modifiers). For example, the attributes of a noun are mainly adjectives, the attributes of a verb are adverbs.

```
function ASSERTION-EXTRACTION(sentence) returns a solution, or failure
        result ← EXTRACT-SUBJECT(NP_subtree)
                ∪ EXTRACT-PREDICATE(VP_subtree)
                ∪ EXTRACT-OBJECT(VP_siblings)
        if result ≠ failure then return result
        else return failure

function EXTRACT-ATTRIBUTES(word) returns a solution, or failure
        // search among the word's siblings
        if adjective(word)
                result ← all RB siblings
        else
                if noun(word)
                        result ← all DT, PRP$, POS, JJ, CD, ADJP, QP, NP siblings
                else
                        if verb(word)
                                result ← all ADVP siblings
        // search among the word's uncles
        if noun(word) or adjective(word)
                if uncle = PP
                        result ← uncle subtree
        else
                if verb(word) and (uncle = verb)
                        result ← uncle subtree
        if result ≠ failure then return result
        else return failure

function EXTRACT-SUBJECT(NP_subtree) returns a solution, or failure
        subject ← first noun found in NP_subtree
        subjectAttributes ←EXTRACT-ATTRIBUTES(subject)
        result ← subject ∪ subjectAttributes
        if result ≠ failure then return result
        else return failure

function EXTRACT-PREDICATE(VP_subtree) returns a solution, or failure
        predicate ← deepest verb found in VP_subtree
        predicateAttributes ←EXTRACT-ATTRIBUTES(predicate)
        result ← predicate ∪ predicateAttributes
        if result ≠ failure then return result
        else return failure

function EXTRACT-OBJECT(VP_sbtree) returns a solution, or failure
        siblings ← find NP, PP and ADJP siblings of VP_subtree
```



```
for each value in siblings do
        if value = NP or PP
                object ← first noun in value
        else
                object ← first adjective in value
        objectAttributes ←EXTRACT-ATTRIBUTES(object)
result ← object ∪ objectAttributes
if result ≠ failure then return result
else return failure
```

Figure 15 The assertion extraction algorithm

2.4.13 Export to RDF

The Export to RDF is an Enrycher Transformation Service, which exports all semantic attributes identified in the document as a set of RDF triples. The document is represented either by the URI, given by the user, or by an automatically generated URN. The RDF schema used to in the exported RDF document is a combination of industry standard schemas (e.g. RDFS, OWL, and Dublin Core) and several Enrycher specific classes, defined at `http://enrycher.ijs.si/rdf/`.

Figure 16 shows a sample from Export to RDF module applied to example from annex A.1. The sample shows how document level attributes and annotations level attributes are exported. Enrycher by itself does remember documents and as such does not provide URIs for documents or annotations. This issue is resolved by using user provided URI to denote the document and by use of URNs to denote annotations. In the last part of the table we list assertions extracted directly from sentences.

Subject	Predicate	Object
http://example.com/doc/123	**dmoz:topic**	http://www.dmoz.org/Sports/Soccer/Competitions/World_Cup/
	enrycher:keyword	*"Sports*
	enrycher:text	*"Slovenia's dramatic win over Russia Wednesday…"*
	enrycher:pipeline	**urn:pipeline001**
urn:pipeline-001	**enrycher:hasService**	http://enrycher.ijs.si/service/raw2xml
	enrycher:hasService	http://enrycher.ijs.si/service/dmozCategorizer
http://example.com/doc/123	**enrycher:hasAnnotation**	**urn:annotation004**
urn:annotation004	**rdfs:type**	**enrycher:namedEntity**
	enrycher:hasInstance	*"Brazil"*
	dc:title	*"Brazil"*
	owl:sameAs	http://dbpedia.org/resource/Brazil
		http://sw.opencyc.org/concept/Mx4rvViPAZwpEbGdrcN5Y29ycA
	rdfs:label	*"Federal Republic of Brazil"*
urn:annotation113	**urn:annotation114**	**urn:annotation115**
urn:annotation117	**urn:annotation119**	**urn:annotation120**

Figure 16 Sample from Export to RDF module applied to example from annex A.1

2.4.14 Visualization

The assertions described in Section 2.4.12 can be visualized as a semantic graph. This type of visualization allows the user to see the text as a directed graph as shown in Figure 17. The nodes are important words and phrases occurring in the text. In our visualization, the yellow nodes represent verbs while the green nodes are nouns. The edges show subject → predicate (from green to yellow) and predicate → object (from yellow to green) relations. The orientation of the edges is determined by the fact that the edges are wider at the start than at the end. Three consecutive nodes where both the first and the last are nouns (green) make up a subject → predicate → object assertion.

Apart from the visual role, the semantic graph also has summarization purposes. To achieve this, a node ranking model has been learned using supervised learning. Ranking those assertions which compose the semantic graph, in decreasing order of importance has the goal of reducing the size of the semantic graph

while retaining the most important nodes. From the list of ranked assertions a smaller semantic graph of any chosen size can be built. The ranking method tried consists of training an SVM model for binary classification of assertions into important and not important. The 69 features used are of three kinds, as already proposed in [3]: document features (e.g. position of the sentence in the document, position of the assertion in the sentence, words in the assertion elements), linguistic features (e.g. part of speech tags, location of the assertion in the parse tree) and graph features (e.g. hub and authority weights, page rank, node degrees, connected components).

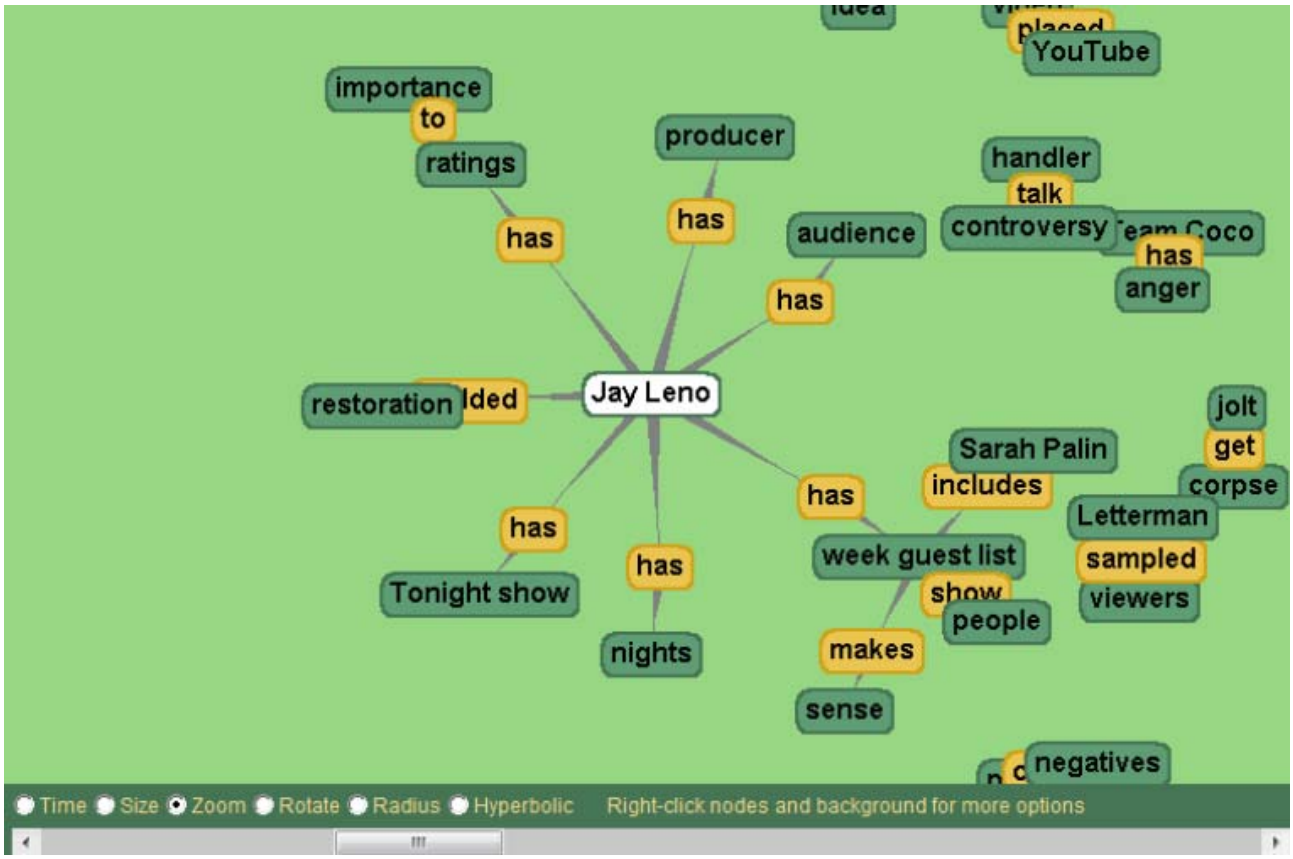


Figure 17 Semantic graph visualization

The semantic graph is displayed as a Java applet. For computing the layout and displaying the semantic graph we have used the TouchGraph[8] library.

For evaluating the visualization component, we selected three web pages: one Wikipedia article about a French nobleman, a news article about the effect of social network sites on television and a blog post which makes a parallel between playing blackjack and trading on the stock market, and asked seven users to rate several visualizations of these web pages, among which the semantic graph, based on how helpful they are for determining the content of the articles. They rated the visualization from 1 (not helpful) to 5 (very helpful). We refer to [28] for the details of the experiment. The semantic graph achieved the second best score (28 points, after the article summary which got 35 points). Users liked the interactive nature of the graph, and found it very useful for the page which focuses on a single topic.

2.5 Related Work

TextRunner [7] is an open information extraction system; it is open in the sense that it does not need a set of pre-defined relations, rather it learns these relations from text using a machine learning algorithm called Conditional Random Fields (CRF). The system was trained in a self-supervised manner by applying a number of relation-independent heuristics to the Penn Treebank in order to obtain a set of labeled examples.

[8] http://touchgraph.sourceforge.net

In the case of our Fact Extraction Service (Enrycher) relationship extraction is just one step. We provide a broader view on fact extraction, by identifying annotations, assertions and categories withing text. The annotations and categories provide additional semantic information compared to just relationship extraction.

Open Calais[9] is a semantic toolkit including a number of components. The Web Service API automatically annotates content with semantic metadata. The semantic metadata includes: a broad number of entities like city, currency, email address, product, programming language, etc., events and facts from a predefined set. It provides semantic metadata not only in English, but also in French and Spanish (for now only in the form of entities). The Calais Linked Data component provides Linked Data (Dbpedia, Wikipedia, Freebase, Reuters.com, GeoNames, Shopping.com, IMDB, LinkedMDB) resources for entities.

Table 5 A comparative view on five systems.
Our Fact Extraction Service, Text Runner, Open Calais, GATE and Read the Web

Features	Enrycher	Text Runner	Open Calais	GATE	NELL
Named Entity Extraction	✓		✓	✓	
Co-reference and Anaphora Resolution	✓		✓	✓	
Entity resolution	✓		✓		
Disambiguation	✓				
Assertion Extraction	✓	Relationship extraction	Events and Facts		Relationship extraction
Categories	✓		✓		✓
Vizualization	✓			✓	✓
RDF Output	✓		✓		
Multi-Language Support	English		English, French, Spanish[10]	✓	
Web Service API	✓		✓	✓	
Can work on a single document	✓		✓	✓	

The main difference between our Fact Extraction Service and Open Calais is that we do not have a pre-defined list of facts and events that we consider, but rather take a natural language processing approach and automatically extract what we call assertions from text by means of syntactic parsing and analysis. Moreover, we not only disambiguate the entities we extract, but try a holistic approach and link (at this point) all possible words/n-grams to WordNet (VUA) resources.

GATE (General Architecture for Text Engineering) [8] is a text analysis infrastructure for developing language processing components. GATE has several components: language resources – lexicons, corpora or ontologies, processing resources – parsers, generators, n-gram modellers and visual resources. Among the processing resources, GATE offers an Information Extraction (IE) system which performs tasks like sentence

[9] http://www.opencalais.com/
[10] For French and Spanish only Entity Extraction is available.

splitting, tokenization, phrase chunking, named entity extraction and co-reference and anaphora resolution.

In our ongoing work, we are using the GATE Information Extraction system for identifying named entities.

Read the Web [20] is a research project at Carnegie Mellon University with the purpose of building a never-ending machine learning system that extracts structured information from unstructured web pages. The system is called NELL (Never-Ending Language Learner). As inputs, NELL requires an ontology defining hundreds of categories (e.g. person, emotion) and relations (e.g. playsOnTeam(athlete,sportsTeam)) expected to be found in web pages, as well as 10 to 15 seed examples of each category/relation. The tasks performed by NELL are extracting new instances of categories and relations, while improving the learning abilities.

As opposed to Enrycher, NELL focuses on category and relationship learning, starting from an ontology and seed examples. As noted in the case of Text Runner, Enrycher also provides additional semantic information.

As a summary of the aforementioned remarks, we provide a comparative view on the four described systems and Enrycher (see Table 5).

3 News Fact Extraction Service

Enrycher as a fact extraction service can be applied to any English text. However, the data available in the research community, which was used to develop modules such as named entity and assertion extraction, is largely based on news articles. For example, the often used corpus for training statistical deep parsers was developed in the Penn Treebank Project[11], which is largely based on Wall Street Journal articles.

The news genre is also relevant within RENDER project, and as such we applied the Enrycher service to a Spinn3r stream. To achieve this we developed an infrastructure for connecting to a real-time stream of news articles, send a subset of articles from relevant mainstream media and blogs through Enrycher and exposed the extracted facts using a publish-subscribe interface.

The rest of this section describes the Spinn3r stream, which was partially detailed in D1.1.1 [21] and defines the interface which can be used by project partners to connect to the stream of annotated articles.

3.1 News Stream

Spinn3r is a start-up company from the United States focused on crawling and streaming mainstream news and social media. It monitors around 40 million blogs and 10,000 news outlets, resulting in up to 30M daily items. Any new items on the monitored sites are automatically crawled, (partly) cleaned, time-stamped and provided to their users as a continuous RSS feed.

Spinn3r API uses the ProtoBuffer protocol, defined by Google for efficient transfer of structured data such as XML. It can be accessed using a command-line Java client provided by Spinn3r. Figure 18 shows the loop used by the client to connect to the stream and download new items into a local directory.

```
LAST_RECEIVED_ITEM_TIME = Now – 10 minutes
NUMBER_OF_ITEMS = 1000
while (true) {
  NEW_ITEMS = receive(LAST_RECEIVED_ITEM_TIME, NUMBER_OF_ITEMS)
  save_as_xml(NEW_ITEMS, LOCAL_DIRECTORY + guid() + ".xml.gz")
  LAST_RECEIVED_ITEM_TIME = NEW_ITEMS.LAST.TIME
}
```

Figure 18 Pseudo code detailing the connection to the Spinn3r stream

Each item crawled by Spinn3r is described by, among others, the following fields:

- **Publisher type** – type of the source; possible types are mainstream news, blog, micro blog (e.g. Twitter) and social media (e.g. Facebook status updates)
- **Publisher URL** – address of the website which published the item
- **Language** – Language of the text in the item
- **Item URL** – URL from where the item was retrieved
- **Item Title** – Extracted from HTML Title field
- **Item Date** – date and time when the item was crawled

The client was tested at the Jožef Stefan Institute for more than a year and is proven to provide real-time news stream access. Technology was also developed for parsing and indexing the items in real-time. The whole system requires a high-end workstation to handle the load. Spinn3r offers special research licenses for their service. It is free for use in non-funded research projects and is offered for a monthly fee (at the time of writing it was 500 USD) when used in funded projects.

[11] http://www.cis.upenn.edu/~treebank/

In the beginning, only a selected subset of mainstream sources (around 100) will be fully processed with the Enrycher service. This number will be increased through the duration of the project based on the required processing costs, available resources and requirements from other technical work packages and case studies.

3.2 Architecture and Interface

The system is built around the IJS system for stream processing and mining, used in several other large-scale installations requiring near real-time processing and response when handling millions of events per day [22]. Figure 19 presents the architecture of the News Fact Extraction Service, which consists of the following elements:

- **IJS News Service** – server responsible for receiving and cleaning the news feed from Spinn3r, sending it through Enrycher and storing it in the internal database.
- **Enrycher** – an instance of the Fact Extraction Service described in Section 2.
- **Web-Service API** – an API exposing the internal database of the IJS News Service
- **Pub-sub bus** – a publish/subscribe interface providing external clients with the ability for subscribing to the real-time feed of processed news articles. The articles can be either in Enrycher XML format (as described in 2.2) or RDF format (result of the transformation service described in 2.4.13).
- **Demo Website** – A website exposing several web-service calls (e.g. list of news sources, content and entity search) and a scrolling list of latest articles published on the pub-sub bus.

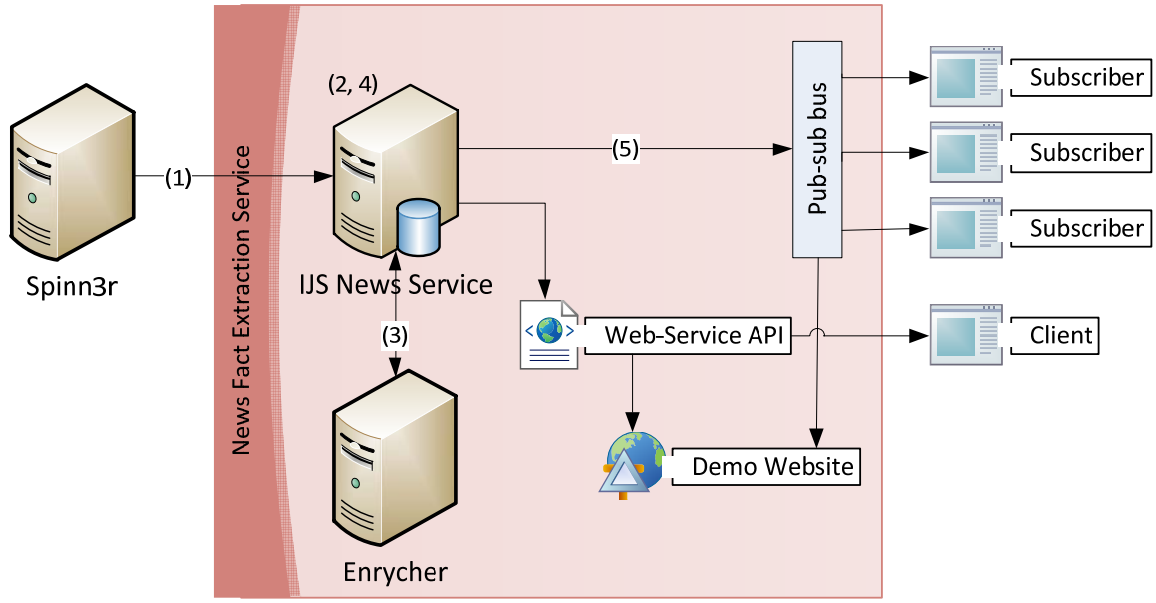


Figure 19 The Architecture of the News Fact Extraction Service

Each news item goes through the following pipeline (stage numbers are also noted in Figure 19):

1. News Service collects the Spinn3r item using the procedure described in Section 3.1.
2. News Service filters the items according to their news source (only the items coming from approved news sources are kept) and extracts meta-data and clean text from each of the items.
3. Clean text is sent through the Enrycher Fact Extraction Service for additional information extraction (annotations, assertions, categories).

4. Each news item is indexed along several dimensions: meta-data, content, annotations, assertions.
5. The item is published to the pub-sub bus.

The following functionality is exposed through the web-service API:

- Subscription to published items. This is done by registering to an URL, to which a HTTP POST request is made for each new item. The body of the request contains the processed news item.
- Search over stored news items along all indexed dimensions.
- List of available news sources and their statistics.

4 Article Template Discovery

While there is presently a lot of interest in structuring text by annotating and identifying a specific subset of information, mostly named entities, little work has been done on semantically capturing the *macro*-level aspects of the text. In this section, we present some early work on constructing *domain templates*, a generic ``summary'' that fits many pieces of text on a specific theme (e.g. news stories about bombings) at the same time.

The genericness of the template provides for data exploration in two ways:

- By automatically mapping specific facts and entities in an article to the more general ones in a template, we are providing structure to the articles as entities from potentially many articles get mapped to a single semantic ``slot'' in a single template.
- By (possibly statistically) inspecting all the articles that were mapped to a chosen template, we can observe the diversity of articles *in a specific aspect*, exploiting the fact that they are semantically aligned to an extent. For example, if the template contains the statement `happen_at..location`, we can find with no further processing the specific locations which the template-mapped articles describe.

To determine entities and relations that subsume those from individual news articles and thus construct a template, we make use of a general-purpose ontology, in our case WordNet. To represent the templates as well as individual news stories, we use *semantic graphs*, i.e. graphs in which entities represented as nodes and the (binary) relationships connecting them are represented as edges.

Domain specifics

Note that media companies have a large interest in semantically annotating text, particularly news items. For this reason, and because of easy availability of datasets, we focus on the domain of newswire in this discussion. Despite this, there is in principle nothing specific this domain that would limit the applicability of our method to it. In general, the required input data is a collection of text items which are assumed to discuss roughly the same aspects of a single topic. Examples of such collections are ``news articles about earthquakes'', ``Wikipedia articles on football players'' or ``smartphone product reviews''.

Related work

Because it aligns articles to a common template, our method has much in common with other information extraction mechanisms. Automatic construction of information extraction templates is already relatively well-researched. Most methods aim for *attribute extraction*, where the goal is to extract a single predefined type of information, e.g. the title of a book. Each separate type of information requires a separate classifier and training data. Examples of such approaches are [10] [11].

More recently, a generalized problem of *relation extraction* has received considerable attention. The goal there is to find *pairs* of items related by a predefined relation. As an example, Probst et al. [14] mine product descriptions to simultaneously identify product attributes and their values. Relation extraction is particularly popular in biomedicine where pairs of proteins in a certain relation (e.g. one inhibits the other) are often of interest.

The task in this article is more generalized still; we attempt to decide both what information is interesting to extract as well as perform the extraction. This is known as *domain template extraction*. To our knowledge, little work has been done in the area so far. The most closely related work is by Filatova et al. [15], who find templates by mining frequent parse sub-trees. Also closely related is work by Li et al. [16]; similarly to Filatova, they mine frequent parse sub-trees but then cluster them into ``aspects'' with a novel graphical model. Both approaches produce syntax-level patterns. Unlike ours, neither of the two approaches exploits background knowledge. Also belonging to this group is our previous work [17] which mostly shares the goal and data representation ideas with this article, but uses different methods apart from pre-processing.

Graph-based templates are also used in [18] in a context similar to ours, though the semantics are shallower. Also, the authors focus on information extraction and do not attempt to generalize the templates.

Templates somewhat similar to those we aim to construct automatically and with no knowledge of the domain have already been created manually by domain experts. FrameNet is a collection of templates for the events like "disclosing a secret", "speaking", "killing", "arresting" etc. They focus mostly on low-level events, of which typically many can be found in a single document, be it a news article or not. The project does not concern itself with the creation of the templates, other than from the methodological point of view. There is little support for automatic annotation of natural language with the FrameNet frames.

4.1 Method Overview

In this section, we cover all the stages of our data processing pipeline. As discussed above, the assumed input data is a collection of text items discussing the same topic and the goal is to identify a pattern matching a substantial number of the input text items.

The key idea is rather simple: we first represent input data as semantic graphs, i.e. graphs of ontology-aligned entities and relations. Then, we define a pattern to be a (smaller) graph such that, by specializing some of its entities, we can obtain a sub-graph of at least T input graphs (T being a parameter). We seek to identify all such patterns.

We proceed to describe our approach to the construction of semantic graphs and to the mining of approximate sub-graphs.

4.1.1 Semantic Graph Construction

Starting from the assertions described in previous sections, we first align all assertion terms to WordNet; that is, for each subject, verb and object appearing in any of the assertions, we try to find the corresponding concept (or "synset", as they are called) in WordNet. We first remove inflection from the words using python NLTK (Natural Language Toolkit)[12], then align it to the corresponding synset. If more than one synset matches, we choose the most common sense which is a well-tried and surprisingly good strategy. For words not found in WordNet, we create a new synset on the fly. If the new word (e.g. ``Obama'') was previously tagged by ANNIE (with e.g. ``person''), the new synset's hypernym is set accordingly.

As we started our research on this topic in parallel to the development of the Enrycher service components, we refer to the previous implementation of the service components, which was based on the python NLTK.

From a collection of assertions, we proceed to construct the semantic graph. Here, we rely rather heavily on the fact that news articles tend to be focused in scope: we do not disambiguate entities other than by name (not necessarily a proper name; e.g. ``book'' is also a name). As an example, if an article mentions two buildings, one of which burns down and the second of which has a green roof, our method detects a single ``building'' and assigns both properties to it. In the newswire domain, we have not found this to be a significant issue: entities which do need to be disambiguated are presented with more unique names (``France'' instead of ``country'' etc.). This rationale would have to be revised if we wanted to apply the approach to texts with a broader scope (e.g. comparative product reviews).

This assumption greatly simplifies the construction of the semantic graph: we start by treating each assertion as a 2-node component of a single very fragmented graph and then collapse the nodes with the same labels.

Dataset specifics

In our experiments, each input ``document'' in the sense described here was in fact a concatenation of actual documents, all of which were reporting on the exact same news event. The Experiments section contains the details and rationale.

[12] http://www.nltk.org/

4.2 Approximate Pattern Detection

Given a collection of labelled graphs, we now wish to identify frequent ``approximate sub-graphs'', i.e. patterns as described at the beginning of Section 4.3.

Formal task definition: Given a set of labelled graphs $S = \{G_1,\ldots,G_n\}$, a transitive anti-symmetric relation on graph labels *genl*(·,·) (with *genl*(A,B) interpreted as ``label A is a generalization of label B'') and a number T, we wish to construct all maximal graphs H that are *approximate sub-graphs* of at least T graphs from S. A graph H is said to be an approximate sub-graph of G if and only if there is a mapping f of V(H) onto a subset of V(G) such that genl(v,f(v)) holds for all v from V(H).

This is not an easy task. Mining frequent sub-graphs is in itself computationally demanding because of the inherently intractable graph isomorphism we have to handle; satisfactorily fast algorithms for this seemingly basic problem are relatively recent [13]. By introducing another degree of freedom, i.e. requiring that the frequent sub-graph only match the input graphs in a soft way implied by a taxonomy (here WordNet hyperymy), the complexity becomes insurmountable. We compensate by introducing two assumptions.

- The hierarchy imposed by genl has a tree-like form, it is not a general DAG. This is true of WordNet: every synset has at most one hypernym defined.
- Very generic patterns are not interesting and can (or even should) be skipped. This too is a safe assumption in our scenario: a pattern in which every node is labelled with the most generic label `entity` is hardly informative regardless of its graph structure.

We can now employ a simple but effective three-stage search. The stages are illustrated in Figure 20 with the minimal example of two two-node graphs.

- Generalize all the labels of input graphs to the maximum extent permissible. Under the first assumption, ``generalizing a label'' is a well-defined operation. The exact meaning of ``maximum extent permissible'' is governed by the second assumption; no label should be generalized so much as to fall in the uninteresting category. In our experience with WordNet, the following simple rule worked very well: generalize verbs as much as possible and generalize nouns to two levels below the hierarchy root. See steps 1 to 2 in Figure 20.
- Mine T-frequent maximal sub-graphs with support of the generalized input graphs. This step cannot be shown in Figure 20 as the graphs are too small.
- Formally, the resulting sub-graphs already satisfy our demands. However, to make them as descriptive as possible, we try to specialize the pattern's labels, taking care not to perform a specialization that would reduce the pattern's support below T. Specialization, unlike generalization, is not a uniquely defined operation (a synset can have many hyponyms). Luckily, the patterns are small and with some care we can afford to recursively explore the whole space of possible specializations. We use the sum of labels' depth in the WordNet hierarchy as a measure of pattern descriptiveness that we optimize. See steps 2 to 3 in Figure 20.

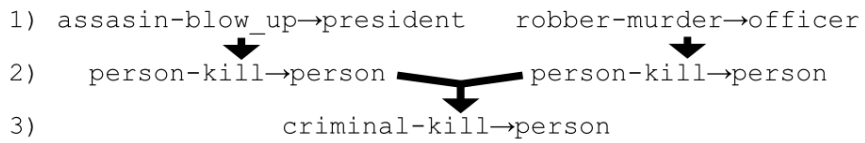


Figure 20 Generalization of input graphs and re-specialization of the pattern

For frequent sub-graph mining, we developed our own algorithm, inspired by the current state-of-art [12] [13]. We included some improvements pertaining to maximality of output graphs and to scalability -- all existing open-source software crashed on our full input data.

4.3 Preliminary Experiments

As a preliminary, let us define some terminology suitable for our experiment domain. An *article* is a single web page which is assumed to report on a single *story*. A story is an event that is covered by one or more articles. Each story may fit some *domain template* (also *event template* or simply *template*) describing a certain type of event.

We obtained a month's worth of articles from Google News by crawling. Each article was cleaned of all HTML mark-up, advertisements, navigation and similar. Articles were grouped into stories according to Google News.

For each *story*, a semantic graph was constructed. The reason to use an aggregate story graph rather than individual article graphs was twofold. First, by representing each story as a single graph, all stories were represented equivalently (as opposed to the case where each article contributed a graph, resulting in stories being weighted proportionally to the number of their articles). Second, the method for extracting assertions has relatively low precision and recall; it therefore makes sense to employ the redundancy inherent in the collection of articles reporting on the same event. To construct the aggregate story graph, we simply concatenated the plain text of individual articles; aggregation at this early stage has the added benefit of providing cross-article entity resolution. Finally, the collection of semantic graphs from stories on a single topic was input to the pattern mining algorithm.

We defined five topics on which to observe the behaviour of the method: bomb attacks, award ceremonies, worker layoffs, political visits and court sentencings. For each, we identified about 10 stories of interest. Note that each story further comprises about 100 articles, clustering courtesy of Google News; in total, about 5000 articles were therefore processed.

As semantic graphs were constructed on the level of stories rather than articles, their structure was relatively rich. They had about 1000 nodes each and an average node degree of roughly 2.5. The 20% most connected nodes, which are also the ones likely to appear in the patterns, had an average degree of about 20.

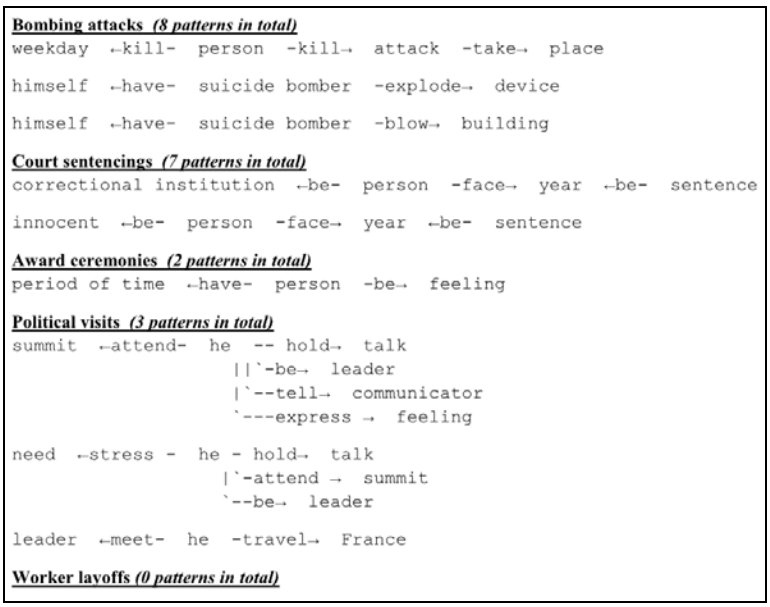


Figure 21 A selection of detected patterns for each of the domains

For each topic, graphs of all its stories were input to the algorithm. The minimal pattern support was set at 30% for all the topics. The algorithm output several patterns for each topic; the sizes of the outputs along with the interesting patterns are presented in Figure 21.

For instance, the last person in the ``visits'' domain shows that in at least 30% of the stories, there was a male person (``he'', e.g. Obama) who travelled to France (a coincidence), and that same person met a ``leader'' (a president in some of the stories, a minister in other).

4.4 Discussion and Future Work

The preliminary results seem sound. The mappings of individual stories onto the patterns (not given here) also provide a semantically correct alignment. We can observe how each story fits the template with slightly different entities. Sometimes, the variations are almost imperceptible -- ``correctional facility'' from the ``court'' domain, for example, appears as either ``jail'' or ``prison'', which for some reason are two distinct concepts in WordNet.

Again in other cases, the distinctions are significant and express the sub-topical diversity we were looking for. For example, the groundings for ``leader'' in the ``visits'' domain varied even in our small dataset over president, minister, instigator or simply leader. In the same domain, ``feeling'' was either sorrow, disappointment or satisfaction. The ``building'' in the ``bombings'' domain was generalized from mosque, restaurant, hotel and building. It might be interesting to investigate this further and use the amount of variation between pattern groundings as a measure of pattern interestingness.

Unexpectedly, diversity can occasionally be found in the natural clustering that the patterns provide. Observe the two patterns in the ``court'' domain: in both, the defendant is facing a sentence of (one or more) years, but is found innocent in one cluster and sent to (?) the jail in the other.

While the current experiments are too small to draw any conclusive evidence, we can make some speculations about precision and recall. While the first is low but usable (a data analyst should not mind going through e.g. 5 patterns to identify a useful one), the latter seems a bigger issue. We hope to improve the results significantly by developing a better fact/assertion extractor; the previously discussed deficiencies of current assertions appear to have the worst hit on performance.

The tests also indicate that the method is not equally suitable for all domains. The ``layoffs'' domain, for example, had no single pattern which would occur in 30% of the stories. (A threshold of 25% produces a single rather nonsensical pattern ``it --- cut → job ← lose --- people''). The ``awards'' domain does not fare much better. Most probably, these two topics are too broad, causing stories to have only little overlap.

Note that in current implementation, all final patterns with less than three nodes (e.g. `worker..lose..job` for the ``layoffs'' topic) were discarded. Partly this is because we are, in perspective, interested in (dis)proving that structured patterns can provide more information than sentence-level patterns found in related work[13]. Partly, however, it is also because including two-node patterns would introduce additional noise in the output. Even now, the precision is relatively low; it would therefore be interesting to investigate measures of interestingness of patterns other than raw frequency.

[13] The ``visits'' domain is a nice indication that this may be true.

5 Conclusions

In this deliverable we presented a prototype of the Fact Mining Toolkit, which handles fact extraction and is designed as a collection of multiple services. In order to exemplify the service usage, we described a News Fact Extraction service which applies fact extraction on a stream of selected news articles. We also elaborated on on-going research work in the direction of improved knowledge fact extraction based on extracting fact templates from multiple documents. Both the Fact Extraction Service and the News Fact Extraction Service were described from an architecture and interface perspective, while detailing each of their components individually.

As an end result of this deliverable, we provide our project partners a web service API for interacting with Enrycher, our Fact Extraction Service. Moreover, we release a news corpus composed of selected items processed with Enrycher, which the partners can subscribe to.

The Fact Extraction Service was evaluated on a component basis – and we provided a summary of the evaluation results for the respective components.

To sum up, the following components were developed within the RENDER project:

- New Enrycher services: Disambiguation Service, Export to RDF
- Upgraded Enrycher services: Text Pre-processing, Named Entity Extraction, Assertion Extraction
- A consolidated system architecture limiting core services to Java or C++
- Enrycher object model and wrappers for the RESTful Web Service, implemented in Java and C++
- The News Fact Extraction Service: an application of Enrycher to a stream of news articles
- On-going research work in the lines of improving fact extraction by identifying fact templates across multiple documents

As future work we plan to finish evaluating all the components (for e.g. assertion extraction) as well as the entire system as a whole, by taking into consideration the evaluation metrics provided by the case studies. Our on-going research on fact template extraction will be implemented and integrated in the next version of the Fact Mining deliverable. On the annotations side, we plan to extend our disambiguation service by providing more disambiguated resources from other datasets aside from WordNet (VUA); moreover, we are preparing two more Named Entity Extraction services: one based on GATE and another one based on the Stanford Named Entity Recognizer. On the assertions side, we are working on improving our current assertion extraction algorithm by analysing more sentence patterns, including negation and passive voice.

References

[1] D. D. Lewis, Y. Yang, T. G. Rose, F. Li. *RCV1: A New Benchmark Collection for Text Categorization Research.* Journal of Machine Learning Research, Vol. 5, 2004.

[2] J. Lescovec, M. Grobelnik, N. Milic-Frayling. *Impact of linguistic analysis on the semantic graph coverage and learning of document extracts*. In Proceeding of the 12th National Conference On Artificial Intelligence, pp. 1069–1074, 2005.

[3] D. Rusu, B. Fortuna, M. Grobelnik and D. Mladenic. *Semantic Graphs Derived From Triplets with Application in Document Summarization***.** Informatica Journal 33, no. 3, pp. 357 – 362, 2009.

[4] Ch. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[5] S. Brin and M. Page. *Anatomy of a large-scale hypertextual Web search engine*. In Proceedings of the 7th Conference on World Wide Web (WWW). Brisbane, Australia, 1998.

[6] R. Navigli, K. C. Litkowski, and O. Hargraves. *Semeval-2007 task 07: Coarse-grained English allwords task*. In Proceedings of the 4th International Workshop on Semantic Evaluations, Prague, Czech Republic, 2007.

[7] M. Banko and O. Etzioni. *The Tradeoffs Between Open and Traditional Relation Extraction*. In Proceedings of the Association for Computational Linguistics, Columbus, Ohio, 2008.

[8] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan. *GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications*. Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). Philadelphia, July 2002.

[9] J. R. Finkel, T. Grenager, and C. Manning. *Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling.* Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370, 2005.

[10] A. Arasu and H. Garcia-Molina. *Extracting structured data from web pages*. pages 337–348, 2003.

[11] S. Brin. *Extracting patterns and relations from the world wide web*. Lecture Notes in Computer Science, pages 172–183, 1999.

[12] Y. Chi, S. Nijssen, R. Muntz, and J. Kok. *Frequent subtree mining-an overview*. Fundamenta Informaticae, 66(1):161–198, 2005.

[13] X. Yan and J. Han. gSpan: *Graph-based substructure pattern mining*. page 721, 2002.

[14] K. Probst, R. Ghani, M. Krema, A. Fano, and Y. Liu. *Semi-supervised learning of attribute-value pairs from product descriptions*. pages 2838–2843, 2007.

[15] E. Filatova, V. Hatzivassiloglou, and K. McKeown. *Automatic creation of domain templates*. In Proceedings of COLING/ACL 2006, Association for Computational Linguistics, pages 207–214, Morristown, NJ, USA, 2006.

[16] P. Li, J. Jiang, and Y. Wang. *Generating templates of entity summaries with an entity-aspect model and pattern mining*. Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 640–649, 2010.

[17] M. Trampus and D. Mladenic. *Learning event templated from news articles*. In Proceedings of SiKDD09, 2009.

[18] H. Tanev and B. Magnini. *Weakly supervised approaches for ontology population*. 2006.

[19] M. Grobelnik, D. Mladenić. Simple classification into large topic ontology of Web documents. ITI, Cavtat, Croatia, 2005.

[20] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr. and T.M. Mitchell. *Toward an Architecture for Never-Ending Language Learning.* In Proceedings of the Conference on Artificial Intelligence (AAAI)*,* 2010.

[21] A. Kiryakov, M. Grinberg, M. Damova, B. Fortuna, M. Trampus. *Initial collection of data*. RENDER Project Deliverable D1.1.1. 2011.

[22] B. Fortuna, C. Fortuna and D. Mladenić. Real-time News Recommender System. Demo at ECML/PKDD, Barcelona, Spain, 2010.

[23] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Lecture Notes in Computer Science, vol. 4825, p. 722, 2007.

[24] F. Suchanek, G. Kasneci, and G. Weikum. *Yago: a core of semantic knowledge*. In Proceedings of the 16th international conference on World Wide Web, pp. 697-706, ACM New York, NY, USA, 2007.

[25] T. Stajner and D. Mladenic. *Entity Resolution in Texts Using Statistical Learning and Ontologies*. In proceedings of the 3rd Asian Semantic Web Conference, Shanghai, 2009.

[26] T. Stajner, D. Rusu, L. Dali, B. Fortuna, D. Mladenic and M. Grobelnik. *Enrycher: Service Oriented Text Enrichment.* Informatica Journal 34, no. 3, pp. 307 -- 313. 2010.

[27] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik and D. Mladenic. *Triplet extraction from sentences*. In Proceedings of the 10th International Multiconference "Information Society - IS 2007". Ljubljana, Slovenia. pp. 218 – 222, 2007.

[28] L. Dali and D. Mladenic. *Visualization of Web Page Content Using Semantic Technologies.* In Proceedings of Information Visualization, July 27th, London, UK, 2010.

[29] D. Rusu, B. Fortuna and D. Mladenic. *Automatically Annotating Text with Linked Open Data*. Linked Data on the Web Workshop, the World Wide Web Conference, Hyderabad, India, 2011.

Annex A Enrycher Object Model

A.1 Example of XML format

```
<item id="6210">
  <metadata>
    <semantics>
      <attribute type="dmoz:topic" displayName="Topic on dmoz.org">Top/Sports/Soccer/…/World_Cup</attribute>
      <attribute type="dmoz:topic" displayName="Topic on dmoz.org">Top/Shopping/Sports/Soccer</attribute>
      <attribute type="rdfs:label" displayName="Keyword">Sports</attribute>
      <attribute type="rdfs:label" displayName="Keyword">Soccer</attribute>
      <!-- ... -->
    </semantics>
    <pipeline>
      <tool time="2010-12-31T12:44:40" durationTimeMillis="102">Raw to XML converter</tool>
      <tool time="2010-12-31T12:45:06" durationTimeMillis="839">Categorizer</tool>
      <tool time="2010-12-31T12:46:07" durationTimeMillis="106">Named-Entity Extraction (Stanford)</tool>
      <tool time="2010-12-31T12:46:33" durationTimeMillis="53">Co-reference resolver</tool>
      <tool time="2010-12-31T12:46:42" durationTimeMillis="835">Triplet Extractor</tool>
      <tool time="2010-12-31T12:47:84" durationTimeMillis="156">Semantic entity resolution</tool>
    </pipeline>
  </metadata>
  <text>
    <title>
      <sentence id="0">
        <plainText>Slovenia's dramatic win over Russia Wednesday, and to a lesser extent …</plainText>
        <tokens>
          <token id="0.0" pos="NNP">Slovenia</token>
          <token id="0.1" pos="POS">&amp;apos;s</token>
          <token id="0.2" pos="JJ">dramatic</token>
          <token id="0.3" pos="NN">win</token>
          <!-- ... -->
        </tokens>
      </sentence>
    </title>
    <p>
      <sentence id="1">
        <plainText>After a century of near domination from the likes of Brazil, Italy and …</plainText>
        <tokens>
          <token id="1.0" pos="IN">After</token>
          <token id="1.1" pos="DT">a</token>
          <token id="1.2" pos="NN">century</token>
          <!-- ... -->
        </tokens>
      </sentence>
      <sentence id="2">
        <plainText>It may not happen this time around, but given the increasing flow of …</plainText>
        <tokens>
          <token id="2.0" pos="PRP">It</token>
          <token id="2.1" pos="MD">may</token>
          <token id="2.2" pos="RB">not</token>
          <!-- ... -->
        </tokens>
      </sentence>
    </p>
    <!-- ... -->
  </text>
  <annotations>
    <annotation id="4" displayName="Brazil" type="enrycher:namedEntity">
      <instances>
        <instance id="5" words="Brazil">
          <token id="1.10"/>
        </instance>
      </instances>
      <semantics>
        <attribute type="dc:title">Brazil</attribute>
        <attribute type="owl:sameAs" resource="http://dbpedia.org/resource/Brazil"></attribute>
        <attribute type="owl:sameAs" resource="http://sw.opencyc.org/concept/Mx4rvViPAZwpEb…"></attribute>
        <attribute type="rdf:type" resource="http://dbpedia.org/…/yago/FormerPortugueseColonies"></attribute>
        <attribute type="rdf:type" resource="http://dbpedia.org/class/yago/G15Nations"></attribute>
        <attribute type="rdfs:label">Brasil</attribute>
        <attribute type="rdfs:label">Brazil</attribute>
        <attribute type="rdfs:label">Federal Republic of Brazil</attribute>
        <attribute type="rdfs:label">Federative Republic of Brazil</attribute>
```

```
        <!-- ... -->
      </semantics>
    </annotation>
    <annotation id="18" displayName="Shay Given" type="enrycher:namedEntity">
      <instances>
        <instance id="24" words="Shay Given">
          <token id="8.6"/>
          <token id="8.7"/>
        </instance>
      </instances>
      <semantics>
        <attribute type="dc:title">Shay Given</attribute>
        <attribute type="owl:sameAs" resource="http://dbpedia.org/resource/Shay_Given"></attribute>
        <attribute type="rdf:type" resource="http://dbpedia.org/…/…/BlackburnRoversF.C.Players"></attribute>
        <attribute type="rdfs:label">Shay Given</attribute>
        <!-- ... -->
      </semantics>
    </annotation>
    <annotation id="49" displayName="Tommy Smyth" type="enrycher:namedEntity">
      <instances>
        <instance id="64" words="Tommy Smyth">
          <token id="20.21"/>
          <token id="20.22"/>
        </instance>
      </instances>
      <semantics>
        <attribute type="dc:title">Tommy Smyth</attribute>
        <attribute type="owl:sameAs" resource="http://dbpedia.org/resource/Tommy_Smyth"></attribute>
        <attribute type="rdf:type" resource="http://dbpedia.org/class/yago/FootballAnnouncers"></attribute>
        <attribute type="rdfs:label">Tommy</attribute>
        <attribute type="rdfs:label">Tommy Smyth</attribute>
        <!-- ... -->
      </semantics>
    </annotation>
    <!-- ... -->
  </annotations>
  <assertions>
    <assertion id="8">
      <subject annotationId="113" instanceId="0" displayName="Slovenia"/>
      <verb annotationId="114" instanceId="0" displayName="has"/>
      <object annotationId="115" instanceId="0" displayName="Valter Birsa Wednesday">
        <modifier annotationId="83" instanceId="0" displayName="&apos;s dramatic"/>
      </object>
    </assertion>
    <assertion id="9">
      <subject annotationId="117" instanceId="0" displayName="Irish">
        <modifier annotationId="126" instanceId="0" displayName="in the"/>
      </subject>
      <verb annotationId="119" instanceId="0" displayName="exerted"/>
      <object annotationId="120" instanceId="0" displayName="pressure">
        <modifier annotationId="121" instanceId="0" displayName="intense"/>
      </object>
    </assertion>
    <assertion id="10">
      <subject annotationId="122" instanceId="0" displayName="star Zinedine Zidane">
        <modifier annotationId="123" instanceId="0" displayName="retired French"/>
      </subject>
      <verb annotationId="124" instanceId="0" displayName="watched"/>
      <object annotationId="125" instanceId="0" displayName="crowd">
        <modifier annotationId="126" instanceId="0" displayName="in the"/>
      </object>
    </assertion>
    <assertion id="11">
      <subject annotationId="127" instanceId="0" displayName="star Thierry Henry">
        <modifier annotationId="123" instanceId="0" displayName="retired French"/>
      </subject>
      <verb annotationId="129" instanceId="0" displayName="controlled"/>
      <object annotationId="130" instanceId="0" displayName="kick">
        <modifier annotationId="131" instanceId="0" displayName="a free"/>
      </object>
    </assertion>
    <!-- ... -->
  </assertions>
</item>
```