

DELIVERABLE SUBMISSION SHEET

To: Rainer TYPKE *(Project Officer)*

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

From: Eoin Kilfeather

Project acronym: Decipher Project number: 270001

Project manager:

Project coordinator Eoin Kilfeather

The following deliverable:

Deliverable title: DECIPHER Semantic Annotator

Deliverable number: D4.3.1

Deliverable date: 31 August 2013

Partners responsible: BUT

Status: Public Restricted Confidential

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

The deliverable is:

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

Sent to Project Officer: <i>Rainer.TYPKE@ec.europa.eu</i>	Sent to functional mail box: <i>INFSO-ICT-270001@ec.europa.eu</i>	On date: 06 November 2013
--	--	---------------------------------

DECIPHER Semantic Annotator

Document identifier: Decipher-D4.3.1-WP4-BUT Semantic Annotator-PU

Project title: Digital Environment for Cultural Interfaces;
Promoting Heritage, Education and Research

Project acronym: DECIPHER

Project identifier: FP7-270001-Decipher

Partners: Dublin Institute of Technology
National Gallery of Ireland
Irish Museum of Modern Art
Open University
System Simulation Limited
Brno University of Technology
Royal Irish Academy

Author(s): Pavel Smrz, Lubomir Otrusina, Jan Kouril, and
Jaroslav Dytrych
Brno University of Technology

Version: v01

Type: Report (Deliverable D4.3.1)

Report availability: Public

Date: August 31, 2013



Contents

1	Introduction	1
2	Individual data sources and their processing	3
2.1	Getty ULAN	3
2.2	Freebase	4
2.3	Wikipedia/DBpedia	5
2.4	Geonames	8
3	Semantic integration of datasets	9
3.1	Data consistency issues and overlaps	9
3.2	Integrated knowledge base	10
4	Disambiguation and co-reference resolution	13
4.1	Disambiguation data and processes	13
4.2	Co-reference resolution	14
5	Resulting component and its evaluation	17
5.1	Semantic annotation module	17
5.2	Performance and coverage evaluation	19
6	SEC API extensions	22
6.1	Service <code>annotate</code>	22
6.2	Service <code>get_entities</code>	24
6.3	Service <code>get_entity_types_and_attributes</code>	26
7	Semantically enriched fulltext search	27
7.1	Metadata indexing	27
7.2	Querying	28
8	Annotation suggestions	31
8.1	System architecture	32
8.2	Computer-assisted annotation of text	35
9	Conclusions	38

1 Introduction

DECIPHER builds on stories that are entered by museum professionals and used in the whole range of narrative construction and knowledge visualisation. To be able to do this, the system needs to “understand” the content – to transform free-text narratives into a structured form with explicitly identified semantics. The process can be manual (for example, entering individual events with all their attributes through a user form in Storyscope) but it can also be assisted by automatic semantic enrichment of the textual content. Various tasks can benefit from the automatic approach – events and their attributes can be automatically extracted from texts, similarities among stories can be evaluated with the help of identified entities and relations, and advanced search interfaces can employ automatically extracted metadata.

This deliverable deals with the automatic information extraction from text. It describes the final version of the DECIPHER Semantic Annotator – the core of the Semantic Enrichment Component (SEC) responsible for generating automatic annotations of input texts that identify domain-related entities and their inter-relations.

The document follows the previous WP4 deliverable – D4.2.1: Relationship mining component. It builds on the architecture of the SEC introduced in the deliverable and details enhancements of the Information Extractor that is invoked by other subcomponents of the SEC as well as other modules of the DECIPHER system as a whole.

In particular, improvements of the Named Entity Recognizer (NER) are discussed, together with an evaluation of resulting services on relevant datasets. We pay attention to resources used to populate lists of named entities and the process of semantic integration of various data they contain. In addition to their domain relevance and coverage, data sources are appraised in terms of their compatibility with the Linked Open Data (LOD) – the very basis of the current web-semanticalisation trend. In this sense, even high-coverage resources such as ULAN – the Union List of Artist Names collected by the Getty Research Institute – which are not available in a LOD-compatible form – have significant drawbacks when compared to Freebase, DBpedia and similar sources.

Having hundreds of thousands of potentially recognizable entities (artworks, artists, places), it is also necessary to disambiguate names that can refer to more than one entity. Static and dynamic disambiguation processes that are employed in the Semantic Annotator are also discussed, as well as additional functions of the tool – domain-specific co-reference resolution and identification of dates, intervals, and monetary values.

In addition to the DECIPHER Content Aggregator and other independent components, annotations play a key role in the semantically enriched fulltext search and in the process generating suggestions for manual anno-

tation of resources. Both the applications demonstrating advanced features of the Semantic Annotator are also described in this document.

The most significant result of the work discussed in this deliverable consists in speed advances and higher coverage of the DECIPHER Semantic Annotator when compared to state-of-the-art systems such as DBpedia Spotlight¹. Resulting datasets and tools are made publicly available to encourage further research in semantics enrichment of the cultural heritage content.

¹<http://spotlight.dbpedia.org/>

2 Individual data sources and their processing

When building a semantic annotator, one needs to identify information sources providing primary data first. Various resources have been explored within the DECIPHER project for this purpose. The list of key datasets employed in the final version of the DECIPHER Semantic Annotator includes Getty ULAN², Freebase³, Wikipedia⁴/DBpedia⁵, and Geonames⁶. This section discusses specific content of the resources as well as procedures applied to transform them into a unified representation.

2.1 Getty ULAN

The Union List of Artist Names (ULAN) is a structured vocabulary that aims at improving access to information about art, architecture, and material culture. The ULAN contains names and additional information referring to two types of entities – individual artists (persons) and collective bodies, including records such as *Museo del Prado* or *de Badajoz family – Spanish family of architects, active 16th century*, cumulated from nine participating Getty documentation projects. Roles of individuals cover a variety of visual art branches, including performance artists, architects, and decorative artists.

The ULAN is available on-line, allowing searching for artists and retrieving their variant names, biographical information, and bibliographic citations. BUT acquired a licence of the resource to employ it in the DECIPHER and other projects.

Although the ULAN dataset contains inconsistencies and errors (see Section 3.1 for more details), it formed a fundamental data source for lists of visual artists and relevant collective bodies integrated into the Semantic Annotator. We extracted 181,718 unique entities representing individuals and 39,387 collective bodies. Generic person names such as *Unknown Adventist* or *Monogrammist M. K.* were filtered out.

For each ULAN record, the following fields were extracted: ID, display term, preferred term, other terms, preferred role, other roles, preferred nationality, other nationalities, description, date of birth, date of death, place of birth, place of death, gender and note. According to the official documentation of the source, the same set of fields is used for collective bodies with corresponding meaning – for example, date of opening of a museum.

²<http://www.getty.edu/research/tools/vocabularies/ulan/>

³<http://www.freebase.com/>

⁴<http://www.wikipedia.org/>

⁵<http://dbpedia.org/>

⁶<http://www.geonames.org/>

2.2 Freebase

Freebase is a large collaborative knowledge base consisting of primary data and metadata composed mainly by its community members. It has over 39 million topics about real-world entities like people, places and things. It was developed by the American software company Metaweb and has been running publicly since March 2007. Metaweb was acquired by Google in 2010.

Freebase data is represented as a graph, topics correspond to nodes in the graph. It can be accessed via a user interface and specific web services. A downloadable version of the data is available in the form of data dumps⁷. The content of these data dumps constitutes a snapshot of the data stored in Freebase and the scheme that structures it at a particular time. The content is provided under the CC-BY license.

To enable fast processing of the data, we downloaded a data dump and used the structure of the graph to get information relevant for the domain of interest. Numbers of entities extracted from Freebase are given in Table 1. In addition to visual artists, a list of all persons covered by Freebase was also prepared for disambiguation purposes (see Section 4.1). Each entity is accompanied by a Freebase ID, Freebase URI, title, aliases, description, and Wikipedia URIs (links to Wikipedias in various languages). Links to images were also stored (when available) for visualisation in the Annotation Editor (see Section 8).

type	number of entities
visual artwork	32,000
visual artist	28,724
person	2,618,382
geographic location	971,149
museum	7,858
historical event	88,365
visual art form	33
visual art genre	131
period or movement	225
visual art medium	885

Table 1: Number of entities of specific types extracted from Freebase

Type-specific fields were extracted for each entity type. Fields artist, art subject, art form, art genre, media, support, period or movement, location, owner, date begun, date completed and dimensions were extracted for artworks. Artists are accompanied by: period or movement, influenced, influenced by, place of birth, place of death, date of birth, date of death, profession, art form, places lived, gender and nationality. Location

⁷<http://download.freebaseapps.com/>

records contain: latitude, longitude, location type, country and population. Museums have the following attributes: type, established, director, visitors, city/town, state/province/region, postal code, street address, latitude, longitude. Finally, events are accompanied by start date, end date, location and notable types. No special attributes were available for the rest of types obtained from Freebase.

Although the number of entities extracted from Freebase is quite large, many of them are incomplete. Often, only the name field is filled in. Also, some entities are not correctly categorized into a template they would correspond to (for example, Salvino Salvini⁸ – an Italian sculptor – is not categorized as a visual artist). This is solved in the process of semantic integration through links from Freebase to Wikipedia pages discussed in Section 3.

2.3 Wikipedia/DBpedia

Wikipedia – the free encyclopaedia containing more than 4.3 million articles in English (and hundreds of thousands to over 1 million in other languages) – needs to be considered in any semantic enrichment activity. Although we mainly dealt with English in DECIPHER, data dumps⁹ of German, Spanish, French, Italian, Polish, and Czech Wikipedias were also processed to identify relevant entities appearing only in national versions of the resource. Available images were also downloaded¹⁰ and linked to relevant entities.

Unfortunately, it is not easy to get a list of all entities of a specific type (for example, all visual artists) from Wikipedia. Three approaches were explored for this task:

1. information extraction from infoboxes structuring data relevant for entities described by articles;
2. classification of articles into specified categories defined by Wikipedia such as *Artists by nationality*;
3. text mining of article content based on patterns tailored for particular entity types such as *X was a sculptor*.

Although there is a significant overlap among the lists of entities acquired by the above-mentioned methods, it is crucial to realize that they are complementary – they never lead to the same list due to their inherent limitations. For example, info-boxes are available for less than 50 % of Wikipedia articles and they can be filled only partially. On the other hand,

⁸<http://www.freebase.com/m/0rpjcdn>

⁹http://wikipedia.org/wiki/Wikipedia:Database_download

¹⁰<http://ftpmirror.your.org/pub/wikimedia/imagetdumps/tarballs/fulls/>

not all articles have an association with a relevant category, and extraction patterns do not cover all potential ways to express relevant information.

Let us discuss the *visual artwork* type of entity in detail as an example of the above-mentioned issues and all the steps necessary for obtaining a single list of entities. Of course, various mentions of artworks appear in Wikipedia, especially in articles dealing with artists or art galleries. Here, we focus only on visual artworks discussed in individual Wikipedia articles.

We completely relied on DBpedia for the infobox-based method. DBpedia is a crowd-sourced community effort to extract structured information from Wikipedia infoboxes and make this information available on the Web. DBpedia version 3.8 – the latest dump of the entire database¹¹ available at the time of writing this report – as well as DBpedia-Live¹² – a version of DBpedia continuously synchronized with Wikipedia – were used in the reported work.

In principle, it is possible to query DBpedia for article categories or to process values of properties such as *shortDescription* using text mining patterns. However, we focus on the use of infobox templates only; the rest of processing is performed on the actual Wikipedia pages. Even with this specific focus, DBpedia’s structuring of information does not guarantee an easy way to find all entities of a given type. Indeed, infoboxes are often applied inconsistently and there is only a limited effort to unify DBpedia data in such cases. For example, to find articles that instantiate an artwork infobox template, one has to search also all redirections to that template. It corresponds to the following query expressed in SPARQL¹³:

```
PREFIX prop: <http://dbpedia.org/property/>
PREFIX ont: <http://dbpedia.org/ontology/>
PREFIX templ: <http://dbpedia.org/resource/Template:>

SELECT DISTINCT(?a) WHERE {
  {?a prop:wikiPageUsesTemplate templ:Infobox_artwork}
  UNION {
    ?a prop:wikiPageUsesTemplate ?i .
    ?i ont:wikiPageRedirects templ:Infobox_artwork}
}
```

This led to a list of 4269 unique artworks. In addition to title and DBpedia URI, we extracted other fields (if they were filled in the template), including: creator, type (for example, *oil painting*), year of creation, owner,

¹¹<http://dbpedia.org/Downloads38>

¹²<http://live.dbpedia.org/>

¹³<http://www.w3.org/TR/rdf-sparql-query/>

museum, abstract, dimensions, and Freebase URI. Links to images were also stored. Further extensions of the list of artworks were performed by the other two extraction approaches – the one based on Wikipedia categories and another one based on text mining of full articles. This separation should show how much data can be reached by a simple use of infoboxes available in DBpedia and what is an additional value of the thorough processing of the Wikipedia content.

Due to the fact that Wikipedia does not group all artworks into a set of easily identifiable categories, we examined the data and found common categories for artworks (focusing on paintings and sculptures such as the category *Paintings by year*). We identified 89 appropriate high-level categories and their 481 leaf subcategories. There were 7 660 unique member articles. Although names of the categories suggest that articles belonging to them refer exclusively to visual artworks, various inconsistencies appear. For example, we filtered out 429 articles referring to artists (with a higher score for type *artist*). If available, the information about author, start date, end date, movement period, type, location and URI was extracted for identified artworks. In total, 7 231 artworks were obtained.

As summarized in Table 2, the additional Wikipedia processing exploring all members of identified categories and applying text mining patterns on the article content brought significantly better results – many artworks could not be identified by a simple use of DBpedia.

how obtained	number of entities
infoboxes from DBpedia	2,873
additional Wikipedia processing	8,201
total artworks	11,074

Table 2: Number of artworks extracted from Wikipedia

Similarly to artworks, lists of other entities were also initially populated from DBpedia and then extended by further Wikipedia processing. DBpedia ontology does not identify visual artists and there are many inconsistencies in property values so that complex SPARQL queries had to be formulated again:

```
SELECT ?artist ?field WHERE {
  {?artist a <http://dbpedia.org/ontology/Artist>;
   <http://dbpedia.org/property/field> ?field .
  FILTER(regex(str(?field), "paint", "i") ||
           regex(str(?field), "sculpt", "i") ||
           regex(str(?field), "photograph", "i") ||
           regex(str(?field), "visual_art", "i") ||
           regex(str(?field), "illustrat", "i") ||
```

```

    ...
  ) }
UNION
{?artist a <http://dbpedia.org/ontology/Architect>}}

```

The resulting list of visual artists containing 6 975 records was further extended by artists belonging to specific categories or matching a text-mining pattern. The data about persons covered by Wikipedia articles were obtained in a joint process. Categories such as *Births by year* or *Living people* were employed together with person-specific text-mining patterns such as *born in PLACE*. Information about name, alternative names, date and place of birth and death, short description and URI were stored. Table 3 shows results for all types of entities.

type	number of entities
visual artwork	11,074
visual artist	52,475
person	1,122,561
geographical location	515,035
museum	18,298

Table 3: Number of entities of specific types extracted from Wikipedia

2.4 Geonames

To be able to recognize and correctly classify mentions of even small villages and other place names with no coverage in Freebase and Wikipedia, we decided to take advantage of the Geonames.org. The worldwide geographical database provides a freely available data covering millions of locations. Over 10.3 million unique geographical names corresponding to 8.5 million unique entities (referred to as *features*) are listed. The features are divided into categories such as populated places, political entities, lakes, or parks. The primary category – populated places – contains 3.2 million records of locations that can be referred to by 4.4 million unique names.

The data is accessible via a number of web services as well as via a daily export¹⁴. The database stores names of places in various languages and integrates additional properties such as elevation, population and latitude and longitude coordinates. Properties relevant for both – prioritization and disambiguation – population and hierarchical classification to regions, states and continents – and for visualisation purposes – coordinates – were stored. The resulting tool recognizes any of the names but it can also distinguish their importance for a particular annotation task.

¹⁴<http://www.geonames.org/export/>

type	number of entities
city, village...	3,189,235
stream, lake...	1,721,911
mountain, hill, rock...	1,066,573
parks, area...	314,187
country, state, region...	288,692

Table 4: Frequent types of geographical locations contained in the Geonames.org database

3 Semantic integration of datasets

The process of taking two or more collections of data (dealing with the same domain) and producing a consolidated base is generally referred to as *semantic integration* [11]. It includes finding equivalences of data instances referring to same entities (realized through sameAs links in OWL resources [7]), merging attributes, or identifying and solving conflicts in values.

Motivated by the current efforts in the Linked Open Data initiative [4], we focused on anchoring all entity mentions in existing resources whenever possible and providing links (URIs) to all the sources containing information about a particular annotated entity. As a by-product, the DECIPHER semantic integration process produced interlinks among the mentioned resources that are freely available for other researchers¹⁵. We believe they can also play a significant role in open linking some currently closed resources, such as in the current effort to provide Getty resources as linked open data¹⁶.

3.1 Data consistency issues and overlaps

The first step of the semantic integration lies in identifying attributes of individual collections to be merged. It is easy if there is a direct link from a record in one dataset to another record in another one. This is the case of many Freebase entries that contain links to Wikipedia and, vice versa, DBpedia records often contain the reverse – a Freebase URI. However, it is tricky to interlink datasets such as ULAN that do not try to link other datasets and whose attribute values are often inconsistent.

Sophisticated techniques inspired by the ontology alignment¹⁷ needed to be used for the semantic integration task in DECIPHER. Not only names but also additional attributes available from the sources were considered

¹⁵<http://knot.fit.vutbr.cz/KB.all>

¹⁶<http://www.getty.edu/about/opencontent.html>

¹⁷<http://oaei.ontologymatching.org/>

	ULAN	Freebase	Wikipedia
ULAN	181,718	13,348	16,589
Freebase	-	28,724	21,435
Wikipedia	-	-	52,475

Table 5: Number of unified entities across data sources

and compared across the datasets. Manual examination of intermediate results then led to identification of attributes that are frequently inconsistent and thus need a special treatment.

For example, while processing data from ULAN, we observed many inconsistencies in the birth and death dates attributes. The dates often indicate a decade only; many people have set the death date to a particular year in the future. When communicating these issues to Getty, it turned out that the data in these attributes is not meant to be displayed directly to the users, rather, it is used internally for retrieval purposes. Special procedures were therefore necessary to facilitate further integration of the ULAN data.

Fortunately, ULAN records link one or more biographies of referred artists. Correct dates of the birth and the death could be extracted from the source text. Extraction patterns for the dates were learned from Wikipedia pages corresponding to biographies and used on the texts of biographies linked from ULAN. A certainty score was also computed that characterizes consistency of the sources (in the case of conflicting dates across the processed biographies). In total, 13,195 date values were corrected and added to the ULAN dataset.

Finally, we explored overlaps among the processed datasets. Results for artist names are summarized in Table 5 showing numbers of unified entities of the type `visual_artist` interlinked between all pairs of relevant sources used in the Semantic Annotator.

The integration process allows not only fully interlinking the resources but it also identifies further inconsistencies in the source data and enables their potential corrections. For example, we found out that 1,218 artists in Freebase do not use correct person templates and initiated steps to correct the errors in the source.

3.2 Integrated knowledge base

After merging individual datasets, an integrated knowledge base of the Semantic Annotator could be created. In addition to key entity types – visual artists, visual artworks, cultural institutions, historical events, and geo-

graphical locations – less numerous domain-specific categories were added – artistic movements or periods, art forms, art genres, and media. Their main role is to provide domain ranges for attribute values of the primary entities but they can be also employed for other tasks – autocompletion in particular entry fields in a user interface, facet definitions in an advanced search, controlled vocabularies or lists of preferred values.

A normalized form of a name, all alternative names, the semantic type and URIs (especially, Freebase and DBpedia) are stored for each entity. Additional information attached to entities is stored specifically for each entity type. This includes roles such as architect or sculptor, nationality, short description, relevant periods or movements, dates and places of birth and death and gender for artists. For artworks, the knowledge base includes creator, dates of creation, art subject, art form, art genre, media, artistic period or movement, owner and dimensions. Geographic entities identify subtypes such as city or state, coordinates, population, and the Geonames database identifier. For cultural institutions, the knowledge base contains the same information as for artists. The primary name, aliases, description, image, start date, end date, locations, notable types, Freebase URL and Wikipedia URL are stored for historical events.

The knowledge base contains 657,862 entities with 3,240,963 distinct attributes. Table 6 shows the numbers for each entity type. Note that numbers of individual attribute values are reported if there are multiple values.

entity type	entities	number of attributes	names	ambiguous
visual artist	227,919	2,498,795	700,768	7,229
visual artwork	32,058	237,697	36,228	1,329
person	2,728,008	39,131,657	3,324,866	94,403
geographical location	6,580,598	48,743,018	21,381,856	1,303,674
cultural institution	44,041	63,303	93,906	522
historical event	88,365	668,581	95,828	877
visual art form	33	223	45	0
visual art genre	131	854	144	0
period of movement	225	1422	274	0
visual art medium	885	4230	1,080	0
nationality	593	4171	725	0
total	9,702,856	91,353,951	25,635,720	1,853,792

Table 6: Numbers of entities, attributes, and alternative names in the knowledge base

Normalized forms of entity mentions together with all known alternate names stored in the knowledge base enable identifying equivalent entities expressed differently in texts. For example, Gabriël Metsu refers to the same person as Gabriel Metsu, Pablo Ruiz y Picasso is known as Pablo

Picasso, or US can stand for the United States (of America). Such information comes either directly from the data records provided by particular sources (for example, alternative names in ULAN) or it was obtained from additional information linked to the records (for example, Wikipedia redirections). The last but one column of Table 6 characterizes quantities of alternate names for each entity type.

As discussed in Section 5.1, names of all entities form a large gazetteer resource that is used to build a finite-state automaton employed by the Semantic Annotator. As the same name can refer to more than one entity, it is also important to characterize numbers of ambiguous names in the knowledge base. The ambiguity can be manifested on two levels – only within a particular entity type (for example, St. Petersburg can refer to a place in Russia, Florida, or Pennsylvania, not speaking about St. Petersburg, Missouri, fictional hometown of Mark Twain’s characters Tom Sawyer and Huckleberry Finn) or across semantic categories (Washington as the name of the state v. a person).

In general, ambiguity increases with the growing number of entities of various semantic types. Places are often named after famous people and artworks refer to places. The last column of Table 6 demonstrate ambiguity of names within specific single categories as well as the total number of ambiguous names.

As mentioned above, the data from the knowledge base are used not only for the primary purpose – the semantic annotation, but also in additional functions such as various autocomplete widgets providing suggestions while a user types into a field. Corresponding web services – `annotate` and `get_entities` (see Section 6) – take advantage of the integrated data and employ the content for their specific needs. To enable an efficient processing, the knowledge base is stored in a single file that takes 143 MB. It is intended to be loaded into the main computer memory only once and stay there during the whole life of an SEC instance.

4 Disambiguation and co-reference resolution

The data integrated into the annotation knowledge base, as discussed in previous sections, would already allow annotating explicit mentions of covered names by links to all entities they can represent. This could be useful for manual processing when a user chooses the correct link to a particular entity in the case of an ambiguous name. However, automatic procedures built on top of the Semantic Annotator ask for a disambiguated input or, at least, a list of potential ambiguous entities sorted by their estimated relevance in a particular context.

There are also words that refer to entities depending on other mentions in a text. Pronouns are typical representatives of these co-reference expressions. However, there are many other types – a part of a name used instead of the full name or entities referred to by their types. The task of co-reference resolution lies in grouping expressions referring to same entities. These areas form topics of this section.

4.1 Disambiguation data and processes

Considering entities of all categories contained in the annotation knowledge base, it is obvious that advanced disambiguation mechanisms need to be employed to correctly annotate data in DECIPHER. When dealing with a specific domain, 80–90 % of the ambiguity can be solved by a simple algorithm preferring always the most frequently mentioned entity that can be referred to by a given ambiguous name. However, it is not easy to get relevant data to compute the frequency. Advanced disambiguation approaches built on top of the simple algorithm consider then also contextual information from a particular occurrence (or even all occurrences) in a text, general co-occurrence statistics from a whole collection and so on.

Entity attributes such as population of places, manually annotated preferences and metadata about the use of on-line resources were used as primary sources for the frequency-based disambiguation. Unfortunately, usage statistics are not freely available for all the sources. That is why we focused on Wikipedia data that contain relevant information.

The Wikimedia Foundation provides page view statistics for all Wikimedia projects (including Wikipedia, Wikibooks and Wiktionary) in all languages. The statistics are available online¹⁸ and contain numbers of page-loads for each URI. The data is available for each hour since January 2007. In order to limit the amount of data and to focus on recent trends, we decided to use only a portion of the data from the first half of 2013. Page-loads for each URI were aggregated into a single value. The resulting data

¹⁸<http://dumps.wikimedia.org/other/pagecounts-raw/>

file took 1.8 GB and contained 57,166,659 records. The analysis of the data showed that 34,998,453 of records had value larger than 20.

To get another measure of relative importance of entities referred to by Wikipedia pages, we also collected numbers of links to a particular pages from all other Wikipedia pages (backlinks). The more backlinks a page has, the more important it is considered. A freely available script¹⁹ was employed and the data was stored as a specific attribute for the particular entities.

Wikipedia also identifies primary topics for some ambiguous names or terms that can refer to multiple things. The decision on what page will be taken as primary for an ambiguous term needs to be done by a Wikipedia editor who considers primarily two aspects – term usage frequency and long-term significance. According to Wikipedia documentation²⁰, a topic is primary for a term in Wikipedia with respect to usage; it should be returned when a reader searches for the term, if it is highly likely – much more likely than any other topic – and more likely than all other topics with a shared name combined. The data on primary topics was also extracted from Wikipedia dumps and added to records corresponding to primary entities.

The relative importance of entities that can be referred by the same name form a basis of the disambiguation process. The textual context in which the name appears is considered next. Rules preferring longer matches provide a primary mechanism for the dynamic disambiguation. For example, a text containing *Bobigny – Pablo Picasso*, refers probably to a station of the Paris Metro and does not primarily deal with the famous Spanish artist.

A simple naive Bayes model is trained on all data available – Freebase and Wikipedia pages, biographies linked from ULAN records and other collected resources. We employ the same data structure as described in [1] to efficiently store disambiguation features. The naive Bayes model also provides a confidence score which is returned together with the most probable annotation.

4.2 Co-reference resolution

Co-reference resolution is the problem of identifying expressions in a text that refer to the same entity [10]. For example, in the sentence: *Vincent van Gogh worked as a missionary in a mining region in Belgium where he began to sketch people from the local community*, the name *Vincent van Gogh* and *he* refer to the same person (the famous Dutch painter). Co-reference resolution provides an enabling step towards advanced semantic enrichment of text in DECIPHER.

¹⁹<https://toolserver.org/~dispenser/cgi-bin/backlinkscount.py>

²⁰<http://en.wikipedia.org/wiki/Wikipedia:Disambiguation>

Corresponding to their frequency and importance for the texts dealt with in DECIPHER, we primarily focus on *anaphoric references* where the algorithm needs to look only to the previous context in a text. Simple *cataphora* which co-refer later expressions in a discourse are addressed too, for example, *After his mother died, Edvard Munch was raised by his father and aunt Karen* or appositions such as *Howe met his future wife, Heidi Hampel, while serving overseas*.

The co-reference resolution in DECIPHER exclusively deals with references to explicitly identified entities *within the same text*, a.k.a. *endophora*. Referring to extra-linguistic knowledge, for example, *the Queen* (the meaning of which may need to be determined by the country and the time in which it is spoken) is out of scope of the module.

The task of co-reference resolution is usually divided according to the linguistic form of expressions linked by co-reference relations. In addition to typical nominal forms (personal and possessive pronouns, proper names and lexical noun phrases), we also distinguish a specific type of non-nominal references to events. Table 7 demonstrates the most frequent types of co-reference relations addressed by the tool.

type	example
pronoun	Salvador Dalí was a skilled draftsman, best known for the striking and bizarre images in his surrealist work.
proper name variant	Yeats's paintings usually bear poetic and evocative titles. Indeed, his father recognized that Jack was a far better painter than he.
entity type	Swift's travels led him to the small fishing village of Carvoeiro in the Algarve . He was so enchanted with the place that he remained.
hypernym	Rachel Lietch began her journey in 2007 when she met a painter in Colorado who lived and worked on a Ranch. Rachel returned from her trip so inspired by this artist , she also wanted to paint from her vacation pictures in Colorado.
synonym	Van Dyck worked as Rubens' student in his vast studio at the age of 15. Rubens noted the young artist as his best pupil
event reference	Casagemas shot himself because of an unrequited love for Germaine Pichot. Picasso's painting <i>La mort de Casagemas</i> , completed early in the year following his friend's suicide , was done in hot, bright hues.

Table 7: Examples of co-reference from data relevant for the project domain

As discussed in previous WP4 deliverables D4.1.1 and D4.2.1, we compared several existing tools that deal with co-reference resolution and evaluated their performance on a subset of the DECIPHER data. The tools

included ARKref²¹, CherryPicker²², Stanford Co-referencer²³, JavaRAP²⁴, BART²⁵, RelaxCor²⁶, Illinois Co-reference Package²⁷, Reconcile²⁸, MARS²⁹, GUITAR³⁰, and A*Star Co-referencer³¹. The first two tools delivered the best accuracy results in our test. However, none of the tools was fast enough to be applicable in our setting. Indeed, the context in which the SEC works (for example, generating immediate suggestions for texts specified by a user – see Section 8) asks for running times corresponding to processing thousands of words per second.

This finding led us to a decision to implement a simple tool that could perform slightly worse than the state-of-the-art systems in terms of the F-measure but that would be able to run in the limited time. We took advantage of the machine-learning models trained on standard datasets (MUC [2, 6], ACE [5], OntoNotes [8, 12]), extracted most significant features that contributed to correct co-reference annotations, and transformed the data into a fast module based on the finite-state technology (a finite-state transducer). The resulting tool makes also use of the knowledge base consisting of all entity types discussed in previous sections. It is therefore able to correctly identify co-reference relations that are contained in the data such as *the sculptor* referring to (*Auguste*) *Rodin* in the sentence *Rilke stayed with Rodin in 1905 and 1906, and did administrative work for him; he would later write a laudatory monograph on the sculptor*. The F-measure of the implemented tool reached the value of 71.6, while the best performing tools had 73.2 on the testing data. On the other hand, the processing was much faster – see the next section.

A detailed analysis of results, especially misses of the co-reference resolution module, showed that there is a room for further improvements of the domain-specific knowledge-based approach. A future development could take into account cross-type entity information available. For example, the information that *Pablo Picasso* was a *Spanish* painter is already stored in the knowledge base. It could be therefore used to identify that *the country* refers to *Spain* in the sentence *Picasso's father and uncle decided to send the young artist to Madrid's Royal Academy of San Fernando, the country's foremost art school*.

²¹<http://www.ark.cs.cmu.edu/ARKref/>

²²<http://www.hlt.utdallas.edu/~altaf/cherrypicker.html>

²³<http://nlp.stanford.edu/software/dcoref.shtml>

²⁴<http://wing.comp.nus.edu.sg/~qiu/NLPTools/JavaRAP.html>

²⁵<http://www.bart-coref.org/>

²⁶<http://nlp.lsi.upc.edu/relaxcor/>

²⁷http://cogcomp.cs.illinois.edu/page/software_view/18

²⁸<http://www.cs.utah.edu/nlp/reconcile/>

²⁹<http://clg.wlv.ac.uk/demos/MARS/index.php>

³⁰<http://cswww.essex.ac.uk/Research/nle/GuiTAR/gtarNew.html>

³¹http://nlp.i2r.a-star.edu.sg/demo_coref.html

5 Resulting component and its evaluation

This section discusses the design and implementation of the final version of the Semantic Annotator. It details its internal structure consisting of an efficient entity-mention spotter, a knowledge-base module, and processing components for disambiguation, co-reference resolution and auxiliary annotations. The resulting component is evaluated in terms of its performance and coverage of entities.

5.1 Semantic annotation module

As discussed in Section 3.2, there are hundreds of thousands of entities in the integrated knowledge base. Advanced mechanisms are necessary to efficiently identify potential mentions of any of the entities in a text. The Semantic Annotator employs a set of finite state automata (FSA) realized by an optimized code in C for this purpose.

The biggest automaton stores a complete list of primary and alternate names that are known to refer to the entities of all types covered by the knowledge base. An index to the knowledge base corresponding to an entity that can be referred to by a given name (or a list of such indices in the case of ambiguous names) is added to each of the names. Special automata are then constructed for entities of each individual type to be supported by the autocompletion or any other function based on the `get_entities` function specifying the type.

The automata are minimized by means of an incremental algorithm introduced in [3]. We take advantage of a freely available implementation of the algorithm³² as well as tools allowing searching in resulting structures. The spotter of entity names builds on this basis. It takes an input text, tokenizes it and applies the search in the automaton. If a potential name is identified in the input, its occurrence is tagged together with entity IDs linking the name to the knowledge base.

The data structure used to store the automata provides also an easy way to realize a prefix search in the data. It is beneficial for the autocompletion function as all names beginning with a given string can be easily found. That is why the `get_entities` function is able to serve both – the search for entities corresponding to a full name and the generation of autocompletion suggestions.

In addition to the FSA-based component, the entity spotter implements an interface to the knowledge base (KB). The service is responsible for loading the KB into the main memory and keeping it there during the whole processing. Having an entity index, the interface simply returns a struc-

³²<http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/fsa.html>

ture corresponding to a full entity record of a given type. A caller can then access this information and use it in further processing.

The actual output of the spotter depends on parameters specified by a calling module. It can be as simple as an identification of the start and the end of an entity occurrence in an input text and a URL linking the entity to Freebase, DBpedia, or another resource. On the other hand, detailed annotations can be produced that explicitly state an entity type, and other properties such as the date of birth and the death of a person or coordinates of a place (see Section 3.2 for a detailed list of attributes for specific types of entities).

In addition to a plain text output, the spotter can generate a HTML visualising semantic types of entities in a special way (for example, in different colours). The meaning of particular types is separated from an actual visual form so that a caller can customize the output. An RDFa representation that corresponds to schemata used by Storyscope can be also produced.

As mentioned above, contextual information in the form of disambiguating words and multi-word expressions is also linked to KB records corresponding to ambiguous entities. The disambiguation, co-reference resolution and relation extraction are performed in a single pass through the text to keep the processing as fast as possible. Additional processing extracting temporal expressions, monetary values, and proper names that are not included in available gazetteers is also performed in this stage.

Although there are tools which could be used for identifying temporal expression in texts, for example, TARSQI³³, Chronic Ruby library³⁴, Heidelberg-Time³⁵, we had to implement own tool to meet requirements on the high speed of processing. The resulting component identifies temporal expressions by means of optimised regular expressions. It deals with various formats and ways to express uncertainty of dates and transforms data into a unified format compatible with the ISO 8601 standard³⁶.

Despite large coverage of the processed lists of entities, there will always be entities whose names are missing from the lists (especially artists and artworks). A set of heuristics to recognize unknown, potentially relevant entities is thus employed. Various features are considered – word capitalization, part-of-speech tags, or co-occurrence with entities of a known type – to identify candidate names first. Pre-defined patterns are then matched against a text containing a candidate. If a pattern matches a text fragment, the candidate is marked as a new entity of a given type.

A specific set of patterns is used for each individual entity type. Three examples of such patterns for unknown artworks are given below:

³³<http://www.timeml.org/site/tarsqi/index.html>

³⁴<http://chronic.rubyforge.org/>

³⁵<http://dbs.ifi.uni-heidelberg.de/index.php?id=129>

³⁶<http://www.w3.org/TR/NOTE-datetime>

- `/?artwork?/ (creation_verb_in_passive){0,1} by /artist/`
- `/artist/ (active_creation_verb) /?artwork?/`
- `/?artwork?/, /?artwork?/, ... and other (artwork_type)`

Strings with question marks and slashes (`/?artwork?/`) correspond to candidates for new entities, whereas those in slashes only (`/artist/`) correspond to currently known entities. Lists of verbs expressing the act of creation (*painted, made...*), artwork types (*sculptures, tapestries...*) and so on are learnt from data by bootstrapping.

5.2 Performance and coverage evaluation

The created component has been evaluated as an integral part of the DECIPHER system during user trials. Results of the final validation and field trials will be available in deliverable D7.3.1 – Final Field Trial and User Evaluation Report. However, even past trial results brought interesting research outcomes. For example, museum professionals rejected the European dataset in evaluations (Phase 2 trials) due to data consistency issues. This adds up with findings discussed in this section.

In addition to its primary use as a background process called by the Content Aggregator, two user components described in Sections 7 and 8 directly employ the semantic annotation. The peer evaluation regarded their use as “power user” tools that should not be evaluated by the visitor cohort taken as “naive users”.

The rest of this section focuses on general characteristics the Semantic Annotator in terms of its accuracy and time and memory requirements. It also shows what coverage can be expected on an example of the list of creators in various art collections.

All entities and relations of relevant types appearing in a subset of the DECIPHER textual content collected within the project were manually identified and linked to corresponding data sources. In total, 986 words or multi-word expressions were annotated. The Semantic Annotator produced 923 annotations out of which 865 matched the manually created ones. This corresponds to an 87.7% accuracy of the tool.

All automata used by the spotting tool recognizing more than 8.5 million distinct names take 140 MB of memory. This means a significant reduction compared to the size of source files – 1.2 GB. The knowledge base together with its interface and all necessary indices take together 976 MB of the memory.

The processing is extremely fast. The data and the processing code are loaded into the main memory when semantically enrichment services start. The initialisation phase takes several seconds. It is performed only once. The actual name spotting then accesses the structures and scans the text

in the speed of about 50,000 words per second. Table 8 demonstrates that the speed is independent of the size of input. All additional processing including disambiguation, co-reference resolution, and relation extraction increase the time but the speed still corresponds to several thousand words per second (the last column of Table 8).

length of input in words	size of input	spotting time	total time
10,000	146 kB	0.4 s	5.8 s
100,000	1.5 MB	2.2 s	18.2 s
1,000,000	15.8 MB	20.1 s	240.9 s
10,000,000	158.2 MB	199.8 s	2533.7 s

Table 8: Performance characteristics of the Semantic Annotator

To evaluate coverage of the Semantic Annotator on a large dataset of artist names, we considered collections explored in the DECIPHER project, processed the list of creators by means of the `get_entities` function and measured a percentage of hits or misses. Table 9 shows results for some of the collections.

collection	distinct creators	found in knowledge base
IMMA	695	424
NGI	1,171	976
Rijksmuseum	5,315	2,412
Christie's	16,628	10,187
Europeana	773,127	31,274

Table 9: Coverage of the entity knowledge base on artists from various collections

Special attention was also paid to the poor result on Europeana. A detailed analysis of misses showed that the coverage relates to the nature of the dataset. The Europeana digital library³⁷ collects millions of records on paintings, books, films, and other museum and archival objects that have been digitized throughout Europe. More than 2,000 cultural and scientific institutions across Europe have contributed to Europeana.

Since Europeana does not precisely define meaning and purpose of each particular field in the database, many mistakes come directly from the unmanaged importing process realized by participating institutions. Fields often mix content of various semantic nature and, occasionally, they are completely misinterpreted (for example, field *creator* stands for the author, but, in many cases, it contains only the institution the data comes from).

³⁷<http://www.europeana.eu/>

Moreover, the data in records is rather sparse – many fields are left empty even though the information to be filled in is included in original museum records (for example, the author of an artwork is known but not entered).

6 SEC API extensions

This section describes extensions to the SEC API (Application Programming Interface) that make available two key functions of the Semantic Annotator – `services annotate` and `get_entities`. Two other meta-services providing a list of semantic types included in the knowledge base and a definition of corresponding attributes stored – `get_entity_types` and `get_entity_type_attributes` are also described. The services can be accessed by clients with varying needs, including the other two key components of the DECIPHER system – the Content Aggregator and Storyscope.

The interface defines an HTTP-based protocol with data in JSON. Clients send data through the standard HTTP POST method. Field `API` in the HTTP header shall have value `SEC_API`, field `version` shall have value `1.2`.

6.1 Service `annotate`

The service returns annotations for a given document. Annotations are primarily represented by their begin offsets, end offset, and internal IDs linking the mentions to the knowledge base. If the parameter `disambiguate` is set to `0`, the Semantic Annotator does not return disambiguated entities but all possible entities corresponding to each particular name. By default, the parameter is set to `1`.

The annotation output can be also produced in one or more formats specified by the parameter `annotation_format`. The following values of the parameter are supported:

- `text` – annotations will be provided in a plain text form intended for human reading, they will be directly included the textual output
- `index` – annotations will be provided in a plain text form intended for fulltext indexing
- `xml` – an XML version of the annotation will be produced
- `html` – an HTML version of the annotation will be produced
- `rdf` – an RDF version of the annotation will be produced

For each format included in the parameter `annotation_format`, an actual content of annotations expected in the output needs to be specified by a relevant parameter `types_and_attributes_*`.

The request has to provide:

- `input_text` – the text to be annotated.
- `annotation_format` – a list of formats in which results should be formatted

Optional fields of the request are:

- `disambiguate` – 1 (default) - produce an disambiguated output, 0 - do not disambiguate
- `types_and_attributes` – a JSON structure specifying what types of entities and relations should be annotated and what respective attributes should be included in the output.

The response contains one field corresponding to format specified in the request.

Syntax

Request:

```
{
  "annotate": {
    "input_text": "",
    "annotation_format": ["text"|"index"
                          |"xml"|"html"|"rdf"],
    "disambiguate": 1,
    "types_and_attributes": { ... },
  }
}
```

Response:

```
{
  "annotation":""
}
```

Example

Request:

```
{
  "annotate": {
    "input_text": "William Orpen was an Irish
                  portrait painter, who worked
                  mainly in London.",
    "annotation_format": ["xml"],
    "disambiguate": 1,
    "types_and_attributes": {
      "person":["full_name", "freebase_url"],
      "location":["name"]
    }
  }
}
```

Response:

```
{
  "annotation": "<person full_name=\"Major Sir William
    Newenham Montague Orpen\"
    freebase_url=\"http://www.freebase.com/m/04rb6k\">
    William Orpen</person>",
}
```

6.2 Service `get_entities`

The service returns data on entities stored in the knowledge base. Two primary domains of its use can be distinguished:

1. semantic enrichment of a textual content found in a particular attribute of a collection record (for example, `creator`, `location`, `cultural institution`, `location`);
2. autocompletion suggestions in an input field if a user types first characters of a name.

The request has to specify:

- `input_string` – the full content of a record field to be semantically enriched or an initial part of a name entered by a user ended by the asterisk sign (*).
- `types_and_attributes` – a JSON structure specifying attributes of given type(s) that should be included in the output.

Optionally, it can also provide:

- `type` – searches only entities of a specified type (the default is "all")
- `max_results` – a maximal number of entities returned (the default is 10)

The response contains a list of entities (values of attributes specified in the input parameter `types_and_attributes`). If the parameter `type` was not provided or it was set to "all", the attribute "type" precedes every record in the output. Otherwise, only attributes specified in the input `types_and_attributes` are returned.

Syntax

Request:

```
{
  "get_entities":{
    "input_string": "",
    "types_and_attributes": [],
    "type": "all|creator|location|...",
    "max_results": 10
  }
}
```

Response:

```
{
  "data": [
    {
      "type": "",
      "name": { },
      ...
    }
  ]
}
```

Example

Request:

```
{
  "get_entities":{
    "input_string": "Met*",
    "types_and_attributes": {"person":
      ["full_name", "freebase_uri"]},
    "type": "creator",
    "max_results": 3
  }
}
```

Response:

```
{
  "data": [
    {
      "full_name": "Gabriel Metsu",
      "freebase_uri": "http://www.freebase.com/m/071k7_"
    },
    {
      "full_name": "Quentin Metsys",
      "freebase_uri": "http://www.freebase.com/m/03h2z4v"
    }
  ],
}
```

```
{
  "full_name": "Willard Metcalf",
  "freebase_uri": "http://www.freebase.com/m/03p484"
}
]
```

6.3 Service `get_entity_types_and_attributes`

The service returns types of entities and their respective attributes stored in the knowledge base.

Depending on the actual content of the knowledge base, the response can look like the following:

```
{
  "data": [
    {
      "type": "artist",
      "attributes": {"full_name",
                    "alternative_names",
                    "freebase_url",
                    "wikipedia_url",
                    ...}
    }
  ]
}
```

7 Semantically enriched fulltext search

To demonstrate advantages of the semantic annotation for advanced fulltext search, we implemented a part of the DECIPHER system – the SEC Store – that takes semi-structured documents (such as records from Europeana, Wikipedia pages. . .), applies the Semantic Annotator, and stores results in an indexed form. It then enables searching in the semantically enriched content by means of an advanced query language, computes facets for easy navigation in collections, and supports similarity search on the semantically enriched content. The functionality is made available to Storyscope users as a form of the external search for objects and stories (see a screenshot in Figure 1. The following subsections detail specific components of the SEC Search.

STORYSCOPE

The screenshot shows the Storyscope search interface. At the top, there are navigation links for 'Home' and 'Dossiers', and a 'Page Help' button. Below this is a breadcrumb trail: 'HIGHLIGHTS OF THE NATIONAL GALLERY OF IRELAND > OBJECTS > ADD OBJECTS /'. A search bar is present with a search button. Below the search bar, it indicates '3613 results'. The main content area displays four search results, each with a thumbnail image, title, creator, data source, and a brief description. Each result has an 'add +' button. To the right of the results is a 'Facets' sidebar with the following sections:

- Where found:**
 - External (3613)
- Has image:**
 - True (3126)
 - False (487)
- Source dataset:**
 - NGI (3613)
 - IMMA (0)
 - RIA (0)
 - Freebase (0)
- Creator:**
 - Nathaniel Hone II (232)
 - Harry Aaron Kernoff (194)
 - Jack Butler Yeats (105)
 - Mary Granville Delany (62)

Figure 1: Integration of the advanced SEC search into Storyscope

7.1 Metadata indexing

ElasticSearch³⁸ – an indexing engine based on the Apache Lucene³⁹ – provides a core of the SEC Store. The tool communicates through a JSON-based API⁴⁰. It is implemented in Java but various bindings for other pro-

³⁸<http://www.elasticsearch.org/>

³⁹<http://lucene.apache.org/>

⁴⁰<http://jsonapi.org/>

gramming languages such as Python (PyEs⁴¹) or PHP (Elastica⁴²) can be used too.

Semi-structured documents corresponding to visual artwork records, artist biographies from Wikipedia and other resources are semantically enriched, transformed into the JSON format and stored into the main SEC Store index. Type-specific metadata from the input (such as creator(s) of an artwork, corresponding URIs...) is stored in separate fields that can be easily queried. The metadata added by the SEC corresponding to entities and relations identified in the text are also stored.

A special form of a positional index was constructed to enable querying the metadata. Elasticsearch does not directly support adding metadata to individual tokens. However, the concept of synonyms, placing multiple tokens to one position, can be utilised for this purpose. A special plugin – a custom token analyser – was implemented that transforms a text with annotations to a form interpreted as a sequence of words with identified synonyms. Metadata is then placed “within” a given token, which ensures correctness of proximity querying.

The following lines give an example of the input to the custom token analyser:

```
Vouet [person;painter_role;craftsman_role;
      French_nationality;European_nationality]
visited[activity;travel]
Rome [location;Italy_country;Europe_continent]
```

The metadata appears right behind recognized entities, within brackets, and the custom token analyser recognizes them as synonyms indexed at the same offset.

7.2 Querying

Users can search relevant indexed fields by simple fulltext queries – entering individual words, phrases (in quotes), wildcard characters (* at the end of words). Multiple words are interpreted as a query for documents where all the words appear (AND queries). The keyword OR can be used to express alternatives, – (the minus sign) expresses negation.

Advanced queries are formed as a combination of keyword queries and a set of specific field queries. Semicolons are used as separators. The syntax of the query on a field corresponds to a simple format:

```
field:value;
```

for example:

```
creator:William Orpen
```

⁴¹<http://pyes.readthedocs.org>

⁴²<http://elastica.io/>

The list of all field names that can be queried is available as an autocomplete when an empty query is entered in the search field (when a space is pressed in beginning of the search field). Other autocomplete suggestions are generated when users enter a value into the search field (see Figure 2). A result of an advanced query is demonstrated in Figure 3.

A dropdown menu showing autocomplete suggestions for the search field 'creator: O'C'. The suggestions are: creator: O'C, James Arthur O'Connor, Rodenc. O'Connor (highlighted), Ellis O'Connell, Anthony O'Carroll, Seamus O'Colmain, Liam O'Callaghan, Joseph O'Connor, and John O'Connor.

Figure 2: Autocomplete suggestions generated by the SEC Store API

The screenshot shows the SEC Store search interface. The search bar contains the query 'dataset: freebase; creator: Rembrandt; genre: portrait'. The results are displayed in a grid format, showing four paintings by Rembrandt: 'Portrait of Titus', 'Girl at a Window', 'Flora', and 'Portrait of Gerard de Lairese'. Each result includes a thumbnail image, the title, creator, data source, and a brief description. There are 'add +' buttons next to each result. On the right side, there are facets for 'Where found', 'Has image', 'Source dataset', 'Style or period', 'Genre', and 'Creator', each with a list of values and counts. An 'Add to query and search' button is at the bottom right.

Figure 3: Results of an advanced query

`keyword mentions` is used to query entities and their attributes identified by the SEC. For example, the query:

```
mentions.person: Jack Butler Yeats
```

returns all documents mentioning the artist, regardless of how the name is expressed in the text – John Butler Yeats, Jack B. Yeats, Jack Yeats. In the case of hierarchically organized concepts (such as cities belonging to countries that further belong to continents), any mention of an entity on a lower level is counted as a mention of all entities higher in the hierarchy too. For example, the query:

```
mentions.location: Britain
```

returns all documents mentioning not only the United Kingdom, Great Britain, the U.K. (if it is disambiguated as referring to the particular location), but also all locations known to be in the U.K. Figure 4 demonstrates such a query.

The screenshot shows a search interface with the following elements:

- Search Bar:** Contains the query `people; mentions.location.country: Ireland`.
- Search Buttons:** "External SEC", "External", "Internal", and a "Search" button.
- Results:** 3 results are displayed.
 - Result 1:** "Killary, near the Mouth of the Bundoracha River, Connemara". Creator: William Evans of Eton. Description: "Data source: National Gallery of Ireland gives a real sense of the living conditions of the ordinary people of Connemara who".
 - Result 2:** "The Liffey Swim". Creator: Jack Butler Yeats. Description: "Data source: National Gallery of Ireland of Dublin in the early 1920s, the Liffey Swim was a celebratory event, in which citizens".
 - Result 3:** "Morning in a City". Creator: Jack Butler Yeats. Description: "Data source: National Gallery of Ireland in Fitzwilliam Square, in the centre of Georgian Dublin. His city subjects assumed a romantic mood".
- Facets Panel:**
 - Where found:** External (3)
 - Has image:** True (3)
 - Source dataset:** NGI (3)
 - Creator:**
 - Jack Butler Yeats (2)
 - William Evans of Eton (1)

Figure 4: A combination of a keyword-based search and a query on the semantically enriched content

Inferred knowledge (corresponding to entity attributes stored in the knowledge base) can be queried in a similar way. For example, the query:

```
mentions.person.nationality: Irish
```

finds documents mentioning any known (art-related) person whose nationality is Irish.

The semantic enrichment is also applied to specific fields so that it is possible to formulate advanced queries such as:

```
person.influenced_by: Pablo Picasso; genre: portrait
```

which would search for portraits by artists influenced by Pablo Picasso.

8 Annotation suggestions

Current information extraction technology is not able to cope with all text mining tasks. Fully automatic semantic enrichment has therefore its limits. This is especially true for mining complex relations such as events from a text. As discussed in deliverable D4.2.1, the performance of state-of-the-art event extraction systems rarely overcomes 60 %. This is not acceptable in many cases. Consequently, many professional domain specialists prefer computer-assisted annotation, that can be manually tuned, over fully automatic processes.

There are at least three particular cases in which an automatic text mining scenario cannot be (fully) applied. First, a variability of natural language constructs to express a semantic relation can be high and there can be not enough data to train a machine learning model. For example, relations of artistic influences (among artists, artworks, themes, styles, techniques, and places) have been studied within the DECIPHER project and it showed up that despite the effort, expressions such as *pays tribute/homage to* are not well covered in results. Although various bootstrapping approaches on web-scale data provide a help [13], at least an initial seed of examples needs to be provided by users. Manual annotation serves then clearly as a source of more training data and generally improves performance of automatic annotation procedures.

Second, the structure of knowledge (a template to be filled in by an automatic procedure) can be complex and natural language processing and machine learning techniques can be unable to deal with it. In the cultural heritage domain, a typical example is a knowledge scheme analysing different attitudes to an artist and his or her work. Many books can be written about the topic, people can have opposite meanings and it is very difficult for automatic methods to generalize in such situations. As complex structures often consist of sub-components that can be recognized automatically, annotation suggestions significantly speed up semantic enrichment in this case.

Finally, the knowledge structure itself can be unclear, not well understood, or fuzzy. When annotating particular pieces of relevant texts, users often become aware of a general semantic pattern of knowledge represented by the texts, they better realize what attributes are crucial for a task in hand and can easily draft a knowledge scheme that reflects their specific needs. This aspect showed many times in DECIPHER. Although we maximally re-used existing knowledge resources such as well-established ontologies and conceptual hierarchies (CIDOC CRM⁴³ or Getty AAT⁴⁴), many tasks required specific knowledge structures created “on the fly”. One can-

⁴³<http://cidoc-crm.org/>

⁴⁴<http://www.getty.edu/research/tools/vocabularies/>

not expect that ordinary end-users will adopt sophisticated ontology tools such as Protégé to make or propose additions to knowledge specification schemata. It is thus beneficial if an annotation tool is able to play this role too.

The 4A annotation system introduced in this section reflects the needs discussed above. A modular design enables extending its functionality to deal with requirements of new environments or user groups. The system can run as an independent tool but it can be also run as an integral part of the DECIPHER system, available whenever a user edits a text in Storscope.

8.1 System architecture

The 4A system consists of a back-end server generating annotation suggestions and client modules presenting the suggestions to users, transferring their feedback and visualising results. A general architectural schema is presented in Figure 5.

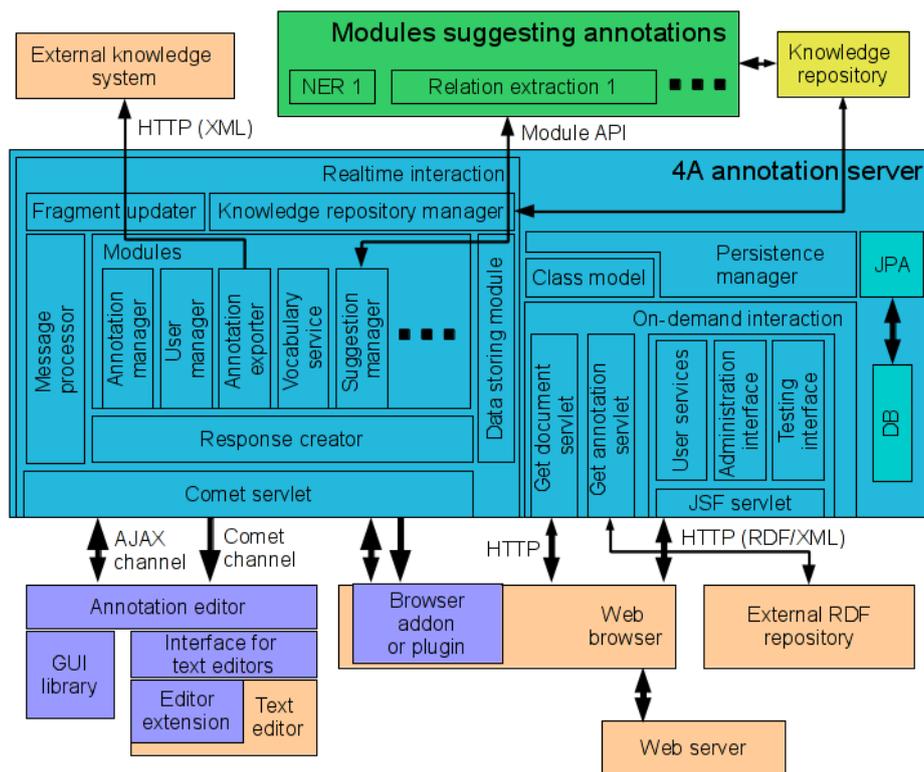


Figure 5: Architecture of the 4A system

The server side uses Modules suggesting annotations that provide semantic enrichment components – named entity recognizers and relation extractors – pre-annotates input texts, marks potential occurrences of relevant entities and relations, and returns them through a unified interface.

Two kinds of clients access the server. The first group is formed by clients intended for annotating existing web pages or documents viewed in a browser. A client realized as an add-on for Mozilla Firefox is currently available, MS Internet Explorer and Opera versions are being developed and Google Chrome is in a long-term plan. The second group of clients enables editing and annotating a text at the same time. Currently, a JavaScript WYSIWYG editor plugin is implemented for TinyMCE⁴⁵ used in Storyscope. Other editors can be supported through a defined abstraction level. The second form of clients provides also a way of integration into popular content management systems (such as Drupal⁴⁶ in the case of DECIPHER).

The 4A Annotation server is logically subdivided into two main blocks. Key modules, detailed in following paragraphs, take care of real-time interaction. Other modules deal with on-demand requests, store the class model, and provide an abstraction over the Java Persistent API (JPA).

The Comet servlet handles client requests. It distinguishes AJAX and Comet requests and returns responses in respective forms. It is also responsible for keeping the Comet connection alive. AJAX requests are handed over to the Message processor to be parsed and then sent to the Response creator to generate an appropriate response which is sent back to the client. Comet requests are used to retrieve data. The servlet checks if it is already available and, if it is, it sends it immediately back to the client.

The Message processor parses client requests and checks whether the data is complete and whether it corresponds to a given request type. Potential errors are reported in a generated XML which is then sent to the client. Instructions for other server modules is included in the output (requested operations, parameters and data).

The Response creator gets data created in the request processing and dispatches it to respective modules. Each processing module adds its output to a shared data structure so that other modules can benefit from it. When all required outputs are collected, the Data storing module is called to save the data. It is done in one transaction to guarantee consistency of the data.

The Annotation manager forms a core of the 4A server. It implements synchronization steps, computes differences between document versions, updates annotations and sends results back to clients. It also validates new or updated annotations by checking whether all required attributes are pro-

⁴⁵<http://www.tinymce.com/>

⁴⁶<http://drupal.org/>

vided, whether all links are correct and so on. Requests indicating new annotation types, requests for reloading annotations, or applying user settings, are processed as well. The module responds to Comet requests with newly added, changed or deleted annotations and semantic types. It analyses which annotation updates a particular client should receive. Clients can subscribe to their own annotations only, all accessible annotations from specified users or user groups, or annotations of given types.

The User manager module takes care of users and user groups. It responds to requests for specific user or group information.

The Annotation exporter is used to inform external systems about new or changed annotations. It sends information about creating, updating or deleting annotations immediately after completing the action. It can be set to send only specified information (given by annotation sources or types). The output is self-contained and optimised for stateless processing by external systems. For example, a subscribed system does not need to construct a full hierarchy of annotated concepts – annotation dependencies are always contained in transferred data.

The Vocabulary service makes available controlled vocabularies stored in the Knowledge repository. It processes requests for lists of values and auto-completion of particular search fields. It also provides a service for creation of annotation suggestions from the output of Modules suggesting annotations if it contains references to the vocabulary.

The Suggestion manager processes suggestion requests and the user feedback. It calls Modules suggesting annotations to receive generated suggestions, sends them to clients and stores them for a future use. Users can accept annotations in the proposed form, modify them, or reject a suggestion altogether. The feedback is stored and used for improving future suggestions. If the document is updated, the module is responsible for suggestion updates that minimise losses of the user feedback in untouched parts of the text.

The Fragment updater is a library manipulating documents and their fragments. It provides specific format parsers, iterators over mark-up structures, methods for searching strings and so on. There is also a highly configurable engine for fragment searching and updating.

On the client side, there is a generic abstraction level for text editors that implements functions relevant for all clients of this kind. Specific support for particular editors (such as the TinyMCE JavaScript editor) then builds on the intermediate level.

The Annotation editor is a component added to a WYSIWYG text editor (for example, TinyMCE) that provides the client side of the whole annotation system in DECIPHER. It communicates with the Annotation server using two channels. The first one sends AJAX requests to the server. The second channel is initialized after the session starts. It takes care of asynchronous updates sent by the server to the client.

8.2 Computer-assisted annotation of text

An essential feature of the 4A system lies in anchoring annotations in text. Although a resulting knowledge structure in RDF can be attached to the whole document, it is more useful to provide semantic interpretation directly to particular words, phrases, sentences, paragraphs or any other piece of text (referred to as textual fragments in this section). The fine-grained annotation is critical for further processing of semantically enriched data and accountability of results. Moreover, pinpointing a source of information helps machine learning methods to infer better models from annotated examples.

A text can come into existence and be immediately annotated or it can be modified while it is supposed to preserve all untouched annotations. Texts and annotations need to be taken separately to support interleaved editing and annotating. A sophisticated annotation management has to guarantee that most of annotations remain valid after a text editing step and only disqualified annotations will be thrown away. The 4A server takes care of annotation updates. To find the best match between a stored and an edited version of an annotated text, a cascade of methods with variable sensitivity is applied. A current node in a hierarchical representation of the text is searched forward and backward first. If no match is found, the content is searched from the current node to other nodes. A fragment in an edited text is taken as a match if its Levenshtein distance from a fragment in an original annotated text is lower than a pre-defined threshold value.

Although annotation structures can be complex, accepting or rejecting suggestions needs to be very easy – it should correspond to one click in most cases. As automatic methods never recognize all potential entities, there also needs to be a simple mechanism for adding new entries into an underlying knowledge base and to maximally reuse existing content. To realize this, the 4A system employs semantic templates that lead the user through the annotation process. For example, when annotating a text corresponding to creating an artwork, the system displays common attributes from the CIDOC CRM. When clicking on attribute `creator` which is known to be typically filled by a URI corresponding to a person, the system suggests primarily the fragments that would fit this semantic preference. Semantic templates, derived from ontologies in an initialization phase, effectively get users over complexities of existing knowledge structures – concepts are suggested as semantic types of attributes, constraints are transformed into template structures and existing annotations are used to infer preferences. Any use of an attribute is also linked back to the original ontology. Thus, suggested changes in knowledge structures can be immediately supported by real world examples.

Annotations can link to other annotations that exist either independently or just as a part of a superordinate annotation. The 4A system sup-

ports unlimited nesting of annotations. For example, an annotation referring to an event can then include annotations of complex attributes and, at the same time, form a part of another annotation expressing a cause relation between two events, which is further attributed to a belief state of a person.

There is also a non-trivial support for overlapping and interleaving annotations in the 4A system. To preserve well-formedness rules of XML documents and other hierarchical formats, the 4A system automatically splits fragments on “the seams” and joins the parts again when representing annotation results. For example, the advanced mechanism enables annotating a sentence *Renoir and Manet made copies of Delacroix’ paintings* by two separate events with different actors (*Renoir* and *Manet*) and shared textual fragments representing the verb phrase and the object of the relations.

Annotation attributes can also be of a specific data type (String, Date, GeoPoint...) and they can be manually entered by users if not explicitly mentioned in a text fragment being annotated. The annotation editor can display such attributes in a specified form (for example, show a GeoPoint on a map). It can also export annotations in a format suitable for visualising by external tools (for example, graph structures of artistic influence relations). Visual representations of identified entities and relations, if available, form a part of the exporting format.

Annotations are displayed in popups on moving the mouse over a relevant fragment (see Figure 6). Links to other annotations can be clicked on and any content of linked and nested annotations can be displayed directly in a particular annotation popup too. Document-level annotations are displayed in a separate window.

Annotation suggestions can be generated only for specified semantic types (and dependent entities) and/or for a given part of a text, depending on parameters set by the user. Annotated fragments can be also highlighted by various means. A particular way is, again, under full user control. The customization involves setting foreground and background colours for specific semantic types and all potential attributes of fonts. Other parameters of the 4A annotating environment can be customized as well such as variants of toolbars or folding of annotations.

Suggested annotations can be accepted one by one (either directly in the text or in a separate window with a list of all suggestions for a quick review) or the system can be set to confirm all suggestions with a confidence value higher than a specified threshold. Users can further manually delete annotations confirmed in the previous step. Similarly, all low-confidence suggestions can be rejected automatically. In any case, the user feedback is always stored and used for improving automatic suggestions.

To prevent problems related to concurrent work of users on the same document, the 4A Annotation editors and the server employ a real-time protocol and send changes immediately to all involved parties. If a new

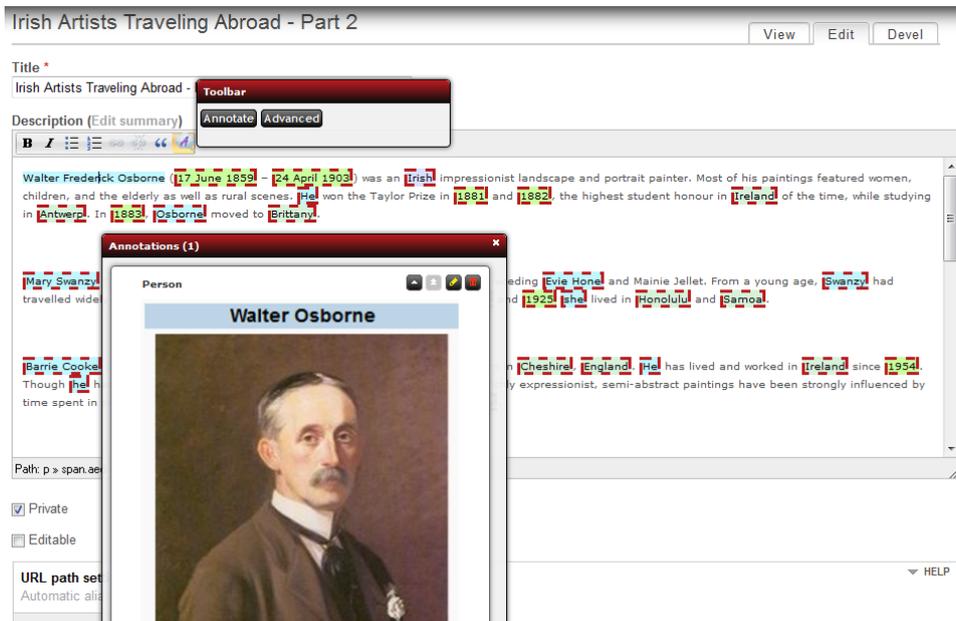


Figure 6: Annotation suggestions showed in popup windows on moving the mouse over a relevant fragment

user opens the same document, the system offers applying changes of others or creating a new copy of the same text.

9 Conclusions

The Semantic Annotator described in this deliverable was successfully deployed within the DECIPHER project. It is currently used for various annotation tasks in the developed system. In particular, the Content Aggregator calls the implemented web services to semantically enrich specific fields of collected records. The module introduced in Section 7 demonstrates advanced features of the fulltext search in the semantically-enriched content. Last but not least, the Semantic Annotator has been successfully applied for annotation suggestions employed in the Annotation editor. Museum professionals can further exploit advanced features of the tool and introduce it to new environments of their interest.

The described components and tools were integrated into Storyscope and will be evaluated in the final (field) trials. There are plans for further improvements to the integration and UI elements of the SEC tools included in the Exploitation planning for the project. These would see the annotation tools streamlined and made more accessible to beginner and intermediate users of the overall system.

The knowledge base used for text mining needs to be regularly updated. This will be linked to the planned opening of various external resources such as that from Getty⁴⁷ and releases of new versions of data sources already employed (for example, DBpedia). Co-reference resolution and disambiguation algorithms will be also enhanced. In particular, we will integrate new techniques for semantic relatedness explored in a related research work [9].

Various improvements are also planned for integration into the next versions of the 4A annotation system. For example, they will bring advanced export functions for ontology extensions proposed by users. The support for new WYSIWYG editors such as Aloha⁴⁸ or CKEditor⁴⁹ will also be extended to realize the potential of the 4A system in new settings.

⁴⁷<http://www.getty.edu/about/opencontent.html>

⁴⁸<http://www.aloha-editor.org/>

⁴⁹<http://ckeditor.com/>

References

- [1] CHAKRABARTI, S., KASTURI, S., BALAKRISHNAN, B., RAMAKRISHNAN, G., AND SARAF, R. Compressed data structures for annotated web search. In *Proceedings of the 21st international conference on World Wide Web* (New York, NY, USA, 2012), WWW '12, ACM, pp. 121–130.
- [2] CHINCHOR, N., AND HIRSCHMANN, L. MUC-7 coreference task definition, version 3.0. In *Proceedings of MUC* (1997), vol. 7.
- [3] DACIUK, J., WATSON, R. E., AND WATSON, B. W. Incremental construction of acyclic finite-state automata and transducers. In *Finite State Methods in Natural Language Processing* (1998), Bilkent University, Ankara, Turkey.
- [4] DENG, D.-P., MAI, G.-S., HSU, C.-H., CHANG, C.-L., CHUANG, T.-R., AND SHAO, K.-T. Linking open data resources for semantic enhancement of user-generated content. In *JIST* (2012), H. Takeda, Y. Qu, R. Mizoguchi, and Y. Kitamura, Eds., vol. 7774 of *Lecture Notes in Computer Science*, Springer, pp. 362–367.
- [5] DODDINGTON, G. R., MITCHELL, A., PRZYBOCKI, M. A., RAMSHAW, L. A., STRASSEL, S., AND WEISCHEDEL, R. M. The automatic content extraction (ACE) – program-tasks, data, and evaluation. In *LREC* (2004).
- [6] GRISHMAN, R., AND SUNDHEIM, B. Message understanding conference-6: A brief history. In *COLING* (1996), vol. 96, pp. 466–471.
- [7] HALPIN, H., HAYES, P. J., MCCUSKER, J. P., MCGUINNESS, D. L., AND THOMPSON, H. S. When OWL:sameAs isn't the same: An analysis of identity in linked data. In *International Semantic Web Conference (1)* (2010), P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Z. 0007, J. Z. Pan, I. Horrocks, and B. Glimm, Eds., vol. 6496 of *Lecture Notes in Computer Science*, Springer, pp. 305–320.
- [8] HOVY, E., MARCUS, M., PALMER, M., RAMSHAW, L., AND WEISCHEDEL, R. OntoNotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers* (2006), Association for Computational Linguistics, pp. 57–60.
- [9] MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. Efficient estimation of word representations in vector space. In *Proceedings of ICLR* (2013). <http://arxiv.org/abs/1301.3781>.

- [10] RECASENS, M., MÀRQUEZ, L., SAPENA, E., MARTÍ, M. A., TAULÉ, M., HOSTE, V., POESIO, M., AND VERSLEY, Y. SemEval-2010 task 1: Coreference resolution in multiple languages. In *Proceedings of the 5th International Workshop on Semantic Evaluation (Stroudsburg, PA, USA, 2010)*, SemEval '10, Association for Computational Linguistics, pp. 1–8.
- [11] VISSER, U. *Intelligent information integration for the Semantic Web*, vol. 3159. Springer, 2004.
- [12] WEISCHEDEL, R., HOVY, E., MARCUS, M., PALMER, M., BELVIN, R., PRADAN, S., RAMSHAW, L., AND XUE, N. Ontonotes: A large training corpus for enhanced processing. *Handbook of Natural Language Processing and Machine Translation*. Springer (2011).
- [13] ZHU, J., NIE, Z., LIU, X., ZHANG, B., AND WEN, J.-R. StatSnowball: A statistical approach to extracting entity relationships. In *Proceedings of the 18th International Conference on World Wide Web (New York, NY, USA, 2009)*, WWW '09, ACM, pp. 101–110.