Using Linked Data for Named Entity Extraction and Disambiguation

Manolis Baritakis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, Heraklion, GR-70013, Greece

Thesis Advisor: Assistant Prof. *Yannis Tzitzikas*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

Using Linked Data for Named Entity Extraction and Disambiguation

Thesis submitted by
Manolis Baritakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Manolis Baritakis

Committee approvals: _____
Yannis Tzitzikas
Assistant Professor, Thesis Supervisor

_____
Dimitris Plexousakis
Professor, Committee Member

_____
Irini Fundulaki
Principal Researcher, Committee Member

Departmental approval: _____
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, February 2016

Using Linked Data for Named Entity Extraction and Disambiguation

Abstract

Named Entity Extraction (NEE) is the process of identifying entities in texts and, very commonly, linking them to related (Web) resources. This task is useful in several applications, e.g. for question answering, annotating documents, processing of search results, etc. However, it is quite common for an entity name to correspond to more than one semantic categories, e.g. Argentina may refer either to *Fish Species* Argentina or to *Country* Argentina. This is the well-known Named Entity Disambiguation (NED) problem. In addition to, existing NEE and NED tools lack an open or easy configuration although this is very important for building domain-specific applications. For example, supporting a new category of entities, or specifying how to link the detected entities with online resources, is either impossible or very laborious. In this thesis we show how we can exploit semantic information (Linked Data) at real-time for configuring a NEE system and disambiguating the mined entities. We introduce an RDF/S vocabulary, called "Open NEE Configuration Model", which allows a NEE service to describe (and publish as Linked Data) its entity mining capabilities, but also to be dynamically configured. We present `X-Link` a NEE framework that realizes this model, and contrary to the existing tools, it allows the user to define the categories of entities that are interesting for the application at hand (by exploiting Linked Data). Then we focus on the problem of NED in this context, i.e. on the problem of selecting the right category for each extracted entity. To this end we introduce three methods, each approaching the problem from a different perspective. The first method is based exclusively on NEE results and selects as more probable category the one with the highest occurrence frequency. The second method moves a step forward and exploits the semantic relations between the mined entities, using their semantic resources, and returns the semantic resource that is closer to the others in the semantic graph. The last method uses machine learning algorithms for classifying the entire document into a specific category based on a train set. Then we report the results of a thorough comparative experimental evaluation using search results from Bing[1] search engine. We evaluate the introduced methods over collections of documents of different size and we measured the achieved precision and the required time for disambiguation. The results allowed us to identify the strong and weak aspects of each method. Overall, the third method works well in most cases apart from small snippets, e.g. tweets, where it achieves almost the same precision with the second method.

[1] https://www.bing.com

Χρήση Διασυνδεδεμένων Δεδομένων για Εξόρυξη και Αποσαφήνιση Οντοτήτων

Περίληψη

Με τον όρο Εξόρυξη Οντοτήτων αναφερόμαστε στη διαδικασία εντοπισμού οντοτήτων[2] σε κείμενα και αρκετά συχνά στην σύνδεσή τους με σχετικούς (διαδικτυακούς) πόρους. Αυτή η διαδικασία είναι χρήσιμη σε πολλές εφαρμογές, όπως στην απάντηση επερωτήσεων, στην επισημείωση κειμένων, στην επεξεργασία αποτελεσμάτων αναζήτησης, κ.α. Ωστόσο, είναι αρκετά σύνηθες ένα όνομα οντότητας να αντιστοιχεί σε παραπάνω από μια κατηγορίες, λόγου χάρη ο όρος Argentina μπορεί να αφορά είτε το είδος ψαριού Argentina, είτε την ομώνυμη χώρα. Αυτό το πρόβλημα είναι γνωστό στη κοινότητα ως πρόβλημα της Αποσαφήνισης Οντοτήτων. Επιπρόσθετα, τα υπάρχοντα εργαλεία εντοπισμού και αποσαφήνισης οντοτήτων στερούνται μιας εύκολης και «ανοικτής» παραμετροποίησης, η οποία είναι σημαντική για τη δημιουργία εξειδικευμένων εφαρμογών. Για παράδειγμα, η υποστήριξη μιας νέας κατηγορίας οντοτήτων ή ο προσδιορισμός του τρόπου σύνδεσης των οντοτήτων με δεδομένα στο διαδίκτυο, είναι από πολύ δύσκολο έως ακατόρθωτο. Σε αυτήν την εργασία επικεντρωνόμαστε στο πως μπορούμε να εκμεταλλευτούμε τις διαθέσιμες σημασιολογικά οργανωμένες πληροφορίες, συγκεκριμένα τα Διασυνδεδεμένα Δεδομένα (Linked Data), για να παραμετροποιήσουμε ένα σύστημα εξόρυξης οντοτήτων καθώς και για να αποσαφηνίσουμε τις ευρεθείσες οντότητες. Προτείνουμε μια οντολογία RDF/S, που ονομάζεται "Open NEE Configuration Model", η οποία επιτρέπει σε μια υπηρεσία εντοπισμού οντοτήτων να περιγράφει (και να εκφράζει ως Linked Data) τις προδιαγραφές της, καθώς και να παραμετροποιείται δυναμικά. Επίσης παρουσιάζουμε το `X-Link`, ένα εργαλείο εξόρυξης οντοτήτων που υιοθετεί το παραπάνω μοντέλο, που σε αντίθεση με άλλα συναφή εργαλεία, επιτρέπει στον χρήστη να προσδιορίζει τις κατηγορίες οντοτήτων που τον ενδιαφέρουν για την εφαρμογή του (εκμεταλλευόμενος τα Linked Data). Εν συνεχεία, κινούμενοι ως προς αυτή την κατεύθυνση, εμβαθύνουμε στο πρόβλημα της αποσαφήνισης οντοτήτων, και πιο συγκεκριμένα στο πρόβλημα της επιλογής της κατάλληλης κατηγορίας για κάθε ευρεθείσα οντότητα. Για τον σκοπό αυτό προτείνουμε 3 μεθόδους, με κάθε μια να προσεγγίζει το πρόβλημα από διαφορετική σκοπιά. Η πρώτη βασίζεται εξολοκλήρου στα αποτελέσματα ενός NEE εργαλείου και θεωρεί ως πιθανότερη κατηγορία εκείνη με την μεγαλύτερη συχνότητα εμφάνισης στα αποτελέσματα. Η δεύτερη επεκτείνει την πρώτη και αξιοποιεί τις σημασιολογικές συσχετίσεις μεταξύ των οντοτήτων που έχουν εντοπιστεί, χρησιμοποιώντας τις σημασιολογικές τους ιδιότητες. Θεωρεί ως πιο πιθανή κατηγορία εκείνη που αντιστοιχεί στον σημασιολογικό πόρο που είναι πιο κοντά (στο σημασιολογικό γράφο) στους υπόλοιπους που εντοπίστηκαν. Η τελευταία μέθοδος χρησιμοποιεί αλγορίθμους μηχανικής μάθησης για την κατηγοριοποίηση του εκάστοτε κειμένου σε μια συγκεκριμένη κατηγορία, έχοντας πρώτα «εκπαιδευτεί» σε μια κατάλληλη συλλογή εγγράφων. Στη συνέχεια παρουσιάζουμε τα αποτελέσματα μιας εμπεριστατωμένης συγκριτικής αξιολόγησης που

[2]Οτιδήποτε υπάρχει αυτοτελώς, έχει διακεκριμένα χαρακτηριστικά που το ορίζουν ως αυτοτελή ύπαρξη.

χρησιμοποιεί αποτελέσματα αναζήτησης από τη μηχανή αναζήτησης Bing. Τα αποτελέσματα της αξιολόγησης μας επιτρέπουν να εντοπίσουμε τα θετικά και τα αρνητικά κάθε μεθόδου. Πιο συγκεκριμένα, αξιολογήσαμε τις μεθόδους μας πάνω σε συλλογές εγγράφων διαφορετικού μεγέθους και υπολογίσαμε την ακρίβεια τους καθώς και τον απαιτούμενο χρόνο εκτέλεσης. Μετά το πέρας των πειραμάτων καταλήξαμε στο συμπέρασμα ότι η τρίτη μέθοδος (κατηγοριοποίηση εγγράφου) λειτουργεί καλύτερα σε όλες τις περιπτώσεις εκτός αυτής που το περιεχόμενο ενός εγγράφου είναι περιορισμένο, πχ. tweets, όπου έχει σχεδόν την ίδια ακρίβεια με την δεύτερη μέθοδο.

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον επόπτη καθηγητή μου κ. Γιάννη Τζίτζικα για την ορθή καθοδήγηση και ουσιαστική συμβολή του στην ολοκλήρωση της παρούσας διατριβής. Επιπλέον, να εκφράσω τις ευχαριστίες μου στον κ. Δημήτρη Πλεξουσάκη και στην κ. Ειρήνη Φουντουλάκη για την προθυμία τους να συμμετέχουν στην τριμελή επιτροπή, καθώς και στον διδακτορικό φοιτητή Παύλο Φαφαλιό για την άψογη συνεργασία και την πολύτιμη βοήθεια του.

Ακόμα να ευχαριστήσω το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας για την πολύτιμη υποστήριξη σε υλικοτεχνική υποδομή και τεχνογνωσία, καθώς και για την υποτροφία που μου προσέφερε καθ' όλη τη διάρκεια της μεταπτυχιακής μου εργασίας.

Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την υποστήριξη, την συμπαράσταση και την εμπιστοσύνη που μου έδειξαν όλα αυτά τα χρόνια των σπουδών μου, αλλά και για την αγάπη τους σε κάθε βήμα της ζωής μου. Σας ευχαριστώ πολύ για όλα.

στους γονείς μου

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Named Entity Extraction (NEE), also known as Named Entity Recognition (NER) and Semantic Annotation, is the process of identifying entities in text belonging to a set of pre-defined categories (class labels) such as Person, Location, Organization, etc. This task usually includes the Entity Linking process which tries to link the named entity with a resource (reference) in a Knowledge Base (KB)[1]. Entity Linking is also considered a way of Named Entity Disambiguation (NED), since a resource (e.g. a URI or a Wikipedia page) can determine the identity of an entity. Specifically, NED deals with the selection of right category for each extracted (ambiguous) entity based on the textual content and the semantics of the document that was found, since an entity name may correspond to more than one categories. NEE is useful in several tasks, e.g. for question answering [48], post-processing of search results [27, 30], annotating (Web) documents [35, 45]. In addition, the importance of NEE, especially for the Semantic Web, is justified by the fact that the Semantic Web realization highly depends on the availability of metadata (structured content in general) describing Web content, defined through a formal semantic structure. Thus, a major challenge for the Semantic Web is the extraction of structured data through the development of automated NEE tools.

There are already several tools that support NEE, NED or both, e.g. DBpedia Spotlight [46], AlchemyAPI [1], OpenCalais [4], Babelfy [49] and AGDISTIS [61]. However, these tools do not allow the user/developer to easily configure them, e.g. to define their own interesting types (categories) of entities (e.g. Swedish First Names), to *extend* an existing category with additional entities coming from a new KB, or to determine how to disambiguate the identified entities according to his needs and the available information. Hence, it is quite difficult to configure them for building domain specific applications. Furthermore, they do not publish in a standard format the "entity mining" capabilities of their (Web) services. Consequently, an application cannot dynamically discover and use the services that best satisfy its annotation needs.

Since a lot of information about *named entities* is already available as Linked

[1]From now on, we consider as NEE the process that includes both NER and Entity Linking.

Open Data (LOD) [17], the exploitation of LOD by a NEE system could bring wide coverage and fresh information. However, existing LOD-based NEE systems (e.g. DBpedia Spotlight) are mainly dedicated to one specific KB which is indexed beforehand, not exploiting thereby the dynamic and distributed nature of LOD. For instance, consider a NEE system that supports a category of entities X. Consider now that a new KB appears which contains plenty of information for entities belonging to X. It would be useful if one could somehow "plug" the new KB in the NEE system (with the less possible effort), enabling thereby the linkage of the identified entities with resources in the new KB. Moreover, the information that the existing NEE systems return for the identified entities is not rich enough and cannot be controlled. For example, one cannot configure the properties that are useful for a particular application, e.g. to restrict the properties to only images or related entities, or properties in a specific natural language, or to inspect whether and how the identified entities are connected, not within the document but as entities in general. However, these correlations between entities can be useful for the disambiguation process.

To tackle this lack of functionality, in this thesis:

- We elaborate on exploiting the LOD at real time for configuring a NEE system and we propose a *generic (abstract) configuration model*. We also discuss *ranking* issues that arise within this context.
- We propose the *Open NEE Configuration Model*, an RDF/S [5] vocabulary which allows a NEE system to describe (and publish as Linked Data) the entity mining capabilities of its services.
- We present `X-Link`, a fully configurable (LOD-based) NEE framework that we have designed and implemented which realizes the proposed configuration model.
- In addition, we propose a set of methods for disambiguating named entity appearances in documents, and we report the results of a comparative experimental evaluation, where we measure their achieved precision and the required execution time over datasets of different length (scales of granularity).

For justifying the value of the proposed approach, below we first present a vertical search scenario that stresses that different communities have different and ever-changing requirements, and then we discuss several benefits of adopting an open and exchangeable configuration model.

1.1 The Value of Configurability in Vertical Search

The motivation for enhancing *configurability* can be made evident from the following scenario, which is a real scenario related to the `iMarine` project[2]:

[2] `http:/www.i-marine.eu/`

Consider that you are responsible for maintaining a search system, called `X-Search`*, a meta-search system that receives a keyword-based query, sends the query to one or more marine sources and retrieves the results. For giving users an overview of the search results and allowing them to explore them in a faceted way, you want to use a NER tool for identifying (at real time)* fish species *in the snippets or the full contents of the top results. You think that it would be also useful to link (on demand) the identified species with related semantic resources, as well as to retrieve more information (e.g. a short description of the species, an image, its taxonomy, etc.) by querying (at real-time) online semantic KBs. Figure 1.1 depicts a screenshot of* `X-Search` *for the query "tuna species". The user can see (in a left bar) the fish species identified in the search results and can also explore an identified species at real-time (the species "Atlantic bluefin tuna" in this example).* ◇

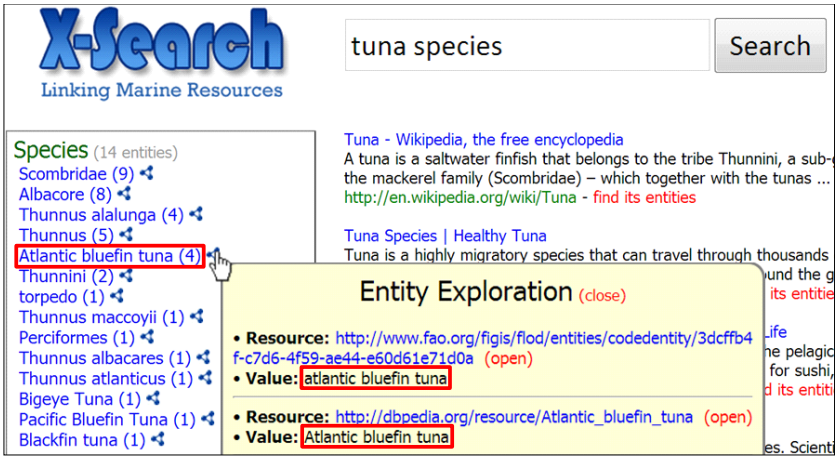

Figure 1.1: Semantic post-processing of search results (for the query *tuna species*) and exploration of the entity *Atlantic bluefin tuna* in `X-Search`.

However, each community of users (e.g. an organization or an institution) has *different needs*, which in our scenario means that `X-Search` should support different configurations. For instance, scientists in an organization may also want to inspect other categories of entities in the search results (apart from fish species), e.g. water areas and countries, and define explicitly the way of disambiguation of ambiguous entities. In addition, different communities/users may want to link and enrich the identified species with resources from different sources; one may want images from `DBpedia` [14], others with papers that describe the genome of the species.

For coping with the above requirements, we would like to be able to easily configure `X-Search` for satisfying the needs of each community of users. In addition, and since the needs of a community constantly change, we would like to be able to dynamically change the configuration at any time without requiring to redeploy

the system (e.g. for updating the list of fish species, for specifying another KB, etc.). It would be also useful if `X-Search` could dynamically (ideally at query-time) discover the NEE services to use according, for example, to the user information needs. For instance, if a user submits a query requesting documents about water areas, `X-Search` could select to use a service that supports identification of water areas. Finally, by accessing the output of the NEE process in RDF [6], `X-Search` could offer advanced exploratory search services over the annotated results. For example, a user could select to inspect *"all results containing information about fish species of genus* `Thunnus`*"*.

In this thesis we present one method to accomplish this scenario.

1.2 The Value of Named Entity Disambiguation

Suppose that the above search results also include the entity *Argentina*. According to our configuration, `X-Search` will assume that it refers to `fish species`, since it is the only supported category. However, as we know *Argentina* is also a country of South America. At this point occurs the problem that the same name can denote entities belonging to different categories. Thus, in such cases we have to find the real category of each entity occurrence. This problem is known as Named Entity Disambiguation or Word-Sense Disambiguation problem, and we will thoroughly deal with it in Chapter 5.

1.3 The Value of having Exchangeable/Portable Configurations

Having open and exchangeable configurations offers many benefits including:

- *Exchangeability and Portability.* Configurations can be exchanged by users/communities, e.g. for annotating different corpora of documents using the same configuration, i.e. the same categories, lists of entities, KBs, etc. In addition, the availability of a model like the one that we propose enables a NEE service to offer an API that accepts and uses such configurations, while the result of the annotation process can be returned in a standard format, allowing its further exploitation in several contexts.
- *Aggregation and Integration of multiple configurations.* A common model allows someone to collect such configurations (provided by different NEE systems) and then, by querying them, to select those services that satisfy the needs of the intended application.
- *Benchmarking.* Common configurations would allow comparative evaluation of different NEE systems, e.g. with respect to efficiency, effectiveness of entity disambiguation, etc.
- *Extendability.* The expression of the model as an RDF Schema allows someone to extend it by exploiting also other vocabularies.

The rest of this thesis is organized as follows: Chapter 2 discusses related works and the difference of our approach. Chapter 3 analyzes the proposed configuration model and presents the *Open NEE Configuration Model* and the extension of the *Open Annotation Data Model*, proposed in [33, 34]. Chapter 4 describes in detail the functionality and configurability of `X-Link`. Chapter 5 focus on named entity disambiguation problem. Chapter 6 reports evaluation results. Finally, Chapter 7 concludes and identifies directions for future research.

Publications Derived by this Thesis

Parts of this work were published in the international conference WIMS [33] and in the international journal IJAIT [34]. In both works we propose an approach for configuring dynamically (at run-time) a NEE tool, using Linked Data. To this end we present `X-Link` a NEE framework that realizes this approach.

Chapter 2

Related Work

There is a plethora of non LOD-based NEE tools like `Wikipedia Miner` [47], `Yahoo! Content Analysis API` [11], and `TagMe` [36]. Since the approach that we propose is based on LOD, below we first discuss the most relevant LOD-based NEE and NED tools of general purpose (§2.1), we then report some semantic annotation systems tailored for the *life sciences* domain (§2.2), and finally we discuss the main differences of our approach (§2.3).

2.1 LOD-based NEE & NED Tools of General Purpose

DBpedia Spotlight [46] is a REST API tool for annotating mentions of DBpedia resources in text, providing a solution for linking unstructured information sources to the LOD. It finds and returns entities that are found in a text, ranks them depending on how relevant they are to the text content, and links them with URIs from DBpedia. The results of the entity extraction process can be stored into various forms (HTML, XML, JSON or XHTML+RDFa). As regards configurability, users can provide whitelists (allowed) and blacklists (forbidden) of resource types for annotation. The available types are derived from the class hierarchy provided by the DBpedia Ontology. In addition, the interesting resources can be constrained using a SPARQL query. However, this configurability allows only the specification of the interesting resources from the existing ones; the user/administrator cannot add a new category of entities (e.g. describing resources coming from another KB), update a category or specify how to link and enrich the identified entities.
AlchemyAPI [1] is a Natural Language Processing (NLP) service which provides a scalable platform for analyzing web pages, documents and tweets along with APIs for integration. The retrieved entities are ranked based on their importance in the given text and the results can be stored as JSON, Microformats, XML and RDF (using a dedicated schema[1]). In addition, the named entity extractor is able to disambiguate the detected entities, link them to various datasets on the LOD and resolve co-references.

[1] `http://rdf.alchemyapi.com/rdf/v1/s/aapi-schema#`

OpenCalais: Calais [4] is a toolkit that allows incorporating semantic functionality within a blog, content management system, website or application. The OpenCalais Web Service automatically creates semantic metadata for the submitted content. Using NLP, machine learning and other methods, Calais analyzes a document, finds the entities within it and gives them a score based on their text relevance. The results can be saved as JSON, RDF (using a dedicated schema[2]), Microformats, N3 or simple text. In addition, it supports automatic connection to the LOD.

AIDA [62] is a framework and online tool for entity detection and disambiguation. Given a natural-language text, AIDA maps mentions of ambiguous names to entities registered in the YAGO2 KB [42]. It accepts plain text, HMTL as well as semi-structured inputs like tables, lists, or short XML files. AIDA is centered around collective disambiguation exploiting the prominence of entities, similarity between the context of the mention and its candidates, and the coherence among candidate entities for all mentions. The results can be stored in JSON.

Wikimeta [10] is a NLP semantic tagging and annotation system that allows incorporating semantic knowledge within a document, website or content management system. It tries to link each detected named entity with an entity in DBpedia based on a disambiguation process that is described in Charton et al. [20]. Wikimeta API is compliant with REST and the responses are formatted in XML and JSON. The datasets used to train the NLP tools of Wikimeta are derived from Wikipedia.

Lupedia [3] uses a gazetteer which is a list of surface forms associated to a subset of entities in DBpedia and LinkedMDB (a dataset that contains movies descriptions). The default configuration takes the longest sequence of consecutive words that corresponds to an entry in the gazetteer and annotates it with the corresponding entity in the KB. The results can be stored in HTML, JSON, RDFa or XML.

AGDISTIS [61] is an open-source and knowledge base independent NED framework. It combines the HITS algorithm with label expansion strategies and string similarity measures. Initially an index of semantic resources is created, using any knowledge base, this step is performed only once regardless the given document. After that, it retrieves the named entities that occur in a document and links them with the appropriate resources utilizing the previous index. Then, a disambiguation graph is formed, applying the Breadth-First-Search (BFS) algorithm in the semantic resources of retrieved entities. Finally, the graph nodes/resources are sorted by their authority score using the HITS algorithm and are kept only these with the highest score.

Babelfy [49] is a graph-based NED and entity linking approach which is based on BabelNet[3] semantic network. Each node of this network is associated with a set of other related nodes, which is called semantic signature. Afterwards, are detected all the linkable fragments of an input text, and are linked with one or

[2] `http://www.opencalais.com/files/owl.opencalais-4.3a.xml`

[3] http://babelnet.org/

more candidate meanings from BabelNet. Then, the meanings of the retrieved fragments are combined with their semantic signatures and form a graph. In the final step, a dense subgraph is extracted using various subgraph heuristics. The resulting graph contains the appropriate senses of linkable fragments.

2.2 Life Sciences-tailored Annotation Tools

Domeo Annotation Toolkit [22] is a collection of software components that enables users to create, share and curate *ontology-based* annotations for online documents. It supports fully automated, semi-automated, and manual *biomedical* annotation with full representation of the provenance of annotations, as well as personal or community annotations with authorization and access control. Annotations are represented using the Annotation Ontology (AO) RDF model [21]. However, Domeo is currently being extended to also support the Open Annotation Data Model [58]. Its user interface is an extensible web component which enables direct biomedical annotation of HTML and XML documents. Domeo performs entity mining and accesses ontologies as well as other automated markup facilities via web service calls.

Utopia Documents [13] is a desktop application for reading and exploring PDF files like scientific papers. By exploiting domain-specific ontologies and plugins, it links both explicit and implicit information (of biological or chemical interest) embedded in the articles to online resources. Utopia Documents allows editors and authors to annotate terms with definitions from online resources and allows readers to easily find these definitions. It also transforms static tables and figures into dynamic, interactive objects and simplifies the process of finding related articles by automatically linking references to their digital online versions. Via its plugins it has access to a wealth of bioinformatics data: each plugin uses appropriate client libraries to access web-service endpoints and other remotely accessible resources, such as relational databases and RDF stores.

The **NCBO Annotator** [43] is an *ontology-based* web service for annotating textual biomedical data with biomedical ontology concepts. The NCBO Annotator provides access to almost two hundred ontologies from BioPortal and UMLS and is an alternative to manual annotation through the use of a concept recognition tool. The annotator is not limited to the syntactic recognition of terms, but also leverages the structure of the ontologies to expand annotations. Such annotations allow unstructured free-text data to become structured and standardized, and also contribute to create a biomedical Semantic Web that facilitates data integration.

Whatizit [52] is a text processing system that allows a user to perform text-mining tasks. Whatizit identifies *molecular biology terms* and links them to related (publicly available) databases. The identified terms are wrapped with XML tags that carry additional information, such as the keys to the databases where relevant information is kept. Any vocabulary can be integrated into Whatizit as a pipeline and also several vocabularies can be integrated in a single pipeline. Examples

of already integrated vocabularies are *Swissprot*, the *Gene Ontology* and *Medline Plus*.

2.3 Differences of the proposed approach

The main difference of our approach is that we focus on *configurability*. Specifically, we propose a method which exploits the dynamic and open nature of LOD for specifying the entities of interest, as well as for specifying how to link and enrich the identified entities. This enhanced configurability allows the dynamic configuration of a NEE system even while a corresponding service is running. On the contrary, the configuration of the existing systems is a laborious task even for persons with computer science background and requires many technical skills. Other differences include:

- Most of the existing index semantic information (e.g. RDF triples or URIs), for example, AGDISTIS and Babelfy create indexes for the candidate meanings and labels of identified entities. However our proposed approach just indexes plain lists of entities (gazetteers) regarding only the supported categories of entities. This makes the NEE system *lightweight* and *portable*.
- By adopting the proposed approach, a NEE system can retrieve at real-time more information about the identified entities (e.g. properties and related entities) and this is configurable. On the contrary, the majority of the existing systems return only the corresponding URIs and maybe some related web pages.
- Existing systems do not describe/publish their entity mining capabilities in a standard format.

We should also stress that the Open NEE Configuration Model that we propose, as well as the extension of the Open Annotation Data Model, can be applied by existing systems. For instance, a NER system that also performs Entity Linking can describe its service through the supported categories of entities and the Knowledge Bases that it exploits. Of course, in this case it is not needed/required to also specify linking template queries since it can directly return the corresponding URI (that has been derived by the Entity Linking process). Likewise, a system that only performs NER and Word-Sense Disambiguation can be LOD aware by offering entity linking and entity enrichment capabilities. In all cases, the result of the NEE process can be described using the proposed extension of the Open Annotation Data Model. Also, in contrast to existing systems, our approach enables the user to specify the way of entity disambiguation and select the appropriate method according to his application requirements, i.e. required disambiguation time, available data, etc. For instance, a supervised method may be inappropriate for a simple user/developer, since it requires a set of hand annotated documents (concerning his categories of interest) for the training task, which is time consuming. Whereas, a company or an institution may prefer supervised methods, since they usually outperform the unsupervised ones. Finally, our approach can easily

adapted for the implementation of a non-LOD based NEE or NED tool. For instance, we can replace the SPARQL queries, that are used in our case, with SQL queries for retrieving information from relational databases apart from semantic ones.

Chapter 3

The Proposed Approach

At first we provide a few fundamental notions and notations (§3.1), then we introduce the proposed configuration model (§3.2), we give an example of that model (§3.3), we describe the semantics of such configurations (§3.4), we evaluate the URI ranking approaches (§3.5), we introduce the Open NEE Configuration Model (§3.6), and finally we present the extension of the Open Annotation Data Model (§3.7).

3.1 Notions and Notations

Since the proposed approach is based on Semantic Web technologies, below we first provide a short introduction to RDF and LOD, and then we introduce a few notions and notations regarding the NEE process.

Let us first formalize the structured knowledge available as LOD or queryable through a SPARQL endpoint [8]. Consider an infinite set U of RDF URI [9] references, an infinite set B of blank nodes [15] and an infinite set L of literals. A triple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an *RDF triple* (s is called the *subject*, p the *predicate* and o the *object*). An RDF KB K, or equivalently an *RDF graph* G, is a set of RDF triples. For an RDF Graph G_i, we shall use U_i, B_i, L_i to denote the URIs, blank nodes and literals that appear in the triples of G_i respectively. The *nodes* of G_i are the values that appear as subjects or objects in the triples of G_i.

Let now $\mathcal{C}$ be a set of *entity categories*, e.g. $\mathcal{C} = \{$Fish Species, Country, Water Area$\}$ are possible categories for the marine domain. For a category $c \in \mathcal{C}$, let $E(c)$ denote the set of *entity names* in c, e.g. $E(\text{Country}) = \{$Afghanistan, Albania, Algeria, ... $\}$. Inversely, let $ctg(e) \in \mathcal{C}$ denote the category of an entity name (e.g. $ctg(\text{Algeria}) = \text{Country}$). For an entity name e, let $U(e)$ denote the URIs that are related to e and exist in one or more RDF graphs, e.g. $U(\text{Chum Salmon}) = \{$`http://dbpedia.org/resource/Chum_salmon`, `https://www.googleapis.com/freebase/v1/rdf/m/03ysh6`$\}$. For an entity URI u, let $Descr(u)$ be a set of RDF triples that express information about u.

For an input document, say doc, we define as $Ent(doc, c)$ the set of (distinct) entity names identified in doc (by applying NER) that belong to the category c. Obviously $Ent(doc, c) \subseteq E(c)$. Thus, the set of all entities identified in doc is $Ent(doc) = \cup_{c \in \mathcal{C}} Ent(doc, c)$.

In general, in a set of documents we can identify entities of various categories, each of these entities is associated with URIs and each of these URIs with triples that describe these URIs. Specifically, if we have a set of documents D then:

- $Ent(D) = \cup_{d \in D} Ent(d)$ is the set of entities identified in D,
- $U(D) = \cup_{e \in Ent(D)} U(e)$ is the set of URIs of these entities, and
- $Graph(D) = \cup_{u \in U(D)} Descr(u)$ is a set of triples about these URIs which essentially define an RDF Graph.

Note that in many cases we have a name that corresponds to entities of different categories. For example, *argentina* may refer to the country Argentina or the fish genus Argentina. In general, a name may correspond to n categories. In such cases we consider that we have n different entities, one for each category. Therefore, each of these entities will have one category (i.e. $|ctg(e)| = 1$). This choice enables to apply afterwards disambiguation methods (more in §5.4).

3.2 The Proposed Configuration Model

Figure 3.1 shows the configuration model that we propose. Each `Category` has a name and can be associated with one or more `Knowledge Base Mirrors` (KBMs).

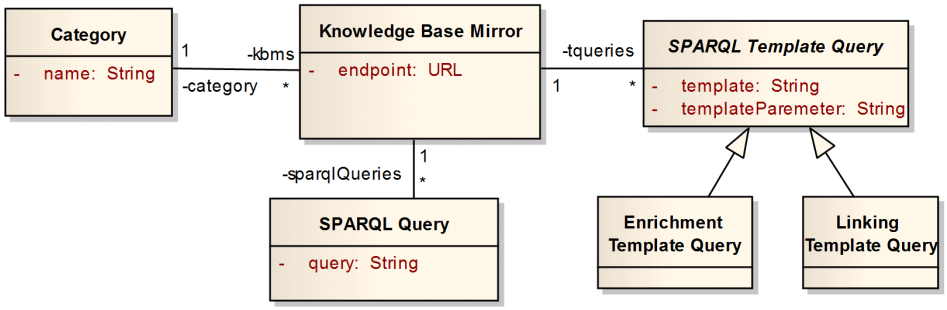

Figure 3.1: A generic (abstract) model for configuring a NEE system.

A KBM holds the URL of a SPARQL endpoint and it is associated with three kinds of elements: (a) SPARQL Queries, (b) SPARQL Template Queries for Entity Linking, and (c) SPARQL Template Queries for Entity Enrichment.

The elements of type (a) are used for specifying the *entity names* of interest by providing a KBM-answerable SPARQL query. The elements of type (b) allow specifying how entity names correspond to *entity URIs*, by providing a KBM-answerable SPARQL query. The elements of type (c) allow specifying what *extra*

information (in the form of RDF triples) should be fetched for each entity URI, by providing a KBM-answerable SPARQL query.

3.3 Example of the Configuration Model

Let's now describe an indicative instantiation of the above model. Consider a set of two categories $\mathcal{C} = \{\text{Fish Species}, \text{Country}\}$. The category Fish Species is associated with two KBMs:

- KBM_1 = `http://dbpedia.org/sparql` (SPARQL endpoint of DBpedia).
- KBM_2 = `http://www.fao.org/figis/flod/endpoint` (SPARQL endpoint of FAO FLOD [2]).

The category Country is associated with one KBM:

- KBM_3 = `http://factforge.net/sparql` (SPARQL endpoint of FactForge [16]).

For the KBM_1, we can set the SPARQL query of Figure 3.2 for specifying the fish species of interest, or the one shown in Figure 3.3 in case we are interested only in English fish names.

```
SELECT DISTINCT str(?label) WHERE {
  ?uri rdf:type <http://dbpedia.org/ontology/Fish> ; rdfs:label ?label }
```

Figure 3.2: SPARQL query for retrieving a list of fish names from DBpedia.

```
SELECT DISTINCT str(?label) WHERE {
  ?uri rdf:type <http://dbpedia.org/ontology/Fish> .
  ?uri rdfs:label ?label FILTER(lang(?label)="en") }
```

Figure 3.3: SPARQL query for retrieving a list of English fish names from DBpedia.

For *Entity Linking*, KBM_1 can be associated with the template query shown in Figure 3.4 which aims at returning URIs of type Fish whose label contains the name of an entity (ignoring case)[1]. Notice that the query contains the character sequence `[ENTITY]` (including the [and]) which is replaced (at query-time) by the entity's name. For example, by providing the string "chum salmon" as entity name, DBpedia returns the URI "`http://dbpedia.org/resource/Chum_salmon`". Of course, one could provide a "stricter" SPARQL template query, e.g. the one shown in Figure 3.5, focusing on bigger precision.

For *Entity Enrichment*, KBM_1 can be associated with the template query shown in Figure 3.6 which retrieves the *outgoing* properties of a URI[2]. Notice

[1]The results of this task are shown in the pop-up window "Entity Exploration" in Figure 3.7.

[2]The retrieved triples are those shown if the user clicks the link of a resource in the pop-up window of Figure 3.7.

```
SELECT DISTINCT ?uri WHERE {
  ?uri rdf:type <http://dbpedia.org/ontology/Fish> .
  ?uri rdfs:label ?label FILTER(regex(str(?label), "[ENTITY]", "i")) }
```

Figure 3.4: Example of a SPARQL template query for linking an identified *Fish* name with resources in DBpedia.

```
SELECT DISTINCT ?uri WHERE {
  ?uri rdf:type <http://dbpedia.org/ontology/Fish> .
  ?uri rdfs:label ?label FILTER (lcase(str(?label)) = lcase("[ENTITY]")) }
```

Figure 3.5: Example of a "stricter" SPARQL template query for linking an identified *Fish* name with resources in DBpedia.

```
SELECT DISTINCT ?propertyName ?propertyValue WHERE {
  <[URI]> ?propertyName ?propertyValue }
```

Figure 3.6: SPARQL template query for retrieving the outgoing properties of resource.

that the query contains the character sequence `[URI]` (including the [and]) which is replaced (at query-time) by the entity's URI. For example, by providing the entity URI "`http://dbpedia.org/resource/Chum_salmon`", one of the RDF triples that is returned by DBpedia is: "`http://dbpedia.org/resource/Chum_salmon` (subject) - `http://dbpedia.org/ontology/genus` (predicate) - `http://dbpedia.org/resource/Oncorhynchus` (object)".

By collecting the RDF triples that correspond to a set of entity URIs, we can form an RDF graph from which we can conclude whether and how these entity URIs are connected. For example, Figure 3.8 depicts a simple RDF graph which shows how the entities *Chum salmon*, *Chinook salmon* and *Coho salmon* are connected (for simplicity we have omitted the namespaces). Of course, one could extend this query in order to obtain more information, e.g. all information (triples) that can be reached (collected) up to a certain radius in the RDF graph.

Analogously, one can specify SPARQL queries and template queries for all KBMs related to the defined categories. Note that any of the above queries can use the *federated features* of SPARQL 1.1 [7]. This means that information from more than one SPARQL endpoints will be used.

3.4 The Semantics of the Configuration Model

A configuration essentially defines an information structure as defined in §3.1. Specifically, it defines the set of categories $\mathcal{C}$. For each category $c \in \mathcal{C}$, the corresponding set of entity names $E(c)$ is obtained by running the corresponding

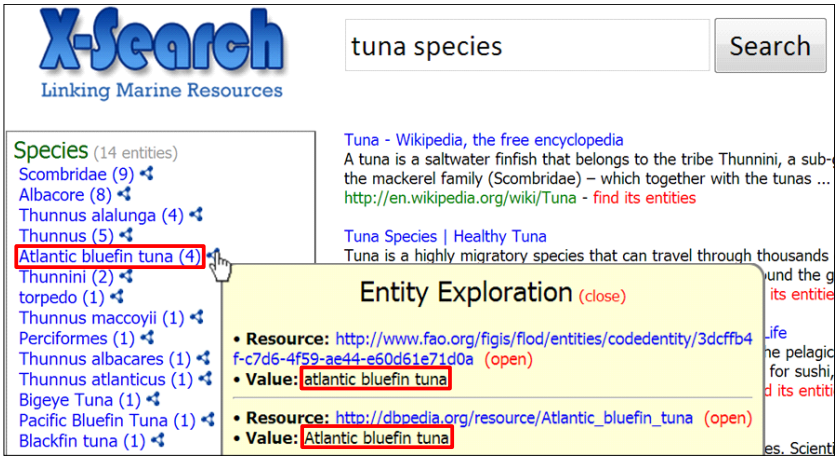

Figure 3.7: Semantic post-processing of search results (for the query *tuna species*) and exploration of the entity *Atlantic bluefin tuna* in `X-Search` (reoccurrence of Figure 1.1).

SPARQL queries to the related KBMs. For each entity name $e \in E(c)$, its linked URIs, $U(e)$, are obtained by running the corresponding Linking Template Queries (where e is passed as parameter), and for each URI $u \in U(e)$ the triples $Descr(u)$ are obtained by running the corresponding Enrichment Template Queries (where u is passed as parameter).

We should stress here that the Linking Template Queries can be also considered a way of "trivial" disambiguation that has the objective to find the resource or the resources that better characterize the entity name. However, a characteristic of this "trivial" disambiguation is that we already know the category of the corresponding entity name and thereby we can form accordingly the SPARQL template query (e.g. we can compare the entity name with the names of entities belonging to a specific RDF class, as in the template queries of Figures 3.4 and 3.5). Furthermore, we should clarify that details like the exact NER method that is applied to the document(s), or the exact NED algorithm that is used for deciding the category of a detected entity name, regard *implementation details* that must be specified by the NEE system that adopts the proposed model. For instance, one can use surface forms for NER (like DBpedia Spotlight [46]), advanced machine learning techniques for NED, etc.

Returning to our setting, for a set of documents D, $Graph(D)$ can now be defined either by collecting the triples $Descr(u)$ for each URI $u \in U(D)$, or by considering also information that can be reached up to a certain radius r. Regarding the latter, let us first introduce some notations. Let S be a set of URIs and

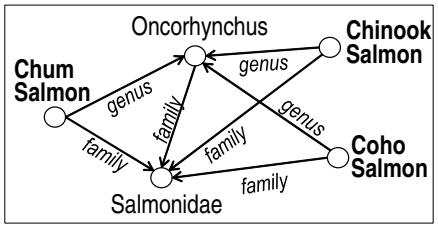

Figure 3.8: An example of an RDF graph.

G_i the RDF graph of the underlying KB. We define $In(S)$ and $Out(S)$ as follows:

$$\begin{aligned} In(S) &= \{(s,p,u) \mid u \in S, (s,p,u) \in G_i\}, \\ Out(S) &= \{(u,p,o) \mid u \in S, (u,p,o) \in G_i\} \end{aligned}$$

The description of u comprising triples that are reachable in radius 1 is defined as:

$$Descr(u,1) = In(\{u\}) \cup Out(\{u\})$$

This is generalizable to higher values of radius as follows:

$$\begin{aligned} Descr(u,r) &= Descr(u,r-1) \\ &\cup \; In(\{u' \mid (s,p,u') \; or \; (u',p,o) \in Descr(u,r-1)\}) \\ &\cup \; Out(\{u' \mid (u',p,o) \; or \; (s,p,u') \in Descr(u,r-1)\}) \end{aligned}$$

Now we can define the graph of D of radius r as follows:

$$Graph(D,r) = \cup_{u \in U(D)} Descr(u,r) \tag{3.1}$$

The value of this graph is that it makes evident how the entities are associated (more in §4.2.4).

Sometimes there is also the need to *rank* the detected entities (i.e. the elements in $Ent(D)$, or in $Ent(doc)$ if we consider a single document) and the URIs that match an entity name e (i.e. the elements in $U(e)$), and this can be configurable. The ranking information is useful, for example, for deciding which entities/URIs to promote in a displayed list, for selecting the one entity that best characterizes a document, or for selecting the one URI that best characterizes an entity name.

As regards the ranking of the detected entities, a straightforward approach is to rank them according to their frequency in the document. Let $count(e)$ be the number of occurrences of the entity name e in a set of documents D (the same approach can be applied considering a single document doc). Then, the normalized score of e can be computed as:

$$score(e) = \frac{count(e)}{\sum_{e' \in Ent(D)} count(e')} \tag{3.2}$$

Of course, several other approaches can be examined, e.g. taking into account the application context or the positions of the entities in the document(s), however this is out of the scope of this thesis.

As regards the ranking of the entity URIs, at first we should stress that both the number of the URIs that match an entity name and their quality (in terms of relevance) highly depend on the KBs that we exploit and the specified linking template queries. For instance, a loose and generic template query could return many irrelevant URIs, while a very "strict" template query could return no URIs. Note also that there might be more than one URIs that semantically are correct, e.g. two URIs coming from two different KBs may refer to the same real-world object. In any case it is useful to score and rank these URIs. One approach is the following: for each URI u that matches an entity name e, we can compare the string of e with the label of u (in a graph G_i) or/and the suffix of the URI string. Specifically, let $label(u)$ be the value of u's `rdfs:label` property in G_i and $suffix(u)$ be the substring of the URI string after the last '\' or '#', replacing the underscore letters that might exist with the space character. Let now $edt(a, b)$ be the Edit (Levenshtein) Distance [50] between the strings a and b (ignoring case). If $l(a)$ denotes the length of a string a, we can define the similarity between two strings a and b as:

$$sim(a, b) = \frac{\max(l(a), l(b)) - edt(a, b)}{\max(l(a), l(b))} \tag{3.3}$$

Then, the score of a URI u that matches an entity name e can be defined as:

$$URIscore(e, u) = \max(sim(e, label(u)), sim(e, suffix(u))) \tag{3.4}$$

Instead of comparing the strings using Edit Distance, we can use the distance function proposed by Stoilos et al. [59], where the similarity between two strings is a function of both their commonalities and their differences. In both cases, the highest the score of a URI is, the more probably that URI characterizes the corresponding entity name. We have chosen the property `rdfs:label` because it is the most common and widely used property for indicating the name/label of an entity. Of course, and according to the KB that we exploit, one could use another property, e.g. `foaf:name`, `skos:prefLabel`, etc. If a URI contains multiple values for this property, we can consider all of them and select the one with the highest similarity score. Moreover, if a URI does not contain a value for this property, we can consider $sim(e, label(u)) = 0$.

3.5 Effectiveness of the URI Ranking Approaches

In order to get a first feedback about the effectiveness of the URI ranking approaches, we performed an evaluation in the marine domain. The objective is to inspect if the proposed ranking schemes can detect the URI that best characterizes the corresponding entity name.

We ran indicative experiments using a collection of 464 Wikipedia pages regarding several fish species.[3] Specifically, we performed entity mining with fish species from DBpedia as the entities of interest, and for each top-ranked (i.e. more frequent) detected entity we retrieved its matched URIs by querying DBpedia's SPARQL endpoint and using the linking template query of Figure 3.4. Then, we computed the top-ranked URI using a) the Edit Distance function, b) the `Stoilos` function [59], and we manually inspected if that URI actually represents or not the corresponding entity name.

The number of entities for which we retrieved their matched URIs was 412 (some pages returned the same top-ranked entity). In average, about 10 matched URIs were returned for each entity name, while for 232 entity names the SPARQL query returned only one (correct) URI and thus we ignored them. Using the Edit Distance function, for the 91.1% of the remaining entity names the top-ranked URI was correct, while for the 2.2% the top-ranked URI was false. Moreover, for the 5.6% of entity names there were more than one URIs with the same top score, containing the correct URI, while for the 1.1% there were more than one URIs with the same top score, not containing however the correct one. Using the `Stoilos` function, for the 91.1% of the entity names the top-ranked URI was correct, for the 1.1% the top-ranked URI was false, for the 6.1% there were more than one URIs with the same top score, containing the correct URI, while for the 1.7% there were more than one URIs with the same top score, not containing however the correct one.

By inspecting the false cases, we noticed that the main cause is that the corresponding entity name is a disambiguated entity and its type/category has been added in parentheses in both its URI and its `rdfs:label`. For example, in DBpedia the label of the fish genus "Gila" is "Gila (genus)" and not "Gila". Thus, an optimization of this ranking method would be to remove any text in parentheses from the `rdfs:label` and the suffix of the URI.

From the above results, we can conclude that comparing the name of the detected entity with the label of the matched URI or the suffix of the URI string, we can find the correct matched resource with precision more than 90%. As regards the distance function, we saw that both Edit Distance and `Stoilos` behave well with almost the same performance.

3.6 The Open NEE Configuration Model

In [34] we proposed an RDF/S vocabulary called "Open NEE Configuration Model", which enables a NEE tool to describe (and publish as Linked Data) its configuration and its "entity mining" capabilities. As depicted in Figure 3.9, the proposed vocabulary[4] consists of 8 classes and 13 individual properties, which are briefly

[3]The dataset used in the evaluation as well as the results are available to download through: `http://www.ics.forth.gr/isl/X-Link/files/rankEval.zip`.

[4]The vocabulary is accessible through: `http://www.ics.forth.gr/isl/oncm`

described in Table 3.1.

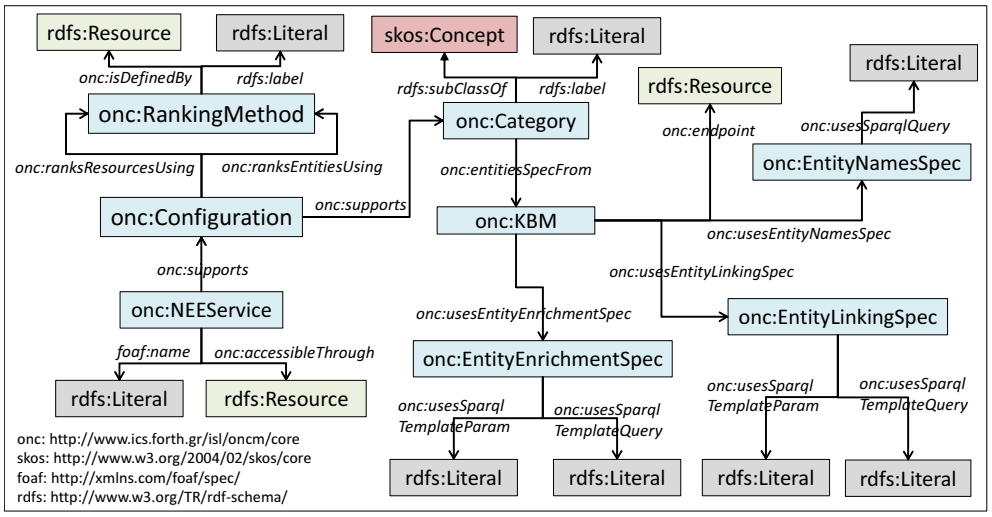

Figure 3.9: The Open NEE Configuration Model.

By publishing the configurations supported by one or more NEE services, an application can dynamically detect and use the services that satisfy its annotation needs, while we are able to run (SPARQL) queries of the form:

- Give me the NEE services supporting a category with name "Mammal" (Figure 3.10).
- Give me all categories that use a KBM with SPARQL endpoint `<http://dbpedia.org/sparql>` (Figure 3.11).
- Give me all KBMs (together with the endpoints) of category with name "Location" (Figure 3.12).

3.7 Exporting/Exchanging the Annotation Results

Correspondingly with Open NEE Configuration Model where we describe the configuration of a NEE tool in Linked Data, it would be useful if we could also export its annotation results in the same way. For this reason in [34] we propose an extension of the Open Annotation Data Model [58]. The extension model[5] (which is an RDF/S vocabulary) is depicted in Figure 3.13 and introduces 1 new class, 8 new properties and 1 new instance. This model enables the user to perform complex SPARQL queries (according to defined vocabulary) over the annotation results and retrieve the resources that meet the criteria. Also can be consider as a way of comparing the quality of results of various NEE tools.

[5]The extension is available at `http://www.ics.forth.gr/isl/oae`

Table 3.1: Classes and properties of the Open NEE Configuration Model.

Class	**Class description**
`NEEService`	A Named Entity Extraction (NEE) service.
`Configuration`	The configuration supported by a NEE service.
`Category`	A category/class of entities supported by a configuration.
`RankingMethod`	A method used for ranking the entities or the entity URIs.
`KBM`	A Knowledge Base Mirror: the gateway for accessing a Knowledge Base.
`EntityNamesSpec`	Specification of the entity names of a category.
`EntityLinkingSpec`	Specification of how an entity name corresponds to entity URIs.
`EntityEnrichmentSpec`	Specification of the extra information that should be fetched for an entity URI.
Property	**Property description**
`supports`	Relates a NEE service to a configuration, or a configuration to a supported category.
`accessibleThrough`	Relates a NEE service to a resource, e.g. to a URL describing the API of a service.
`ranksEntitiesUsing`	Relates a configuration to a method for ranking entities.
`ranksResourcesUsing`	Relates a configuration to a method for ranking resources.
`isDefinedBy`	Relates a ranking method to a resource, e.g. to a URL describing the ranking approach.
`entitiesSpecFrom`	Relates a category to a KBM.
`endpoint`	Relates a KBM to the URL of a SPARQL endpoint.
`usesEntityNamesSpec`	Relates a KBM to specification of entity names.
`usesEntityLinkingSpec`	Relates a KBM to an entity-linking specification.
`usesEntityEnrichmentSpec`	Relates a KBM to an entity-enrichment specification.
`usesSparqlQuery`	Relates a specification of entity names to a SPARQL query.
`usesSparqlTemplateQuery`	Relates an entity-linking or entity-enrichment specification to a SPARQL template query.
`usesSparqlTemplateParam`	Relates an entity-linking or entity-enrichment specification to a SPARQL template parameter.

```
SELECT ?tool ?name WHERE {
  ?tool a onc:NEEService ; foaf:name ?name ; onc:supports ?config .
  ?config onc:supports ?categ . ?categ rdfs:label "Mammal" }
```

Figure 3.10: SPARQL query for retrieving the name and the URL of all services that support the category "Mammal".

```
SELECT ?name WHERE {
 ?categ rdfs:label ?name ; onc:entitiesSpecFrom ?kbm.
 ?kbm onc:endpoint <http://dbpedia.org/sparql>}
```

Figure 3.11: SPARQL query for retrieving the categories that use a KBM with SPARQL endpoint *http://dbpedia.org/sparql*.

```
SELECT ?kmb ?endpoint WHERE {
  ?categ rdfs:label "Location"  ;  onc:entitiesSpecFrom ?kbm.
  ?kbm onc:endpoint ?endpoint}
```

Figure 3.12: SPARQL query for retrieving the KBMs (together with the endpoints) of category with name "Location"

By performing NEE in a set of documents and exporting the results using the proposed extension, we can run (SPARQL) queries of the form:

- Give me documents referring entities whose category is subclass of "Eukaryote" (Figure 3.14).
- Give me documents containing information about mammals of genus "Panthera" (Figure 3.15).
- Give me countries with population over 30 million (Figure 3.16).

An application can now offer advanced exploratory search services over the annotated set of documents, e.g. according to the faceted interaction paradigm over RDF data [37, 44].

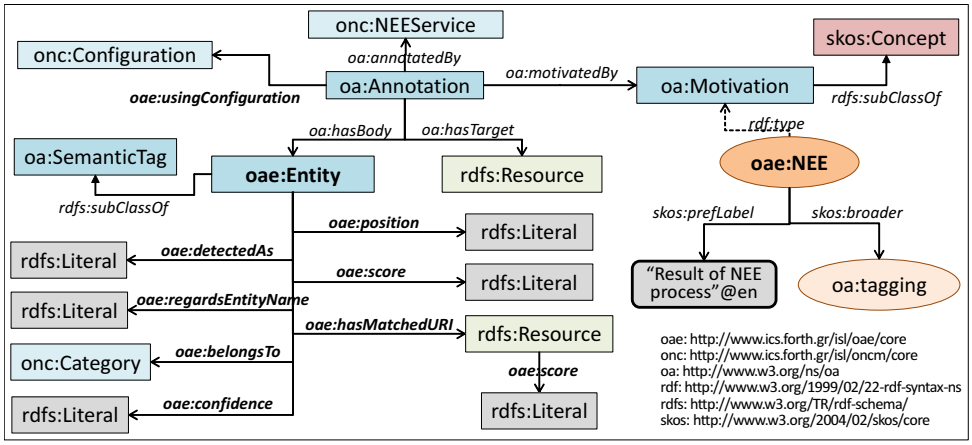

Figure 3.13: The extension of the Open Annotation Data Model.

```
SELECT ?doc WHERE {
  ?annot a oa:Annotation ; oa:hasTarget ?doc ; oa:hasBody ?ent .
  ?ent a oae:Entity ; oae:belongsTo ?cat .
  ?cat a onc:category ; rdfs:subClassOf ?superCat.
  ?superCat rdfs:label "Eukaryote" }
```

Figure 3.14: SPARQL query for retrieving the documents referring entities whose category is subclass of “Eukaryote”

```
SELECT ?doc WHERE {
  ?annot a oa:Annotation ; oa:hasTarget ?doc ; oa:hasBody ?ent .
  ?ent a oae:Entity ; oae:belongsTo ?cat ; oae:hasMatchedURI ?uri .
  ?cat a onc:category ; rdfs:label "Mammal" .
  ?uri dbp-owl:genus dbp:Panthera }
```

Figure 3.15: SPARQL query for retrieving the documents containing information about mammal of genus “Panthera”.

```
SELECT ?entName WHERE {
  ?ent a oae:Entity ; oae:belongsTo ?cat ; rdfs:label ?entNam .
  ?cat a onc:category ; rdfs:label "Country".
  ?ent dbo-owl:populationTotal ?pop. FILTER(?pop > 30000000)}
```

Figure 3.16: SPARQL query for retrieving countries with population over 30m.

Chapter 4

The X-Link Framework

X-Link is a LOD-based NEE framework that we have designed and implemented which realizes the configuration model described in the previous section. Below, we describe its architecture (§4.1), its functionality (§4.2), the supported configurability (§4.3), two applications that currently use X-Link (§4.4), and finally we evaluate its effectiveness (§4.5).

4.1 Architecture

X-Link is based on the Gate ANNIE [18, 25] system and supports both gazetteers and NLP functions. Gate ANNIE is a ready-made information extraction system which contains several components (e.g. Tokeniser, Gazetteer, Sentence Splitter, Orthographic Coreference, etc.). X-Link extends Gate ANNIE in order to be able to create a new supported category and update an existing one (using gazetteers). This gives us the opportunity to adapt its functionality according to our needs, making X-Link configurable and extendible. We should also stress that X-Link can use any NER system (as a component) that takes as input a text and returns a list of entity names.

Figure 4.1 shows the architecture of X-Link. The core component is the Controller which links and controls all the components. Configuration Manager is responsible for reading and changing the configuration files (Gate and X-Link Configuration Files). Entity Miner is an extension of Gate ANNIE and performs the entity mining process in the contents of a document (the document is read by the Text Extractor component). The components Entity Linker, Entity Enricher and Entity Connector are responsible for retrieving the corresponding semantic information by querying external SPARQL endpoints (using the SPARQL Query Runner component). Finally, the results are exported using the Result Exporter component.

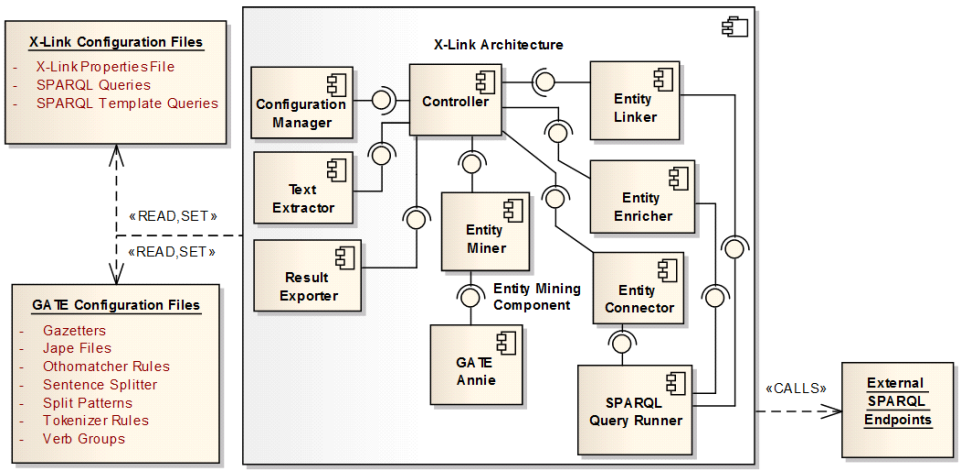

Figure 4.1: The architecture of `X-Link`.

4.2 Functionality

4.2.1 Supported File Types

Currently `X-Link` supports the analysis of plain text files, HTML pages, Microsoft Word and Powerpoint files (`.doc`, `.docx`, `.ppt` and `.pptx`), PDF files, and XML-based files (e.g. XML and RDF files).

4.2.2 Entity Mining

`X-Link` at first reads the contents of the requested document. Then it applies entity mining using Gate ANNIE according to the specified categories of interest. In our setting, Gate ANNIE takes as input the contents of a document and the categories of interest, and the output is a set of detected entities. Each detected entity is accompanied by its category, its position(s) in the document and its score. `X-Link` ranks the detected entities according to their frequency in the document as described in §3.4. Note also that Gate Annie internally "cleans" the document contents by removing useless text (like the HTML tags of a Web page).

The user is also able to activate or not a "*`fuzzy matching`*" function which enables the identification of an entity that does not match exactly with an entity in a category's gazetteer (using the Edit - Levenshtein - distance [50]). The allowed edit distance value depends on the length of the matched entity and expresses the percentage of the required single-character edits with regard to the entity name's length. If p denotes the allowed percentage of single-character edits and $l(e)$ is the length of an entity e, then the allowed edit distance value (for which the candidate string will match the entity e) is $p * l(e)$. For instance, if $p = 0.2$ then we allow 2 edits for an entity name with length 10 characters. The above functionality is

offered by the `BWPGazzetteer` plugin [51] of `Gate ANNIE`.

Furthermore, it is quite common in many documents that named entities are mentioned with their full length only in their first occurrence(s), while in the rest of them are partially mentioned as substring(s) or acronyms of their gazetteer entry. For instance, consider the following textual snippets:

p_1 : City win the League.

p_2 : Putin asks to talk with Tsipras immediately.

p_3 : Raikkonen takes no pleasure from beating Vettel.

p_4 : IMF might have lost its policy cloud over Greece.

p_5 : The ISWC is coming to the Lehigh Valley this fall.

p_6 : Papadopoulos and Kresic are recently transferred in Leverkusen.

The desired output of a NER tool (e.g. `X-Link`) is the following:

- In snippet p_1 the entities are *City* and *League*, whose real complete labels are *Manchester City* and *Premier League*.
- In snippet p_2 the entities are *Putin* and *Tsipras*, whose real complete labels are *Vladimir Putin* and *Alexis Tsipras*.
- In snippet p_3 the entities are *Raikkonen* and *Vettel*, whose real complete labels are *Kimi Raikkonen* and *Sebastian Vettel*.
- In snippet p_4 the entities are *IMF* and *Greece*, where the real complete label of *IMF* is *International Monetary Fund* .
- In snippet p_5 the entities are *ISWC* and *Lehigh Valley*, where the real complete label of *ISWC* is *International Semantic Web Conference*.
- Finally, in snippet p_6 the entities are *Papadopoulos*, *Kresic* and *Leverkusen*, whose real complete labels are *Kyriakos Papadopoulos*, *Dario Kresic* and *Bayer Leverkusen*.

From now on, we consider as complete label of a partial entity matching, the label of the named entity as is stored to a gazetteer list and corresponds to this matching. Notice that a partial matching may refer to more than one complete labels, which are deemed to be equiprobable and we keep them all (see Figures 4.9 & 4.10). In both snippets, it is almost impossible for a NER tool to identify most of these mentions as entities. Hence, except from *"fuzzy"*, the user can use a "***partial matching***" function which enables the identification of partial matchings, permutations and acronyms of a complete entity label in a category's gazetteer. To succeed this we extend the previous plugin, and we store the changes into a new `Gazetteer` component named `PartialMatchingGazetteer`, as described briefly in Algorithm 1, and Figure 4.2. In particular, we modify the function that annotates

named entities occurrences in a document, in which we split a complete entity label into substrings/tokens, using as separator the space character, and we concern them as separate entities (Figure 4.5, lines 66-80). Additionally, in the case that an entity name consist of 3 or more terms we create acronyms, using their first characters (Figure 4.5 and 4.6, lines 22-65). We also enable the identification of subterm permutations of a named entity label (Figure 4.6, line 82 and Figure 4.7), i.e. *Ioannis Kapodistrias*, *Kapodistrias Ioannis*, etc.

An alternative approach is to extend the function that adds a new category of entities and create an additional gazetteer file for keeping the partial matchings, permutations and acronyms of new entities, as shown in Figure 4.3. The partial matchings, the permutations and the acronyms are retrieved in a similar way as above. Additional, in order to be aware of the provenance of partial matchings, we keep in an external file the complete entity label(s) that correspond to them. Notice that the above functionality is also applied when we replace or update an existing category with new entities.

To make it clear consider a gazetteer list that includes several organizations, where one of them is the *"International Monetary Fund"*. Its partial matchings include the terms *"fund"*, *"international"*, *"international monetary"*, *"international monetary fund"*, *"monetary"* and *"monetary fund"*, whereas its possible acronyms are *"i.m.f."*, *"i.m.f"*, *"imf"* and *"i m f"*. Notice that the *"fuzzy matching"* functionality is also available for partial matchings[1]. Nevertheless, many of these substring matchings are useless, in the sense that they do not offer us any information if they are split, e.g. the terms *"Fédération Internationale de Football Association"*, *"Fédération Internationale de l'Automobile"* or more commonly referred as *"FIFA"* and *"FIA"*, are almost never seen in their full length, and none of their partial matchings is semantically equal with their complete terms.

Depending on adopted approach, there is an extra cost in execution time and storage. If we use the `PartialMatchingGazetteer` it takes longer to identify named entities in document, because we must always create their partial matchings. On the contrary, the 2nd approach requires more space for the additional gazetteer files where we keep the entity partial matchings of the supported categories. It also requires more time to load the gazetteer files in main memory during the initialization of `X-Link`, which is an extra cost that we have to pay only when we start or restart it. Thus, the basic difference between these approaches is that the 1st is applied at runtime, whereas the 2nd only in the initialization step, which is better in our setting.

However, although in these cases more entity names are identified, the precision falls off because `X-Link` may identify entities that are lexicographically close to an entity in a category but semantically totally different. This problem is more acute when partial matching is enabled, as perceived by the following textual snippets:

p_6 : The car that involved in the hit-and-run accident was white.

[1]From now on, we consider that partial matchings include also acronyms and permutations.

p_7 : The average cost per mile for a small sedan is 46.4 cents.

p_8 : Pope Francis would make a two-day visit to the city of Valleta.

Suppose that our configuration supports gazetteer lists about films, politicians, countries and football teams. Consequently, the retrieved entities will be, *"car"*, *"white"* (p_6), and *"mile"* (p_7), which are identified as acronym of *Central African Republic* and as partial matchings of Australian politician *David White* and film *The Green Mile* respectively. Finally, in p_8 term *"city"* is considered as partial matching of football team *Manchester City* and country *Vatican City*. Nevertheless, in both snippets none of these terms correspond semantically to the retrieved entities. A possible solution is to keep only the partial matchings whose complete label appears as entity in the list of identified entities. This heuristic can be applied a posteriori, after NEE process. An alternative solution is to keep only the partial matchings whose string similarity with their complete label is over a threshold, which is applied during the creation of a new category. Although both solutions are naive, they seem to work well in our case.

Algorithm 1 Entity Name Partial Matching Annotation

Input: *text*, the text where the search should be performed,
annotationSet, the current set of annotations associated with the text,
entity, the word that has to be found,
lookup, the type of annotation that will be associated with occurrences of the word in the text

```
1: labels ← ∅
2: entity_nostw ← removeStopwords(entity)
3: if entity_nostw.length() > 0 then
4:    acronyms ← getPossibleAcronyms(entity_nostw)
5:    partialMatchings ← getPossiblePartialMatchings(entity_nostw)
6:    labels.add(acronyms)
7:    labels.add(partialMatchings)
8:    for all label ∈ labels do
9:       annotateOccurrences(annotationSet, lookup, label, entity, text)
```

4.2.3 Entity Linking

As regards the *entity linking* process, `X-Link` returns a *ranked* list of URIs that match a detected entity name and lets the application (that uses `X-Link`) to decide how to cope with them. For instance, in the application example of Figure 1.1

```
/** Does the actual search for a given word in the text and annotates the occurrences found.
 *
 * @param text the text where the search should be performed
 * @param annotationSet the current set of annotations associated with the text
 * @param entity the word that has to be found
 * @param lookup the type of annotation that will be associated with occurrences of the
 *               word in the text
 */
  private void annotate(char[] text, AnnotationSet annotationSet, String entity,
     Lookup lookup){

1       for(String label : findAcronymsPartialsPermutations(entity).keySet()){
            // for each 'label' we annotate its occurrences in the given text
2           annotateOccurrences(annotationSet, lookup, label, entity, text);
3   }   }
```

Figure 4.2: Annotation function of *partial matching* plugin.

the system presents to the user all the URIs that match an identified entity, while another application could return only the top-ranked URI.

For ranking the URIs, `X-Link` supports the approach described in §3.4 which computes the similarity between the name of the entity and the label of the URI in the KB or the suffix of the URI string. As regards the distance function, `X-Link` supports both Edit Distance and the one proposed by Stoilos et al. [59]

4.2.4 Entity Enrichment

As regards *entity enrichment*, i.e. the retrieval of RDF triples that describe the entity URIs, `X-Link` offers two different functions: a) retrieve triples that are interesting for the application at hand, and b) inspect the connectivity of the entity URIs.

As regards the former, for an entity URI u, $Descr(u)$ is obtained either by running the corresponding Enrichment Template Queries or by selecting to retrieve one of the following (common) types of properties: a) outgoing (u is the `subject` in the RDF triple), b) incoming (u is the `object` in the RDF triple), c) both outgoing and incoming, d) outgoing in a specific language, e) both outgoing in a specific language and incoming. Note that it is in the responsibility of the application that uses `X-Link` to decide how to exploit all this semantic information. For instance, one can use it even for disambiguating the identified entities, or for ranking the URIs, etc.

As regards the connectivity of the entity URIs, `X-Link` supports $Graph(D, r)$ as defined in §3.4. In addition, it computes a subgraph of $Graph(D, r)$, which is denoted by $ConnectGraph(D, r)$, for making more evident how the entity URIs are associated. Specifically, this graph contains only the triples which are involved in paths whose both start and end vertex are URIs in $U(D)$. For example, for $r = 1$ the graph can show entity URIs that share common properties or which are directly connected (so properties that are not reachable by at least two URIs

are omitted). Figure 4.8 depicts an example of a $Graph(D, 1)$. Consider that the entity URIs in $U(D)$ are three: *Chum Salmon*, *Chinook Salmon* and *Coho Salmon* (for simplicity we have omitted the namespaces). The graph enclosed in the dashed shape, containing the black nodes, is the $ConnectGraph(D, 1)$.

4.2.5 Entity Disambiguation

In §3.1 we argued that we may have a name that corresponds to entities of different categories (recall the *"argentina"* example which may refer to the Country Argentina or the Fish Genus Argentina). Even if only one category is supported, we cannot be sure if a detected entity (that matches an entity name in the gazetteer of the supported category) actually belongs to this category. Likewise, when ***partial matching*** is used, it is highly possible that a document term will be incorrectly identified as entity or may correspond to more than one entity names of different (or same) categories. Recall the textual snippets in §4.2.2, whose extracted entities are depicted in Figures 4.9 & 4.10. This is the well-known entity disambiguation (or *word-sense disambiguation*) problem whose solution stills an open challenge [19]. Several approaches have been proposed in the literature, e.g. exploiting Wikipedia data [24, 40], using statistical methods [1], exploiting ontologies [41], or graph-based approaches [49, 61]. `X-Link` currently does not apply any disambiguation method, i.e. if an entity name exists in the gazetteers of two supported categories, then this entity is returned twice, one for each supported category. However, in Chapter 5 we propose some entity disambiguation methods that allows the application that uses them to disambiguate afterwards the identified entities, e.g. by exploiting context information or user feedback. For instance, in `Theophrastus` system [35] if the user requests the exploration of a detected entity with ambiguous name (i.e. which belongs to more than one of the supported categories), the system informs the user through a popup window and the user can disambiguate the entity by selecting the appropriate category (the `Theophrastus` system is briefly described in §4.4). In our setting, exploiting the RDF triples that correspond to the detected entities, i.e. $Graph(doc)$, can help towards this direction. For example, by adopting a Link Analysis-based approach [28, 32] for ranking the elements (entities and properties) of a graph related to a set of search results, we could isolate entities irrelevant to the search context (they will receive low score).

Note also that in some application scenarios, especially in *professional systems*, even if we are not sure about the relevance of an entity, it is preferable to retrieve and return it, i.e. recall (the retrieval of as much as possible relevant information) is crucial. For instance, in professional search (e.g. medical search, patent search, bibliography search) it is often unacceptable to miss relevant documents, therefore the retrieval of nearly all relevant documents is sometimes necessary.

4.2.6 Overall Process

Summarizing the above functionality, the following steps are:

1. We instantiate the entity mining component based on given configuration.
2. We select whether to enable or not the fuzzy and partial matching functionality. *(optional)*
3. We instantiate `X-Link` using the entity mining component.
4. We apply entity mining in the given document.
5. We select whether to link and enrich the retrieved entities with semantic resources. *(optional)*
6. We select whether to disambiguate the retrieved entities. *(optional)*

The beyond steps are also depicted in Algorithm 2.

Algorithm 2 Named Entity Extraction and Disambiguation process

Input: *text*, the text where the search should be performed,
categories, the set of active categories,
configuration, the file path of the configuration file

1: $emc = new\ GateEntityMiningComponent(config)$ // instantiation of *GateEntityMiningComponent*
2: $emc.startup()$
3: $emc.enablePartialMatching()$ //enable partial matching functionality *(optional)*
4: $emc.enableFuzzyMatching()$ // enable fuzzy matching functionality *(optional)*
5: $emc.changeEditDistanceValue(0.2)$ //set the maximum allowed percentage of (lexicographical) distance between 2 terms to 0.2
6: $xlink = new\ \ XLink()$
7: $xlink.setEntityMiningComponent(emc)$ //instantiation of `X-Link`
8: $xlink.retrieveEntities(text, categories)$ //apply entity mining
9: $xlink.xlink()$ //entity linking functionality *(optional)*
10: $xlink.enrich()$ //entity enrichment functionality *(optional)*
11: $xlink.disambiguate()$ //entity disambiguation functionality *(optional)*
12: $entities = xlink.getEntities()$

4.2.7 Output

`X-Link` exports the results in XML, CSV and RDF. As regards RDF, `X-Link` exploits the extension of the Open Annotation Data Model (described in §3.7) and supports the formats RDF/XML, N-Triples, and Notation3 (N3).

4.2.8 Ways to Use

`X-Link` is a framework that can be used (and extended) by other applications according to their needs, allowing its exploitation in a plethora of contexts and application scenarios. Specifically, `X-Link` can be used as a:

- **Java Library** which can be integrated in the code of the intended application.
- **Web Application** that can receive submissions and return the outcomes of the analysis.
- **Web Service** which can be used through a REST API.

In the last two cases, it is assumed that a running instance exists, therefore the `X-Link` library offers operations that allow *changing* the configuration model. This allows changing or refreshing the "knowledge" of `X-Link` without having to redeploy the application that uses it.

More information is available at: `http://www.ics.forth.gr/isl/X-Link`.

4.3 Configurability

`X-Link` supports the configuration model described in §3 in two ways: (a) it can read such a configuration from a *properties file*, and (b) it offers a configuration API. It can also read a configuration expressed in RDF using the Open NEE Configuration Model. For publishing the configuration supported by an `X-Link` service, `X-Link` offers a function which creates an RDF file describing its current configuration using the Open NEE Configuration Model. For instance, the configuration that is currently supported by an `X-Link` service configured for the *marine* domain is publicly available at `http://www.ics.forth.gr/isl/X-Link/marine/config.n3`.

4.3.1 File-based Configuration

An indicative part of the properties file (configured for the *marine* domain) is shown in Figure 4.11. In that example, `X-Link` supports 7 categories of entities (line 1), i.e. the entity names of these categories have been retrieved and stored in Gate ANNIE. However, the *active categories* are only Fish, Country and Water Area (line 2), i.e. the remaining categories are inactive. The set of *active categories* allows us to define which of the supported categories are interesting for an application, thus `X-Link` can identify entities that belong to these categories only. The category Fish uses one KBM (line 3), which is actually the SPARQL endpoint of DBpedia (line 4), and for updating this category `X-Link` can use the SPARQL query given in a file (line 5). In addition, we can see the file paths and the parameters of the template queries that are used for linking and enriching the identified fishes (lines 6-9). Finally, the radius for inspecting the connectivity of the identified entities is 1 (line 10), while *fuzzy & partial* matching is allowed with $p = 0.2$ (lines 11-12).

4.3.2 Configuration while Running

`X-Link` can be configured through its API even while a corresponding service is running. In particular, the following functions are supported:

- Add a new category (using one or more lists of entities and/or one or more SPARQL queries).
- Update an existing category (using one or more lists of entities and/or one or more SPARQL queries).
- Remove a category.
- Change the displayed name of a category (i.e. rename).
- Set/change the KBMs of a category.
- Set/change the SPARQL queries and the template queries of a KBM.
- Set/change the *active* categories.
- Set/change the value of radius r.
- Set/change if *fuzzy matching* is allowed and the value of p.
- Set/change if *partial matching* is allowed and the value of p.
- Set/change the URI ranking distance function.

Regarding the update of an existing category, the user/developer is able to either totally *replace* a category (i.e. remove its old entity names and add the new ones) or just add the new entity names. We should also note that each of the above functions changes accordingly the properties file and also it updates several files in Gate ANNIE. For example, when a new category is created, the corresponding gazetteer file is created and loaded in Gate ANNIE, the name of the category is added in the set of supported categories in the properties file, etc.

4.3.3 Portability of Configurations

The configurations can be exchanged. For instance, consider that a person A configures the system and then sends the configuration files to a person B. The person B sets the system to use the configurations files received by the person A (by simply providing some paths). Now the person B is able to enjoy exactly the same configuration as person A.

The size of the configuration files is relatively small and mainly depends on the number of supported categories and on the number of named entities in each category. Indicatively, the configuration files for supporting 4 categories related to the marine domain have size less than 5MB. These files include the gazetteers of the supported categories and several files required by Gate ANNIE. Note that `X-Link` does not store any semantic information (e.g. URIs or RDF triples), since the entity linking and the entity enrichment processes are performed at real-time.

We should also stress that adopting the Open NEE Configuration Model simplifies even more the exchange of configurations since an RDF file (describing the configuration using the proposed vocabulary) can be just provided.

4.4 Current Applications of X-Link

X-Link is currently used by the systems X-Search[2] and Theophrastus[3].

X-Search [27, 31] is a meta-search engine that reads the description of a search source, queries that source, analyzes the returned results in various ways and also exploits the availability of semantic repositories. X-Search exploits X-Link in two different contexts: in the *marine* domain (in the context of the iMarine[4] project) and in *patent search* (in the context of the PerFedPat[5] project). In iMarine, X-Link has been configured to identify Fish Species, Water Areas, Countries, and Regional Fisheries Bodies, while the KB that is exploited is the MarineTLO-based Warehouse[60]. In PerFedPat, X-Link has been configured to identify the (medicine-related) categories Diseases, Drugs, Proteins, and Chemical Substances, while the online version of DBpedia is exploited as the underlying KB [29].

Theophrastus [35] is a system that supports the automatic annotation of web documents through entity mining and provides exploration services by exploiting LOD at real-time. The system, which aims at assisting biologists in their research about species and biodiversity, exploits X-Link for performing entity mining and entity exploration in web documents, and has been designed to be highly configurable regarding a number of different aspects like entities of interest, information cards (semantic information related to a detected entity) and external search systems.

4.5 Effectiveness of X-Link

There are various papers that aim at evaluating the effectiveness of NEE tools [38, 53, 54, 55]. The effectiveness of X-Link highly depends on how the user/developer has configured it, i.e. on the completeness of the specified categories, the quality of the underlying KBs, the specified SPARQL template queries, etc. In X-Link we have focused on the *configurability* of a NEE system and on how we can exploit the LOD; we have not proposed a new entity mining method, since it currently relies on Gate ANNIE (the proposed approach can be also applied by existing NEE systems). Therefore, the quality of the identified entities is out of the scope of this thesis. Finally, as we already mentioned above X-Link does not apply any disambiguation method at run-time, however in §5 we propose a set of methods for disambiguating afterwards the identified entities, that are appropriate for our setting, and then we evaluate them in §6.

[2] http://www.ics.forth.gr/isl/X-Search
[3] http://www.ics.forth.gr/isl/Theophrastus
[4] http://www.i-marine.eu/
[5] http://www.perfedpat.eu/

```
/**
 * Adds the list file of the new category.
 *
 * @throws IOException
 */
  private void addListFile() throws IOException {

1    BufferedWriter output = new BufferedWriter(new OutputStreamWriter(
                             new FileOutputStream(LIST_FILE_NAME), "utf8")),
2              output_refs = new BufferedWriter(new OutputStreamWriter(
                             new FileOutputStream(LIST_FILE_NAME_PARTIAL_REFS), "utf8"));
3    for(String entity : entities) {
4        entity = entity.replace("\"", "");
5        output.write(entity);
6        output.write("\n");
7    }

8    output.close();
9    System.out.println("# The " + LIST_FILE_NAME + " file was created!");

10   output = new BufferedWriter(new OutputStreamWriter(
                                new FileOutputStream(LIST_FILE_NAME_PARTIAL), "utf8"));
11   TreeMap<String, TreeSet<String>> labels = findAcronymsPartialsPermutations(entities);

12    for (String entity : labels.keySet()) {
13       TreeSet<String> getLabels = labels.get(entity);

14       if(IsStringSimilarityOverThreshold(entity, getLabels, 0.5)){
15           output.write(entity);
16           output.write("\n");

17           String refs = "";
18           for(String temp : getLabels)
19               refs += temp + ";";

20           refs = refs.substring(0, refs.length()-1);

21           output_refs.write(entity + " -:- " + refs);
22           output_refs.write("\n");
23   }   }

24   output.close();
25   System.out.println("# The " + LIST_FILE_NAME_PARTIAL + " file was created!");

26   output_refs.close();
27   System.out.println("# The " + LIST_FILE_NAME_PARTIAL_REFS + " file was created!");
28 }
```

Figure 4.3: Add named entities in gazetteer file.

```
/**
  * Checks if the string similarity of a term with at least one (string) element of a set
  * is over a given threshold and returns true if holds, false otherwise.
  * @param partialMatchLabel A term.
  * @param realLabels A set of string elements.
  * @param threshold The threshold we want to check.
  * @return true if holds, false otherwise.
  */
    public static boolean IsStringSimilarityOverThreshold(String partialMatchLabel,
                          Set<String> realLabels, double threshold){
1      for(String realLabel : realLabels){
2         if(XLink.computeStoilosSimilarity(realLabel.toLowerCase(),
                                          partialMatchLabel.toLowerCase()) >= threshold)
3               return true;
4      }
5      return false;
6   }
```

Figure 4.4: Checks if the string similarity of a term with at least one set element is over a threshold.

```
/**
 * Returns partial matchings, acronyms and permutations for a set of given named entities.
 * @param entities The set of named entities whose partial matchings and
 *                 acronyms we want to find.
 * @return A set of partial matchings and acronyms.
 */
  public static TreeMap<String, TreeSet<String>> findAcronymsPartialsPermutations
                                                  (TreeSet<String> entities){
1    Set<String> stopWords = new LinkedHashSet<String>();
2    TreeMap<String, TreeSet<String>> labels = new TreeMap<String, TreeSet<String>>();
3    try {
4        BufferedReader bf = new BufferedReader(new FileReader(
                               Resources.XLinkRepo_forTestClients + "stopwordsEn.txt"));
5        for(String line;(line = bf.readLine()) != null;)
6              stopWords.add(line.trim());
7        bf.close();
8    } catch (IOException ex) {
9        Logger.getLogger(HelpingFunctions.class.getName()).log(Level.SEVERE, null, ex);
10   }

11   for(String entity : entities){
12       String entity_nostw = "";
13       String[] split = entity.split(" ");

        // Remove stopwords from a named entity label
        // e.g.  entity = "United States of America"
        //       entity_nostw =  "United States America"
14       for (int i = 0; i < split.length; i++) {
15           if(!stopWords.contains(split[i])){
16               if(split[i].trim().length() > 0)
17                   entity_nostw += split[i] + " ";
18       }   }

19       if(entity_nostw.trim().length() > 0 ){
20           String[] split2 = entity_nostw.trim().split(" ");
21           String label = "";

        //  Acronyms
22       String temp = "", temp_dot = "", temp_sp = "";

23       if(split2.length >=3){
24           for (int i = 0; i < split2.length; i++) {
25               char tmp = split2[i].charAt(0);
26               temp = temp + tmp;                 // USA
27               temp_dot = temp_dot + tmp + ".";   // U.S.A.
28               temp_sp = temp_sp + tmp + " ";     // U S A
29           }

30           TreeSet<String> get = labels.get(temp.toLowerCase().trim());
31           if(get == null){
32               get = new TreeSet<String>();
33               get.add(entity);
34               labels.put(temp.toLowerCase().trim(), get);
35           }   else{
36               get.add(entity);
37               labels.put(temp.toLowerCase().trim(), get);
38           }

39           get = labels.get(temp_dot.toLowerCase().trim());
40           if(get == null){
41               get = new TreeSet<String>();
42               get.add(entity);
43               labels.put(temp_dot.toLowerCase().trim(), get);
44           }   else{
45               get.add(entity);
46               labels.put(temp_dot.toLowerCase().trim(), get);
47           }
            ...
```

Figure 4.5: Find partial matchings, acronyms and permutations.

```
    ...
48   get = labels.get(temp_sp.trim().toLowerCase().trim());
49   if(get == null){
50       get = new TreeSet<String>();
51       get.add(entity);
52       labels.put(temp_sp.trim().toLowerCase().trim(),get);
53   }   else{
54       get.add(entity);
55       labels.put(temp_sp.trim().toLowerCase().trim(),get);
56   }

57   get = labels.get(temp_dot.substring(0, temp_dot.length() - 1).toLowerCase().trim());
58   if(get == null){
59      get = new TreeSet<String>();
60      get.add(entity);
61      labels.put(temp_dot.substring(0, temp_dot.length() - 1).toLowerCase().trim(),get);
62   }  else{
63      get.add(entity);
64      labels.put(temp_dot.substring(0, temp_dot.length() - 1).toLowerCase().trim(),get);
65   }

    // e.g.  labels = {"united states of america", "usa", "u.s.a.", "u s a", "u.s.a"}

    // Partial matchings
66   for(int i = split2.length; i>=1; i--){
67       for(int j = 0; j <= split2.length - i; j++){
68           for(int k = j; k < j + i; k++)
69               label+=split2[k] + " ";

70           get = labels.get(label.trim().toLowerCase().trim());
71           if(get == null){
72               get = new TreeSet<String>();
73               get.add(entity);
74               labels.put(label.trim().toLowerCase(), get);
75           }   else{
76               get.add(entity);
77               labels.put(label.trim().toLowerCase(), get);
78           }
79           label = "";
80    }   }

     // e.g.  labels = {"united states of america", "usa", "u.s.a.", "u s a", "u.s.a",
     //                 "america", "states", "states america", "united",
     //                 "united states", "united states america"}

     //  Permutations
81    String[] split_p = entity_nostw.split(" ");
82    TreeSet<String> permutations = permute(Arrays.asList(split_p),0);

83    for(String permutation : permutations){
84       TreeSet<String> get = labels.get(permutation.toLowerCase().trim());
85       if(get == null){
86           get = new TreeSet<String>();
87           get.add(entity);
88           labels.put(permutation.toLowerCase().trim(), get);
89       }   else{
90           get.add(entity);
91           labels.put(permutation.toLowerCase().trim(), get);
92    }  }

     // e.g. entity = 'barack obama'
     //      permutations = {'barack obama', 'obama barack'}

93   }   }
94   return labels;
95 }
```

Figure 4.6: Find partial matchings, acronyms and permutations (cont.).

```
/**
 * Find all combinations of a given word terms.
 * @param arr An array that contains the terms of a word.
 * @param k The starting position of array.
 * @return
 */
  public static TreeSet<String> permute(List<String> arr, int k){

1    TreeSet<String> labels = new TreeSet<String>();
2    for(int i = k; i < arr.size(); i++){
3        java.util.Collections.swap(arr, i, k);
4        labels.addAll(permute(arr, k+1));
5        java.util.Collections.swap(arr, k, i);
6    }

7    if (k == arr.size() -1){
8        String label = "";
9        String[] array = (String[]) arr.toArray();
10       for(String str : array)
11           label+=str + " ";
12       labels.add(label);
13   }

14   return labels;
15  }
```

Figure 4.7: Find String permutations of given elements.

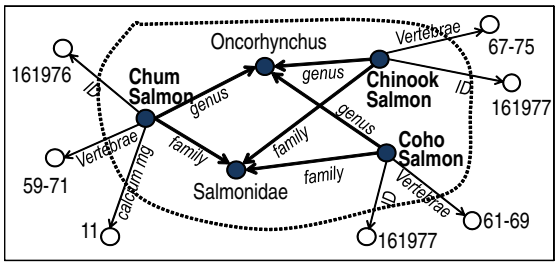

Figure 4.8: An example of a *ConnectGraph*.

```
- Input: "City win the League."
- Output:

# Detected entities:
Entity name: City
Category: country
Complete label: [vatican city]

Entity name: City
Category: football_teams
Complete label: [manchester city]

Entity name: League
Category: sports_league
Complete label: [premier league]


- Input: "Putin asks to talk with Tsipras immediately."
- Output:

# Detected entities:
Entity name: Tsipras
Category: politicians
Complete label: [alexis tsipras]

Entity name: Putin
Category: politicians
Complete label: [vladimir putin]


- Input: "Raikkonen takes no pleasure from beating Vettel."
- Output:

# Detected entities:
Entity name: Raikkonen
Category: formula_1_drivers
Complete label: [kimi raikkonen]

Entity name: Vettel
Category: formula_1_drivers
Complete label: [sebastian vettel]


- Input: "IMF might have lost its policy cloud over Greece."
- Output:

# Detected entities:
Entity name: Greece
Category: country
Complete label: [greece]

Entity name: IMF
Category: organizations
Complete label: [international monetary fund]


- Input: "The ISWC is coming to the Lehigh Valley this fall."
- Output:

# Detected entities:
Entity name: ISWC
Category: conferences
Complete label: [international semantic web conference,
                 international symposium on wearable computers]

Entity name: Lehigh Valley
Category: location
Complete label: [lehigh valley]
```

Figure 4.9: Output of X-Link using partial matching.

```
- Input: "Papadopoulos and Kresic are recently transferred in Leverkusen."
- Output:

# Detected entities:
Entity name: Papadopoulos
Category: football_players
Complete label: [avraam papadopoulos, dimitris papadopoulos, kyriakos papadopoulos]

Entity name: Kresic
Category: football_players
Complete label: [dario kresic, milos kresic]

Entity name: Leverkusen
Category: football_teams
Complete label: [bayer leverkusen]

Entity name: Papadopoulos
Category: politicians
Complete label: [tassos papadopoulos]


- Input: "The car that involved in the hit-and-run accident was white."
- Output:

# Detected entities:
Entity name: car
Category: country
Complete label: [central african republic]

Entity name: white
Category: politicians
Complete label: [david white]


- Input: "The average cost per mile for a small sedan is 46.4 cents."
- Output:

# Detected entities:
Entity name: mile
Category: film
Complete label: [the green mile]


- Input: "Pope Francis would make a two-day visit to the city of Valleta."
- Output:

Entity name: city
Category: country
Complete label: [vatican city]

Entity name: city
Category: football_teams
Complete label: [manchester city]
```

Figure 4.10: Output of X-Link using partial matching (cont.).

```
1   xlink.categories.supported = Fish;Country;Water_Area;Disease;Drug;Protein;Chemical_Substance
2   xlink.categories.active = Fish;Country;Water_Area
3   xlink.categories.Fish.kbms = dbpedia_fish
4   xlink.categories.Fish.kbms.dbpedia_fish.endpoint = http://dbpedia.org/sparql
5   xlink.categories.Fish.kbms.dbpedia_fish.entitynames = C:/xlink/queries/dbp_fishes.sparql;
6   xlink.categories.Fish.kbms.dbpedia_fish.templatequeries.linking =
                                          C:/xlink/templates/dbp_fish_linking.tquery
7   xlink.categories.Fish.kbms.dbp_fish.templatequeries.linking.parameter = [ENTITY]
8   xlink.categories.Fish.kbms.dbp_fish.templatequeries.enriching =
                                          C:/xlink/templates/dbpedia_fish_enriching.tquery
9   xlink.categories.Fish.kbms.dbpedia_fish.templatequeries.enriching.parameter = [URI]
10  xlink.connect.radius = 1
11  xlink.fuzzy = true
12  xlink.fuzzy.value = 0.2
13  xlink.partialmatching = true
```

Figure 4.11: A part of `X-Link`'s properties file configured for the marine domain.

Chapter 5

Named Entity Disambiguation

In this chapter we focus on `NED` problem. Initially, in Section 5.1 we define the problem statement. In Section 5.2 we give some motivating examples. In Section 5.3 we present our research directions. In Section 5.4 we propose 3 disambiguation methods, and then we evaluate them over motivating examples.

5.1 Problem Statement

Let us assume that there is an entity name e that belongs to the set of entities of more than one categories, and we want to find the most probable category or categories of e, denoted as C_{prob}, where $C_{prob} \subseteq ctg(e)$. Consider a document doc that contains an entity e, such that $e \in Ent(doc)$ and there are two (or more) categories, say c and c', such that $e \in E(c) \cap E(c')$.

Let $occur_i(e, doc)$ denote the `i-th` occurrence of entity e in document doc. For an entity occurrence and a category c we shall use the notation $belongs(occur_i(e, doc), c)$ to define that the real category (what the author of the text means) of that entity is c. Also, let $occurCount(e, doc)$ be the number of occurrences of e in doc. If $occurCount(e, doc) = 0$ it means that entity e does not occur in document doc. We would like to estimate one of the following:

- Find the probability that the $occur_i(e, doc)$ corresponds to category c, i.e. compute $P(belongs(occur_i(e, doc), c)), \quad \forall c \in ctg(e)$
- Find the category $c \in ctg(e)$, denoted as $bestCtg(occur_i(e, doc))$, that maximizes the above probability, i.e. $c = argmax_{c \in ctg(e)} P(belongs(occur_i(e, doc), c))$.

5.2 Motivating Examples

Consider the following 5 textual snippets, and suppose that we are asked to retrieve entities that belong to the following categories, `Fish Species`, `Location`, `Food` and `Person`.

p_1 : Argentina and Salmon live in deep waters.

p_2 : Argentina is sharing land borders with Chile.

p_3 : Sebastian Leto was born in Argentina.

p_4 : I like sushi pizza with Argentina. Usually I eat them during summer time. By the way, the next summer I plan to visit Argentina because I have a cousin who studies in the university of Buenos Aires and he could host me there.

p_5 : The Argentina is 3 years of light away from earth.

The expected output of a NEE tool (e.g. `X-Link`) is the following:

- In snippet p_1 the retrieved entities are *Argentina* and *Salmon*.
  - *ctg* (*Argentina*) = {Location, Fish Species}
  - *ctg* (*Salmon*) = {Fish Species }
- In snippet p_2 the retrieved entities are *Argentina* and *Chile*.
  - *ctg* (*Argentina*) = {Location, Fish Species}
  - *ctg* (*Chile*) = {Location}
- In snippet p_3 the retrieved entities are *Argentina* and *Sebastian Leto*.
  - *ctg* (*Argentina*) = {Location, Fish Species}
  - *ctg* (*Sebastian Leto*) = {Person}
- In snippet p_4 the retrieved entities are *Argentina*, *sushi pizza* and *Patagonia*.
  - *ctg* (*Argentina*) = {Location, Fish Species}
  - *ctg* (*Sushi pizza*) = {Food}
  - *ctg* (*Buenos Aires*) = {Location}
- Finally, in snippet p_5 the only retrieved entity is *Argentina*
  - *ctg* (*Argentina*) = {Location, Fish Species}

In all snippets the ambiguous entity is the *Argentina*. A rising question is "How can we understand when *Argentina* refers to the Fish Species Argentina, the Location Argentina or something else?".

The ideal output of a NED tool should be the following:

- In snippets p_2 & p_3 the real category of *Argentina* is Location.
  - *belongs*(*occur*$_1$(*Argentina*, p_2), Location)
  - *belongs*(*occur*$_1$(*Argentina*, p_3), Location)
- In snippet p_1 the real category of *Argentina* is Fish Species.
  - *belongs*(*occur*$_1$(*Argentina*, p_1), Fish Species)
- In snippet p_4 there are 2 occurrences of *Argentina*. The 1st refers to Fish Species, while the 2nd to Location.

  - $belongs(occur_1(Argentina,\ p_4)$, Fish Species)
  - $belongs(occur_2(Argentina,\ p_4)$, Location)
- Finally, in snippet p_5 it is highly possible that the entity *Argentina* refers to a celestial body. Even if we do not known if there is a asteroid or planet named Argentina, we could say that is more probable that this occurrence refers to a Location.

5.3 Research Directions

Some questions we need to have in mind during the selection of the appropriate disambiguation method, are the following:

1. Should we do something in advance (before the entity recognition)?
2. Should we use more information (external data)? Which and how?
3. Just do something at execution time (without having to do anything in advance, or requiring any extra information)?

5.4 Possible Approaches

Below we describe 3 methods that we propose for disambiguating the occurrences of one (or more) identified ambiguous named entities, and then we evaluate them over the above motivating examples. Let $Amb(doc)$ be the set of ambiguous entities identified in document doc, m be a NED method and $numOfCA(m, doc)$ the number of ambiguous entities of doc that have been annotated with their real category by method m (correct annotations). Notice that if there are 2 or more occurrences of the same entity name that belong to different categories, they are considered as different entities and are counted accordingly. In order to estimate the precision of our methods we define the metric:

$$prec(m, doc) = \frac{numOfCA(m, doc)}{|Amb(doc)|}$$

In detail it holds that :

$$numOfCA(m, doc) = \sum_{\substack{e \in Ent(doc) \\ ctg(e) > 1}} \sum_{i=1}^{occurCount(e, doc)} \underbrace{belongs(occur_i(e, doc), bestCtg(occur_i(e, doc))) == TRUE}_{\text{we consider that the statement is equal to 1 if it holds, 0 otherwise}}$$

and

$$|Amb(doc)| = |\{e \in Ent(doc) | ctg(e) > 1\}| = \sum_{\substack{e \in Ent(doc) \\ ctg(e) > 1}} occurCount(e, doc)$$

5.4.1 Method 1: `Category Frequency in Snippets`

The main idea is that if a document *doc* contains many entities that belong to a specific category c, then the $occur_i(e, doc)$ will probably refer to the same category. We define the probability that the real category of the `i-th` occurrence of entity e is c, as follows:

$$P(belongs(occur_i(e, doc), c)) = \frac{|Ent(doc, c)|}{|Ent(doc)|}, \quad |Ent(doc)| > 0 \tag{5.1}$$

However, it is more accurate to take into account only the categories of the non-ambiguous entities. Let $NonAmb(doc) = \{e \in Ent(doc) \big| |ctg(e)| = 1\}$ be the set of non-ambiguous entities identified in *doc*, and $NonAmb(doc, c) = Ent(doc, c) \cap NonAmb(doc)$ be the set of non-ambiguous entities identified in *doc* and belong to category c. Therefore, the above probability can be modified into:

$$P(belongs(occur_i(e, doc), c)) = \frac{|NonAmb(doc, c)|}{|NonAmb(doc)|}, \quad |NonAmb(doc)| > 0 \tag{5.2}$$

Even in this case, we can not be sure about the ambiguity of an entity because it depends on current configuration, i.e. the set of supported categories. Specifically, suppose that we are only interested for Fish Species. In this case, the `Argentina` will be concerned as non-ambiguous entity, which is not correct. Either way, this method assigns the same category to all occurrences of an entity e.

If the supported categories are $C = \{c_{Fish}, c_{Loc}\}$, where c_{Fish} corresponds to Fish Species and c_{Loc} to Location, then the expected output of m_1, for the motivation examples, is the following:

- In snippet p_1 the only non-ambiguous entity is the *Salmon*.

  The outcome of eq. 5.1 is:

  - $P(belongs(occur_1(Argentina, p_1), c_{Fish})) = \frac{|Ent(p_1, c_{Fish})|}{|Ent(p_1)|} = \frac{2}{2} = 1$
  - $P(belongs(occur_1(Argentina, p_1), c_{Loc})) = \frac{|Ent(p_1, c_{Loc})|}{|Ent(p_1)|} = \frac{1}{2} = 0.5$

  The outcome of eq. 5.2 is:

  - $P(belongs(occur_1(Argentina, p_1), c_{Fish})) = \frac{|NonAmb(p_1, c_{Fish})|}{|NonAmb(p_1)|} = \frac{1}{1} = 1$
  - $P(belongs(occur_1(Argentina, p_1), c_{Loc})) = \frac{|NonAmb(p_1, c_{Loc})|}{|NonAmb(p_1)|} = \frac{0}{1} = 0$

  In both cases, it holds that $bestCtg(occur_1(\texttt{Argentina}, p_1)) =$ Fish Species, which is the real, so the precision is $prec(m_1, p_1) = \frac{numOfCA(m_1, p_1)}{|Amb(p_1)|} = \frac{1}{1} = 1$.

- In snippet p_2 the only non-ambiguous entity is the *Chile*.

  The outcome of eq. 5.1 is:

  - $P(belongs(occur_1(Argentina, p_2), c_{Fish})) = \frac{|Ent(p_2, c_{Fish})|}{|Ent(p_2)|} = \frac{1}{2} = 0.5$
  - $P(belongs(occur_1(Argentina, p_2), c_{Loc})) = \frac{|Ent(p_2, c_{Loc})|}{|Ent(p_2)|} = \frac{2}{2} = 1$

  The outcome of eq. 5.2 is:

  - $P(belongs(occur_1(Argentina, p_2), c_{Fish})) = \frac{|NonAmb(p_2, c_{Fish})|}{|NonAmb(p_2)|} = \frac{0}{1} = 0$
  - $P(belongs(occur_1(Argentina, p_2), c_{Loc})) = \frac{|NonAmb(p_2, c_{Loc})|}{|NonAmb(p_2)|} = \frac{1}{1} = 1$

  In both cases, it holds that $bestCtg(occur_1(\texttt{Argentina}, p_2)) = \mathsf{Location}$, which is the real, so the precision is $prec(m_1, p_2) = \frac{numOfCA(m_1, p_2)}{|Amb(p_2)|} = \frac{1}{1} = 1$.

- In snippet p_3 `&` p_5 there is not any non-ambiguous entity, so the eq. 5.2 cannot be measured (division with 0).

  The outcome of eq. 5.1, for $i = 3, 5$, is:

  - $P(belongs(occur_1(Argentina, p_i), c_{Fish})) = \frac{|Ent(p_i, c_{Fish})|}{|Ent(p_i)|} = \frac{1}{1} = 1$
  - $P(belongs(occur_1(Argentina, p_i), c_{Loc})) = \frac{|Ent(p_i, c_{Loc})|}{|Ent(p_i)|} = \frac{1}{1} = 1$

  The probabilities are equal for both categories, so we cannot select one. In such cases we select the most `popular` category of the specific entity. In our approach we assume that the most popular category is this with the highest frequency in the collection of documents that we have already check, and there is not a tie in at least one of the probabilities 5.1 and 5.2. Let $freq(e, c)$ be the frequency of entity e with category c. Consequently, the probability of 5.1 can be modified into

  $$P(belongs(occur_i(e, doc), c)) = freq(e, c), \quad \sum_{c \in ctg(e)} freq(e, c) = 1$$

  An other option is to keep the category that has the most incoming edges in a semantic graph of a KB. In these special cases we assume that the precision of our method is

  $$prec(m, doc) = \frac{1}{ctg(e)}$$

  Thus, the precision for $i = 3, 5$ is $prec(m_1, p_i) = 0.5$

- In snippet p_4 the non-ambiguous entity is the *Buenos Aires* .

  The outcome of eq. 5.1, for $i = 1, 2$ is:

  - $P(belongs(occur_i(Argentina, p_4), c_{Fish})) = \frac{|Ent(p_4, c_{Fish})|}{|Ent(p_4)|} = \frac{1}{2} = 0.5$
  - $P(belongs(occur_i(Argentina, p_4), c_{Loc})) = \frac{|Ent(p_4, c_{Loc})|}{|Ent(p_4)|} = \frac{2}{2} = 1$

  The outcome of eq. 5.2, for $i = 1, 2$ is:

  - $P(belongs(occur_i(Argentina, p_4), c_{Fish})) = \frac{|NonAmb(p_4, c_{Fish})|}{|NonAmb(p_4)|} = \frac{0}{1} = 0$
  - $P(belongs(occur_i(Argentina, p_4), c_{Loc})) = \frac{|NonAmb(p_4, c_{Loc})|}{|NonAmb(p_4)|} = \frac{1}{1} = 1$

  In both equations, for $i = 1, 2$, it holds that $bestCtg(occur_i(\texttt{Argentina}, p_4)) =$ Location, which is true only for the 2nd occurrence of *Argentina*, so the precision is $prec(m_1, p_4) = \frac{numOfCA(m_1, p_4)}{|Amb(p_4)|} = \frac{1}{2} = 0.5$.

As we observe, method m_1 finds the real category of *Argentina* only in snippets p_1, p_2 and p_4 (2nd occurrence). In the rest of them the resulting probabilities do not give us enough information so as to find the real category. The main reason is that m_1 is based only on text statistics, which works well only if all occurrences of an ambiguous entity refer to the same category. However, quite often we have to handle the case that two occurrences of the same entity label belong to different categories, e.g. in p_4 the 1st occurrence of entity *Argentina* refers to Fish Species, whereas the 2nd to the Location. Hence, we need a method that can exploit the semantics of a document textual content, so as to be able to select the best category for each entity occurrence.

5.4.2 Method 2: `Distance in the Semantic Web`

As stated above, a restriction of method m_1 is that ignores the semantics of documents. However, these semantics offer valuable information that let us to understand better the text content and find the proper category for each entity occurrence. Thus, in method m_2 we estimate the semantic distance between two entities. There are many definitions and metrics for computing the semantic distance between two words. Of course this requires having a corpus of information about words and concepts, e.g. a dictionary, an ontology, a thesaurus, either a generic one (e.g. WordNet) or domain specific. We hereafter assume that we have one such source in the form of an RDF graph (e.g. DBpedia). We make this assumption because RDF is the `'lingua franca'` for expressing metadata and knowledge in the sense that any dictionary, thesaurus or ontology can be described in RDF form. As we mentioned in Section 3.1, we assume that one such RDF graph is accessible through a SPARQL endpoint.

In our approach we consider the semantic distance of entities e and t as the

length of the shortest (undirected) path that connects their semantic resources (URIs) in a specific semantic graph, where t can either be an entity, a concept, a text fragment or phrase that has been exported from the text around e, denoted as s. We extend the notations of Section 3.1 and we denote as $\mathtt{U}(e, c)$ the set of URIs that are related to e, when e belongs to category c. Hence, the distance of e and t is denoted as

$$dist(e,t) = min\{dist_{graph}(u_e, u_t) \big| u_e \in U(e,c), u_t \in U(t,c'), c \in ctg(e), c' \in ctg(t)\}$$

It is worth mentioning that the possible categories of t, $ctg(t)$, are not known in advance, so we need to find them when we want to conduct experiments. A possible way to detect them is to use the template query shown in Figure 5.1. More specifically this query searches for resources whose `rdf:label` property value contains the t, and returns their resource classes, which may not belong to the set of supported categories. In the same way we can retrieve the semantic resources of t, in the case that belongs to a specific category, using the template query shown in Figure 5.2. Notice that the queries contain the character sequences `[FRAGMENT]` and `[CLASS]` (including the [and]) which are replaced (at query-time) by the current text fragment and resource class.

In the ideal case we assume that exist at least one non-ambiguous named entity t, so if we know its semantic resources (URIs) we can then estimate its distance from all candidate resources of an ambiguous entity e and keep only the resource with the minimum, i.e. compute the $argmin_{c \in ctg(e)} dist_{graph}(\mathtt{U}(e,c), \mathtt{U}(t,c'))$, where c' if fixed. This approach is resembling with the `Entity Enrichment` functionality described in Section 3.3, where we can form an RDF graph by collecting the RDF triples of the corresponding entities and conclude whether and how these entities are related (Figure 3.8).

The above process, as described in Algorithm 3, is as follows: Initially we apply a `NLP` function (e.g. `Gate Annie`) to split the textual snippet s into tokens and we store in S only these that are nouns and noun phrases (see line 4). Then we keep only the linkable elements of set S and store them in S' (see line 5), i.e. text terms that can be matched with one (or more) semantic resources. In the final step, for each possible combination of semantic resources of $occur_i(e, doc)$ and linkable fragment $term$ of s we estimate their semantic distance, using the Djikstra's algorithm[1], and we keep the category c of $occur_i(e, doc)$ from the combination with the minimum distance (see lines 6-13). This category we presume that is the best.

However, most of these entities correspond to at least one abstract category, such as `http://www.w3.org/2002/07/owl#Thing`. Consequently, many semantically irrelevant entities seem to have semantic distance equal to 2, which has a negative influence in our results. Therefore, in order to avoid such improper connections we exclude from the RDF graph all the triples that have as predicate the `rdf:type`[2] and as object any of these abstract semantic categories.

[1] `https://en.wikipedia.org/wiki/Dijkstra's_algorithm`

[2] rdf : `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

Concerning the selection of s, we assume that there is a method that given a specific occurrence of e returns the respective textual snippet. A naive approach is to return the textual content between two '.'. Notice that in this case the size of s is not fixed and can be ranged from a few words to a whole paragraph. Hence, it is preferable to use a sliding window, with adjustable size, centered in $occur_i(e, doc)$. Finally, the semantic resources existence of t can be controlled through the queries shown in Figures 5.1 and 5.2 as we mentioned above (see line 5).

Algorithm 3 Estimate Minimum Semantic Distance

Input: e, a named entity, s, a textual snippet
Output: $categ$, the best category of e

1: $S \leftarrow \emptyset$
2: $S' \leftarrow \emptyset$
3: $minDist \leftarrow k$ // k is a large integer
4: $S \leftarrow findNouns(s)$ // returns the nouns and noun phrases of s
5: $S' \leftarrow findLinkableElements(S)$ // returns the linkable elements of S'
6: **for all** $term \in S'$ **do**
7: **for all** $c \in ctg(e)$ **do**
8: **for all** $c' \in ctg(term)$ **do**
9: $dist \leftarrow dist(U(e, c), U(term, c'))$
10: **if** $minDist > dist$ **then**
11: $minDist \leftarrow dist$
12: $categ \leftarrow c$
13: **return** $categ$

```
SELECT DISTINCT ?class WHERE {
  ?uri rdfs:label ?label FILTER(regex(str(?label), "[FRAGMENT]", "i"))
  ?uri rdf:type ?class }
```

Figure 5.1: Example of a SPARQL query for retrieving the possible categories of a fragment.

```
SELECT DISTINCT ?uri WHERE {
  ?uri rdf:type <[CLASS]>
  ?uri rdfs:label ?label FILTER(regex(str(?label), "[FRAGMENT]", "i")) }
```

Figure 5.2: Example of a SPARQL template query for linking an text fragment with resources in a KB.

The expected output of m_2 for the above motivating examples, using the semantic

graph of DBpedia (as depicted in Figure 5.3) is the following:

- In snippet p_1 the candidate semantic resources of *Argentina & Salmon* are `http://www.dbpedia.org/page/Argentina_(fish)`, `http://www.dbpedia.org/page/Argentina` & `http://www.dbpedia.org/page/Salmon`. Their semantic distances are:
  - `dist(dbpedia:Argentina_(fish), dbpedia:Salmon) = 2`
  - `dist(dbpedia:Argentina, dbpedia:Salmon) =` ∞

  The above results indicate that $bestCtg(occur_1(\texttt{Argentina}, p_1)) = \textsf{Fish Species}$. In this case the precision is $prec(m_2, p_1) = \frac{numOfCA(m_2,p_1)}{|Amb(p_1)|} = \frac{1}{1} = 1$.
- In snippet p_2 the candidate semantic resources of *Argentina & Chile* are `http://www.dbpedia.org/resource/Argentina_(fish)`, `http://www.dbpedia.org/resource/Argentina` & `http://www.dbpedia.org/resource/Chile`. Their semantic distances are:
  - `dist(dbpedia:Argentina_(fish), dbpedia:Chile) =` ∞
  - `dist(dbpedia:Argentina, dbpedia:Chile) = 2`

  The above results indicate that $bestCtg(occur_1(\texttt{Argentina}, p_2)) = \textsf{Location}$. In this case the precision is $prec(m_2, p_2) = \frac{numOfCA(m_2,p_2)}{|Amb(p_2)|} = \frac{1}{1} = 1$.
- In snippet p_3 we have:
  - `dist(dbpedia:Argentina_(fish), dbpedia:Sebastian_Leto) =` ∞
  - `dist(dbpedia:Argentina, dbpedia:Sebastian_Leto) = 2`

  It holds that $bestCtg(occur_1(\texttt{Argentina}, p_3)) = \textsf{Location}$, so the precision is $prec(m_2, p_3) = \frac{numOfCA(m_2,p_3)}{|Amb(p_3)|} = \frac{1}{1} = 1$.
- In snippet p_4 the semantic distances are:
  - 1st occurrence of *Argentina*
    - `dist(dbpedia:Argentina_(fish), dbpedia:Sushi_pizza) = 3`
    - `dist(dbpedia:Argentina, dbpedia:Sushi_pizza) =` ∞
  - 2nd occurrence of *Argentina*
    - `dist(dbpedia:Argentina_(fish), dbpedia:Buenos_Aires) =` ∞
    - `dist(dbpedia:Argentina, dbpedia:Buenos_Aires) = 1`

In this snippet there are two occurrences of *Argentina*. For the `1st` occurrence it holds that $bestCtg(occur_1(\texttt{Argentina}, p_4)) = \textsf{Fish Species}$, while for the `2nd` that $bestCtg(occur_2(\texttt{Argentina}, p_4)) = \textsf{Location}$. So the precision is $prec(m_2, p_4) = \frac{numOfCA(m_2,p_4)}{|Amb(p_4)|} = \frac{2}{2} = 1$.

- Finally, in snippet p_5 it holds that:

  - `dist(dbpedia:Argentina_(fish), dbpedia:Earth) =` ∞
  - `dist(dbpedia:Argentina, dbpedia:Earth) = 2`

  For this snippet it holds that $bestCtg(occur_1(\texttt{Argentina}, p_5)) = \textsf{Location}$. However, it neither belongs to Location nor Fish Species, so the precision is $prec(m_2, p_5) = \frac{numOfCA(m_2,p_5)}{|Amb(p_5)|} = \frac{0}{1} = 0$.

From the above results, we conclude that m_2 performs better than m_1 in most cases. The only snippet where both of them fail to find the real category of *Argentina* is the p_5. In such snippets, where the entity e does not belong to any of the supported categories c, it is rational that none method will return the right category. Generally, the quality of results of m_2 depends mainly on retrieved entities and the used ontology of used KB. However, according to our requirements we may need more than one KBs, which may be structured over a totally different ontology. Therefore, we need to ensure that there will not be any conflict even in the case that we use many KBs.

In addition, m_2 requires a KB that contains information for a wide set of categories. To meet this requirement, we can either use a public KB, or create our own. However, relying on a public KB could prove problematic in the case of a server downtime. On the other hand, the creation and maintenance of our own dataset requires advanced computing equipment. Thus, for our needs we create a local copy of DBpedia that includes only RDF triples of specific categories (see 6.3) and we keep them in a local instance of OpenLink Virtuoso Server[3]. Furthermore, we assume that t concerns only named entities, since all linkable document terms will correspond to entries of the supported categories. This lead us to miss valuable information which could help us to find the proper entity. To make this evident, in our experiments we create a new category named *Rest*, whose entities are defined by hand and belong to various categories (different from these that we already support). Then we estimate the difference in achieved precision when we also support this category (see more in 6.3).

5.4.3 Method 3: `Train Set-based/Text Categorization`

The method m_3 relies on `Text Categorization`, whose aim is to classify a collection of documents, based on their textual content, into various categories, e.g.

[3] `http://virtuoso.openlinksw.com/`

```
prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix dbr:  <http://dbpedia.org/resource/>
prefix dbo:  <http://dbpedia.org/ontology/>
prefix dbp:  <http://dbpedia.org/property/>
prefix yago: <http://yago-knowledge.org/resource/>



dbr:Argentina_(fish)      rdf:type        dbo:Fish.
                          .
                          .
                          .
dbr:Salmon                rdf:type        dbo:Fish.
                          .
                          .
                          .
dbr:Argentina             rdf:type        dbo:Location.
dbr:Argentina             rdf:type        yago:YagoGeoEntity.
dbr:Argentina             dbp:capital     dbo:Buenos_Aires.
                          .
                          .
                          .
dbr:Buenos_Aires          rdf:type        dbo:Location.
                          .
                          .
                          .
dbr:Sebastian_Leto        dbp:birthPlace  dbr:Buenos_Aires.
                          .
                          .
                          .
dbr:Sushi_pizza           dbo:ingredient         dbr:Salmon
                          .
                          .
                          .
dbr:Earth                 rdf:type        yago:YagoGeoEntity.
```

Figure 5.3: Semantic graph snapshot of DBpedia

sports, politics, electronics, etc. This classification is achieved by using one or more Machine Learning (ML) Classifiers, e.g. Naive Bayes, Decision Tree, Support Vector Machine (SVM). A ML classification consists of two main steps, the `training` and the `classification`. In the training step we select the classifier features, i.e. the parameters that will be used to represent the textual content of a sentence or a whole document. This is a paramount step and we must be careful so as to avoid side effects such as overfitting and underfitting. In the classification step, every new sample is represented using the same parameters and based on their values the classifier returns the most possible category.

In our approach these parameters correspond to document terms that give us enough information so that we can classify a new document d to a specific category c. Let $docs(c_i)$ be the set of documents of category c_i, $docs(w, c_i) = \{d \in docs(c_i) \big| Contains(d, w)\}$ the set of documents of category c_i that contain word w,

and $Contains(d, w)$ denotes that word w occurs in document d. In order to measure the importance of each word w we define the `Inverse Category Frequency` (`ICF`) as:

$$ICF(w) = \frac{|C|}{\sum_{c_i \in C} \frac{|docs(w,c_i)|}{|docs(c_i)|}}$$

or

$$ICF'(w) = \frac{|C| + 1 - |C_w|}{|C|} \times \sum_{c_i \in C} \frac{|docs(w, c_i)|}{|docs(c_i)|}$$

where C_w is the set of categories that word w occurs in.

Our methodology is as follows:

1. Find the words W from a document collection (ignoring stopwords) that can be used as parameters.
2. Estimate the $ICF(w), \forall w \in W$.
3. Sort W w.r.t. ICF.
4. Select the best-K words for each category that will be used as parameters.
5. Create a Train Set using the best-K words as parameters.
6. Train a classifier.
7. Classify a new sample.

However, based on a brief experimental evaluation, we conclude that the ICF measure is insufficient. Therefore, we need an additional ranking formula which can be placed as an intermediate step between `3` and `4`. A possible solution is to exploit the frequency of word w in the documents of a specific category. The new formula is defined as:

$$ScoreB(w, c_i) = \frac{|docs(w, c_i)|}{|docs(c_i)|}$$

To make it clear, suppose that we use as document collection the textual snippets from the above motivating examples, where each snippet is annotated with the relevant category (Table 5.1). Thus, we can create a train set, as depicted in Table 5.2, where each parameter is initialized with `0` or `1` depending on whether the corresponding word occurs in the documents of a specific category.

A restriction of ML classifiers is that they require a large document corpus as train set. However, this is an initial step that it performed once, and in the

Samples	Text Content	Category
s1	Argentina and Salmon live in deep waters.	c_{Fish}
s2	Argentina is sharing land borders with Chile.	c_{Loc}
s3	Sebastian Leto was born in Argentina .	c_{Loc}
s4	I like sushi pizza with Argentina.	c_{Fish}
s5	I plan to visit Argentina because I have a cousin who studies in the university of Patagonia and he could host me there.	c_{Loc}

Table 5.1: Train Set document collection

	Parameters								
Samples	deep	waters	land	borders	born	sushi	pizza	university	**Category**
s1	1	1	0	0	0	0	0	0	c_{Fish}
s2	0	0	1	1	0	0	0	0	c_{Loc}
s3	0	0	0	0	1	0	0	0	c_{Loc}
s4	0	0	0	0	0	1	1	0	c_{Fish}
s5	0	0	0	0	0	0	0	1	c_{Loc}

Table 5.2: Train Set example

cases that we want to classify a new domain/category of entities or to add more documents for the existing categories.

Nevertheless, the improvement of ML methods and techniques is out of scope of our work since we emphasize on usage of Linked Data. However we include this approach because it is used for text classification/categorization which is an equivalent problem with NED. Hence, in order to avoid any implementation failure and to ensure its proper operation, we use the well known ML toolkit `Weka` [39]. More specific, we use the `FilteredClassifier` class with the `StringToWordVector` as filter, and we evaluate the performance of some well known classifiers, e.g. Naive Bayes, Support Vector Machine, Decision Trees, etc.

Chapter 6

Evaluation

First, in §6.1 & §6.2 we report a *user study* that demonstrate the usability of `X-Link`, and a *case study* regarding the efficiency of the functions described in §3. Then, in §6.3 we report the results of a comparative experimental evaluation of the NED methods described in §5.

6.1 Task-based User Study

In [33] we performed a user study whose purpose was a) to test the *usability* of the proposed approach, i.e. how *fast* and *conveniently* a user can configure `X-Link`, and b) to identify usability problems that will allow us to improve the tool. Note that the target user is an *administrator* or a *developer* who wants to use `X-Link` for building and dynamically configuring an application. Below we summarize the results.

6.1.1 Tasks and Scenario

We deployed `X-Link` as a Web application configured for the marine domain which can identify Fish Species in a text or Web document. The administrator of the system can change the configuration through an administration page. Specifically, he/she can add, remove and update categories, specify how to link and enrich the identified entities, and define the SPARQL endpoints to use.

The 11 subjects that participated in the user study are 23 to 34 years old, members of the Information Systems Laboratory at FORTH-ICS, they have computer science background and a basic knowledge of Linked Data and the SPARQL query language. We shortly (in about 5 minutes) described and demonstrated the application and its functionality to the participants, and then we asked them to perform the following tasks:

(T1) *Add* a new category of entities
(T2) *Update* a category
(T3) Specify how to *link* the identified entities of a category
(T4) Specify how to *enrich* the entity URIs of a category

(T5) Inspect the *connectivity* (for $r = 1$) of the entity URIs

For the tasks T1 to T4, the participants could also load an example of a SPARQL query and modify it (instead of writing it from scratch). We recorded whether they succeeded to complete each task, as well as the time to successfully accomplish each task.

6.1.2 Results

The results showed that all participants managed to complete the tasks T1, T2 and T5. However, 18% of the participants (two persons) failed to complete T3 and 9% (one person) failed to complete T4. The difficulty behind T3 and T4 is the comprehension of the *SPARQL template query*, specifically, the purpose of the template parameter and how it is used for constructing the template query.

This proves that adopting the LOD-based approach that we propose and understanding the notion of the SPARQL template queries, one can configure a NEE system within a few minutes. However these tasks can be proved laborious even for users with computer science background. We should also stress that if we had dedicated more time for explaining the notion of the template queries (e.g. with more examples), perhaps all the participants would have also successfully completed T3 and T4.

6.1.3 Formulation of SPARQL Queries

There are many tools that can facilitate the construction of SPARQL queries, without requiring any advanced knowledge in SPARQL [12, 57]. Furthermore, there are natural language approaches that guide users in formulating queries in a language seemingly akin to English and translate them to SPARQL [26]. In this thesis, we consider that the administrator of the underlying application knows the SPARQL query language.

6.2 Case Study: Querying Online DBpedia

In [33] we also performed a case study for testing the feasibility of the entire approach. Specifically, we used online DBpedia as the underlying KB and we measured the time for (1) creating a new category, (2) linking an identified entity with semantic resources, (3) enriching an entity URI, and (4) inferring the connectivity of the entity URIs.

6.2.1 Creating a New Category

We used 7 sets of DBpedia resource classes. Each set has 5 different resource classes containing a particular number of entities (thus, totally 35 different resource classes were used). Each resource class actually corresponds to the new category that we want to create in `X-Link`. We measured a) the time for running the SPARQL

query at DBpedia's SPARQL endpoint (which retrieves the labels of the entities belonging to the corresponding resource class), and b) the time for reading the answer and creating the category in `X-Link`.

As expected, the most time consuming task is the execution of the SPARQL query, since we query DBpedia's SPARQL endpoint at real time (the remaining tasks cost less than 10 seconds in all cases). We saw that for resource classes with small number of entities (up to 10,000) the time is less than 20 seconds, while for resources classes with about 100,000 entities the time is about 5 minutes.

6.2.2 Time for Linking an Identified Entity

For this task we used 8 sets of DBpedia resource classes, each one containing classes of a particular number of entities. Each set has 5 different resource classes (thus, totally 40 different resource classes were used). Note that each resource class actually corresponds to the category of an identified entity. For every resource class, we randomly selected 10 labels of entities belonging to that class and measured the average time for running the SPARQL query.

The results showed that for entities belonging to categories with up to 100,000 entities, the average time is less than 1 second, while for entities in categories with up to 1 million entities, the linking time is about 5 seconds. In addition, for linking an entity belonging to a category with 6 million entities the time is about 25 seconds.

6.2.3 Time for Enriching an Entity URI

For this task we ran experiments for the following types of properties: i) incoming, ii) outgoing, iii) outgoing of a specific language, and iv) union of incoming and outgoing.

We randomly selected 160 URIs from DBpedia and measured the average time required for retrieving the properties. The required time is very low (less than 300 ms) for all types of properties, since the entity URI for which we want to retrieve properties is known, thus no many string comparisons are required like in the case of entity linking.

6.2.4 Time for Inspecting the Connectivity of the Entity URIs

We ran experiments for $r = 1$ and $r = 2$. Obviously, the time depends on the number of entity URIs for which we want to inspect the connectivity. We ran experiments for 10, 50 and 100 randomly selected URIs belonging to the same resource class.

Results showed that for $r = 1$ the time is proportional to the number of URIs (specifically, about 10 seconds are required for every 50 URIs). However, for $r = 2$ the task is very time consuming; the time increases exponentially to the number of URIs (e.g. for 100 URIs about 12 minutes are required). This is a predictable result since each URI may have many related URIs.

It is worth mentioning that the above experiments are carried out using the `DefaultGazetteer`, otherwise the respective execution time for some tasks might differ. Recall §4.2.2 where the semantic resource of entity *"Vatican City"* is `http://dbpedia.org/resource/Vatican_City`, whereas the term *"Vatican"* may either refers to named entities *"Vatican"* and *"Vatican City"*, whose semantic resources are `http://dbpedia.org/resource/Prisoner_in_the_Vatican`, `http://dbpedia.org/resource/Vatican_City`. In such cases we have to pay the (negligible) extra cost.

6.3 Disambiguating Named Entities

We performed a comparative experimental evaluation of the NED methods proposed in §5. Specifically, we evaluated them according to the following aspects: (1) their precision, and (2) the required time for disambiguating an entity appearance in a document, with respect to its length. Our objective is to find the method that achieves the best trade-off of effectiveness and time efficiency, according to our needs.

6.3.1 Evaluation Collection

Since NED still remains an open challenge, there is not any widely accepted corpus of documents that can be used as an evaluation dataset for our setting. Recall that we are looking for the most probable category c of an ambiguous entity a. Thus, we created our own gold standard which consist of documents that contain occurrences of one (or more) ambiguous entities, whose real category is known in advance.

The methodology that we use for the creation of our evaluation dataset is briefly described in Algorithm 4. Specifically, consider a set of ambiguous entities denoted as A and a set of categories C, where $\forall a \in A, \exists c, c' \in C : ctg(a) = \{c, c'\}$. For each $a \in A$ and for each $c \in ctg(a)$ we select a set of documents/paragraphs that mention a and its sense is certainly c (see line 4). These documents can either be collected manually or automatically. For the second case, we submit a query like *"<ambiguous entity name> <category> <helpful comments>"* in any search engine (e.g. Google), we keep the top-K results (e.g. K = 5) and we store them in the ideal dataset. Notice that the term *helpful comments* is optional and is used if we want to retrieve documents with specific content, e.g. *"Argentina Country Debt Crisis"*.

A paramount issue in our evaluation is to spot any difference in the effectiveness and performance of our methods when they are applied in documents of different size. For this reason we split the retrieved documents (into smaller) and we create additional datasets, where each of them corresponds to a different scale of granularity. In our evaluation we use 4 scales of granularity, (1) whole text, (2) half text, (3) per 200 words, and (4) per $\approx 15-25$ words (snippet size), denoted as D_1, D_2, D_3 and D_4 respectively. Either way, each one of the new (sub)documents

must contain at least one occurrence of ambiguous entity a, otherwise we remove it from the corresponding dataset. Also, we create 2 versions of the above datasets, one including stopwords and one omitting them, and we evaluate our methods over them.

Evaluation Setting In our evaluation, we suppose that $A = \{Apple,\ Argentina,\ Beetle,\ Ibiza,\ Jaguar,\ Java,\ Mustang\}$, whose candidate categories are:

- $ctg(Apple) = \{Company,\ Plant\}$
- $ctg(Argentina) = \{Country,\ Fish\ Species,\ Plant\}$
- $ctg(Beetle) = \{Automobile,\ Insect\}$
- $ctg(Ibiza) = \{Automobile,\ Island\}$
- $ctg(Jaguar) = \{Automobile,\ Mammal\}$
- $ctg(Java) = \{Island,\ Programming\ Language\}$
- $ctg(Mustang) = \{Automobile,\ Mammal\}$

Hence, we have $|A| = 7$ ambiguous entities, and each entity corresponds to $\lambda = avg\{|ctg(a)|\big|a \in A\} \geq 2$ possible categories. Consequently, we will need $|A| \times \lambda \simeq 14$ queries. For each query we keep only the top-15 results, so we will retrieve 210 web pages. Assuming that each web page has a size of 2000 words, then dataset D_1 will consist of 210 documents, D_2 of 420 documents, D_3 of 2100 documents, and D_4 of $\approx$ 16800 documents (the real numbers are reported in Table 6.1). Also, for each web page we keep a properties file with some useful information, i.e. its URI, the title of the retrieved doc, the real category of ambiguous entity, etc.

Notice that most of modern search engines use various heuristics for improving the quality of searching results. Such heuristics include user personalization, results diversity, exploitation of query content, etc. However, search results tend to be user centric, which is not desirable in our case. Therefore, we use the Bing Search Engine, which allows us to ignore such heuristics, and we retrieve the desired documents through the query `http://www.bing.com/search?q=[QUERY]&count=[RESULTS]&first=0&format=rss`, where the the character sequences `[QUERY]` and `[RESULTS]` (including the [and]) are replaced (at query-time) by the current query and the number of search results we want to keep.

Additionally, due to the nature of ML methods (m_3) we need separate corpus of documents for the training and classification phase. In any case, for the reliability of results, all methods must be evaluated on same dataset. Nevertheless, as stated earlier we will create our own dataset, hence we decide to use as train set the 80% of our initial corpus (the first 12 documents of each ambiguous entity from dataset D_1 and the respective from D_2, D_3, D_4), and use the rest 20% for the evaluation.

Dataset	Train Set	Test Set
D_1	162	45
D_2	303	88
D_3	1544	350
D_4	5190	1264
Total documents	**7215**	**1747**

Table 6.1: Dataset contents

Algorithm 4 Create Evaluation Dataset

Input: A, a set of ambiguous entities,
k, the number of documents we want to retrieve
Output: D, a collection of documents
1: $D \leftarrow \emptyset$
2: **for all** $a \in A$ **do**
3: **for all** $c \in ctg(a)$ **do**
4: $D \leftarrow D \cup retrieveDocs(a, c, k)$ // returns k documents about entity a of sense c
5: **return** D

6.3.2 Evaluation Process

The objective of our evaluation, as we mentioned above, is the assessment of the achieved precision of our methods (as it is defined in §5.4) when they are applied on documents of different size. A second issue is the measurement of the required disambiguation time of each method, and the investigation whether is affected by document length.

Within our experiments, we apply our methods on the aforementioned datasets. Initially, we retrieve the named entities from each document of datasets D_1, D_2, D_3, D_4 using a NER tool (e.g. `X-Link`). Afterwards, we try to disambiguate them using each individual method. During the disambiguation step every (ambiguous) entity is matched with the best category in the manner described in §5.4.1, §5.4.2 and §5.4.3. To this end we compare the annotated categories with the real, we count the correct matchings and we then estimate the achieved precision in each dataset. Recall that each document corresponds to a specific ambiguous entity, whose real category is already known and is stored in the corresponding *properties file*. The above process is briefly described in Algorithm 5.

Our experiments are structured in such way so that they can be reproduced anytime, using any set of ambiguous entities and corpus of documents. Also we are able to evaluate a new method and compare it with the existing ones.

Algorithm 5 Evaluation Process

Input: $Methods$, a set of NED methods,
$Granularity$, a set of granularity scales,
$AmbEnt$, a set of ambiguous entities
Output: $Results$, the file of results

```
1: Docs ← CreateEvalDataset() // create and returns an evaluation dataset (Alg. 4)
2: for all a ∈ AmbEnt do
3:   for all m ∈ Methods do
4:     for all g ∈ Granularity do
5:       for all doc ∈ Docs(a, g) do //for all documents relating entity a and granularity g
6:         C_prob ← getCategoryOf(a, doc, m) // guess the category of a in doc using method m
7:         compare(C_prob, C_real) // compare the real category of a with this annotated by m
8:         update(Results)
9: return Results
```

6.3.3 Achieved Precision

We evaluate our methods over both versions of datasets D_1, D_2, D_3 and D_4 (with and without stopwords), and we compare their performance when we use the *exact* and *partial matching* functionality. From now on, we denote as m_1 the *Category Frequency* method, as m_2 the *Semantic Distance* method with radius equal to 1, and as m_3 the *Text Classification* method that uses the Support Vector Machine algorithm which is trained using documents of dataset D_4, since it achieves the best precision over the rest algorithms, as depicted in Figure 6.1.

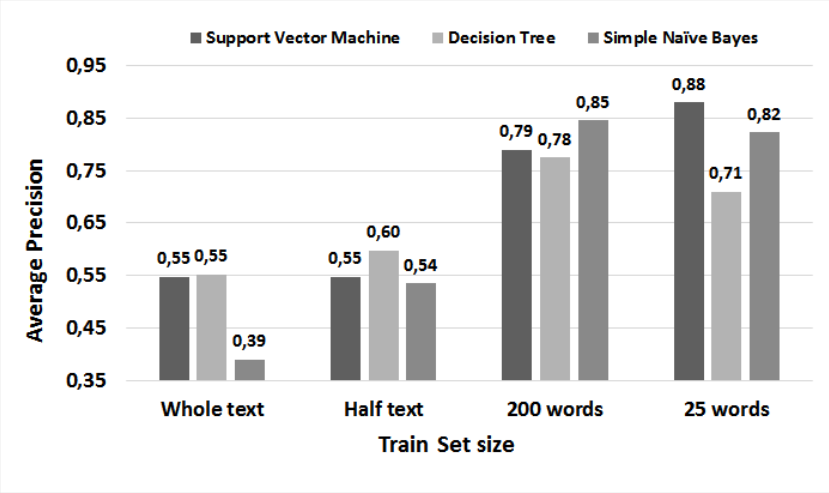

Figure 6.1: Achieved precision of ML algorithms using as train set documents of different size

Figures 6.2, 6.3, 6.4, 6.5 illustrate the achieved precision of our methods when they are applied on documents of different size, using *exact matching*. As we can see, m_3 outperforms the rest methods in all datasets except from D_4 (see Figures 6.5, 6.6) . This happens because ML algorithms exploit only the text content of a document, e.g. keywords, terminology, etc, and are unaware of retrieved entities. In contrast, m_1 and m_2 are based solely in the output of a NEE process and the semantic relations between identified entities, which are not always available. Also, recall the fact that a document term can be incorrectly identified as entity, which may also influences their performance.

In addition, as we mention above (see also 5.4.2), we support only a few categories of entities. This lead us to miss many semantic relations, which could help us to find the proper entity. To test this conjecture we create an additional category named *Rest*, which consists of entities of various categories, that are defined by hand. As shown in Figure 6.8, if we also support the *Rest* category the precision of m_2 increased by 10.7%[1], which realizes our hypothesis. The same may holds for m_1 if we use a general category, e.g. *Eukaryote*, instead of specific ones such as *Fish, Mammal, Insect*, etc, however due to limitations of computing resources we cannot test it. Notice that the experiments are conducted using the *Rest* category.

Regarding the dataset D_4, we observe that for small snippets, e.g. tweets, where the textual content is restricted, the m_2 and m_3 differ only by 3.9%[2], which makes evident the importance of text semantics, i.e. the semantic relations between mined entities.

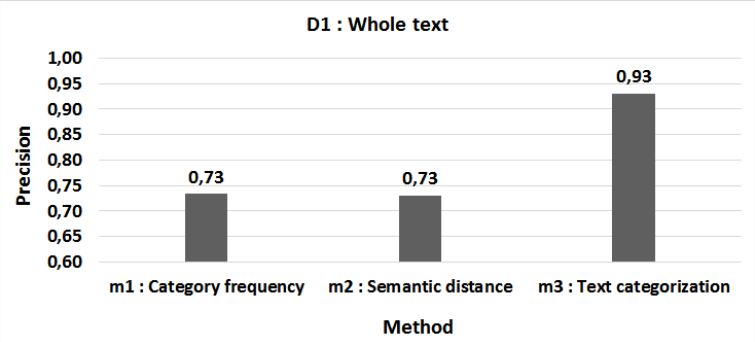

Figure 6.2: Achieved precision of our methods on dataset D_1

[1] `Percentage change` $= (\Delta V/V_{old}) * 100 = (V_{new} - V_{old}/V_{old}) * 100$

[2] `Percentage difference` $= (|\Delta V|/(\Sigma V/2)) * 100 = (|(V_{new} - V_{old})|/((V_{new} + V_{old})/2)) * 100$

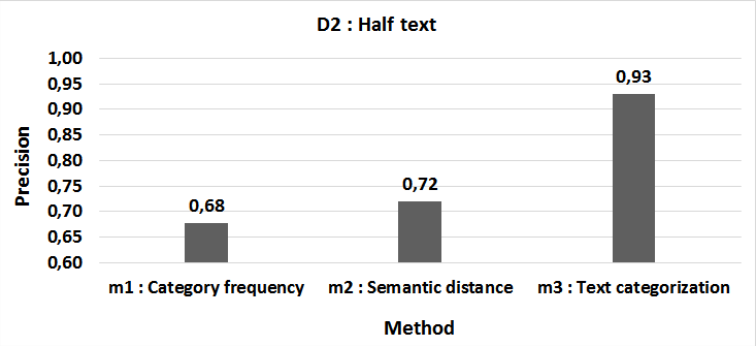

Figure 6.3: Achieved precision of our methods on dataset D_2

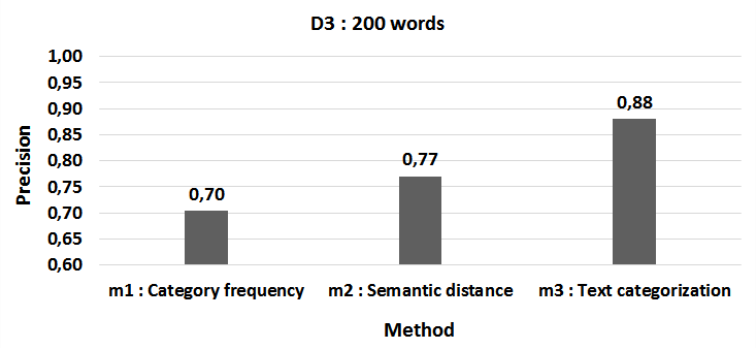

Figure 6.4: Achieved precision of our methods on dataset D_3

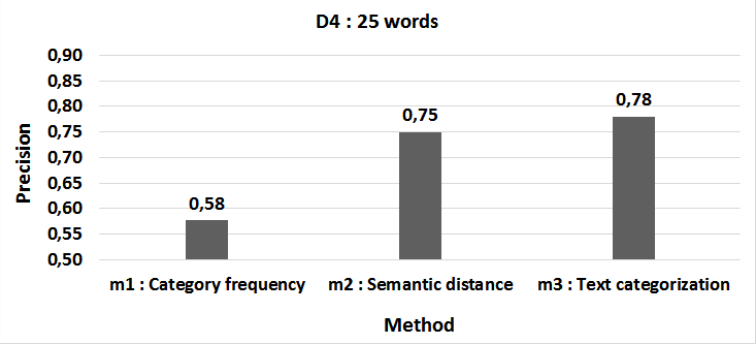

Figure 6.5: Achieved precision of our methods on dataset D_4

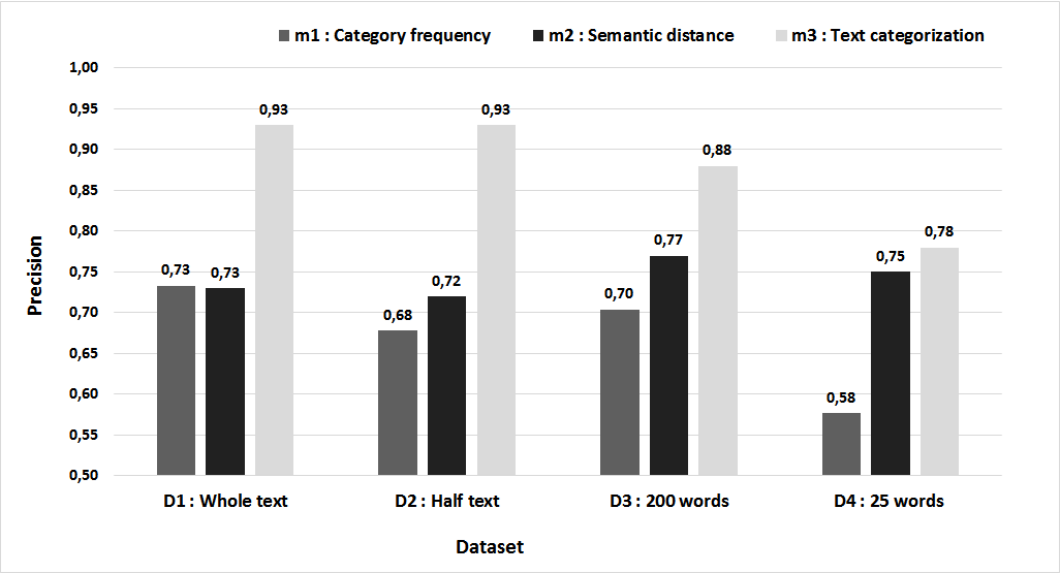

Figure 6.6: Achieved precision of our methods in both datasets

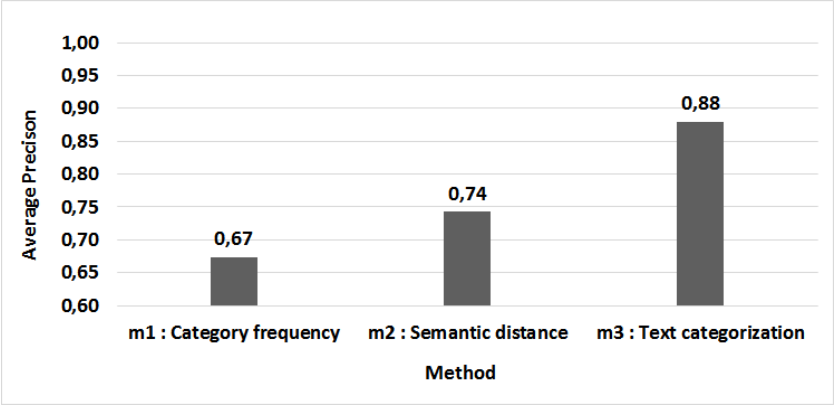

Figure 6.7: Average achieved precision of our methods

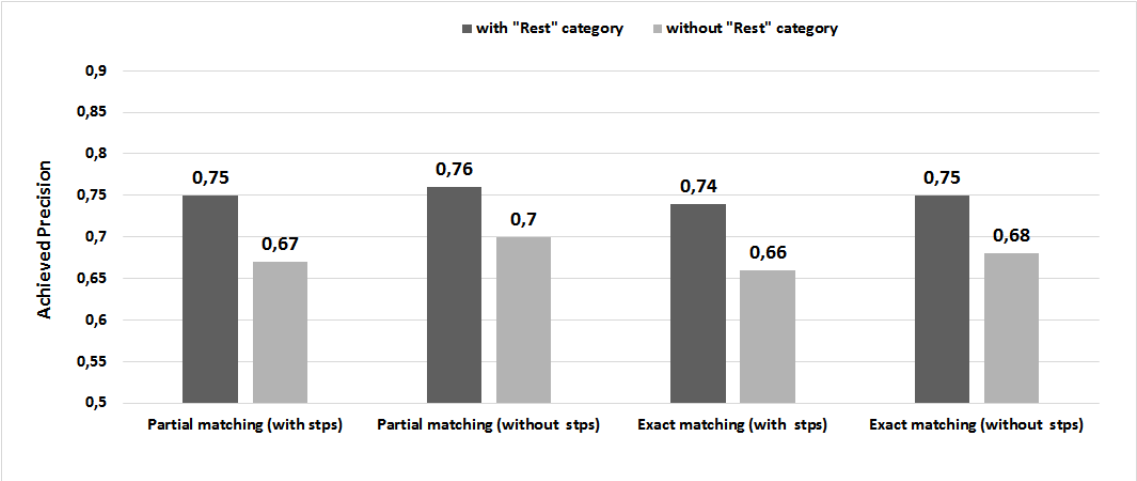

Figure 6.8: Average precision of m_2 when we support or not the *Rest* category

Figures 6.9, 6.10 6.11 and 6.12 depict the impact of omitting the stopwords from a document. As we observe, the precision of our methods improved only by 1.42% on average, since it is extremely rare that a stopword corresponds to an entity name, so the output of m_1 and m_2 remains almost the same. Additionally, ML algorithms (m_3) use many heuristics for extracting the best features, i.e text terms, that represent a document and are used to find its appropriate category. So, it is likely that they ignore already most of the stopwords.

Concerning the impact of partial matching, the results are not the expected, since the precision increased only 3% for m_1 and 1.35% for m_2 as shown in Figures 6.13, 6.14 and 6.16, while m_3 remains the same, since it ignores the NEE output (Figure 6.15). At priori, this could mean that either there are not many partial matchings that help us to find the best category, or that the exact matchings give us enough information for disambiguating an entity occurrence.

6.3.4 Time for Disambiguating an Entity occurrence

Figure 6.17 reports the average required disambiguating time of our methods. As we observe the most time consuming method is the m_2, since it requires 8.3 seconds (on average) for disambiguating an entity occurrence when we use the *exact matching* functionality and 36.5 seconds (on average) when we use *partial matching*. The main reason is that m_2 uses external data (one or more KBs), whose access and process is time consuming. Recall that we retrieve the semantic resources (RDF triples) of identified entities from a KB through a SPARQL query, we create a graph based on their relations and then we keep the pair of resources

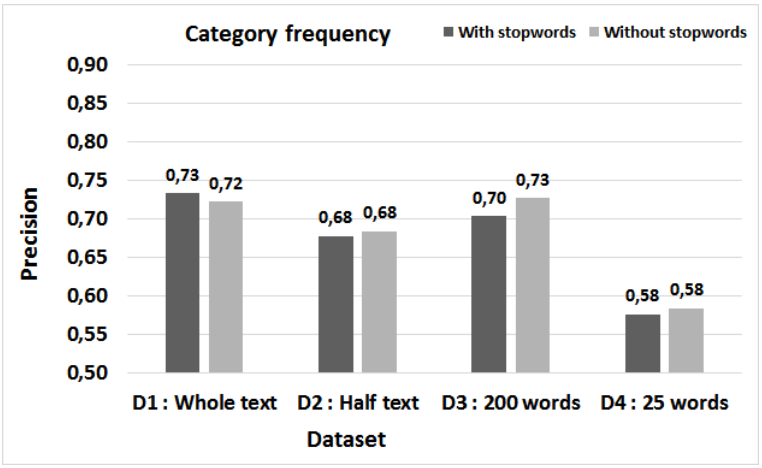

Figure 6.9: Achieved precision of m_1 in both versions of datasets (with and without stopwords)

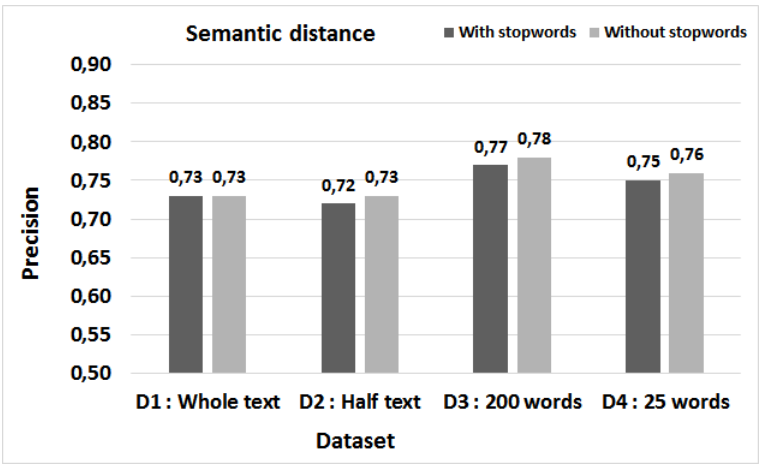

Figure 6.10: Achieved precision of m_2 in both versions of datasets (with and without stopwords)

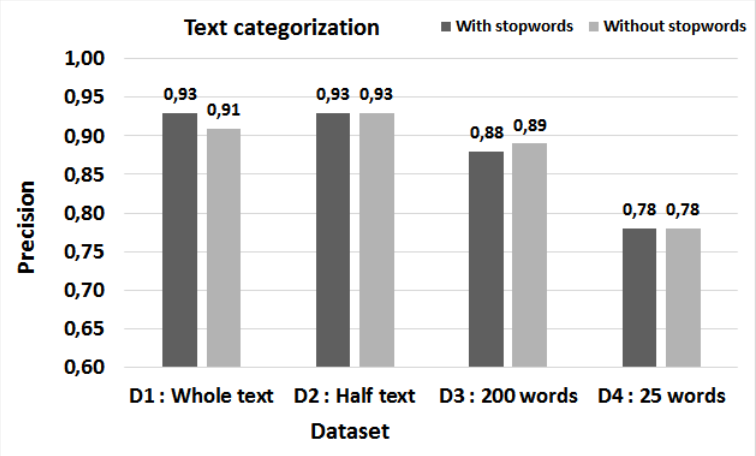

Figure 6.11: Achieved precision of m_3 in both versions of datasets (with and without stopwords)

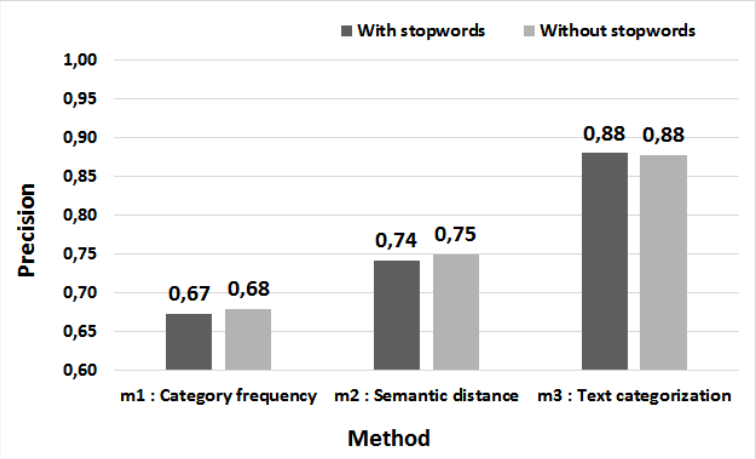

Figure 6.12: Average precision of our methods in both versions of datasets (with and without stopwords)

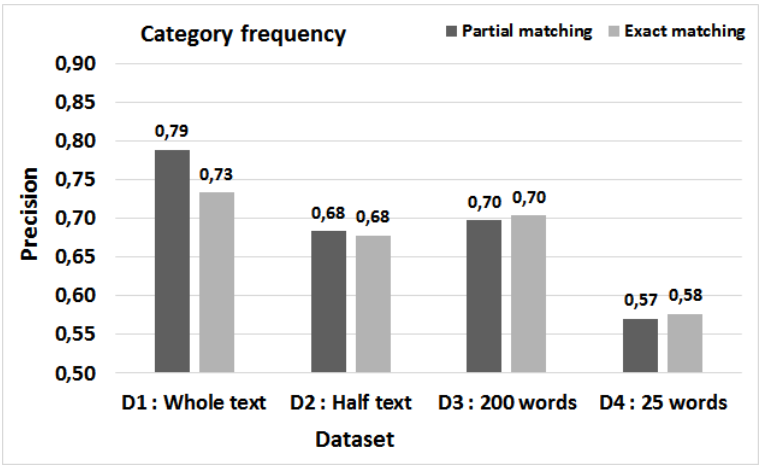

Figure 6.13: Average precision of m_1 when we use partial and exact matching

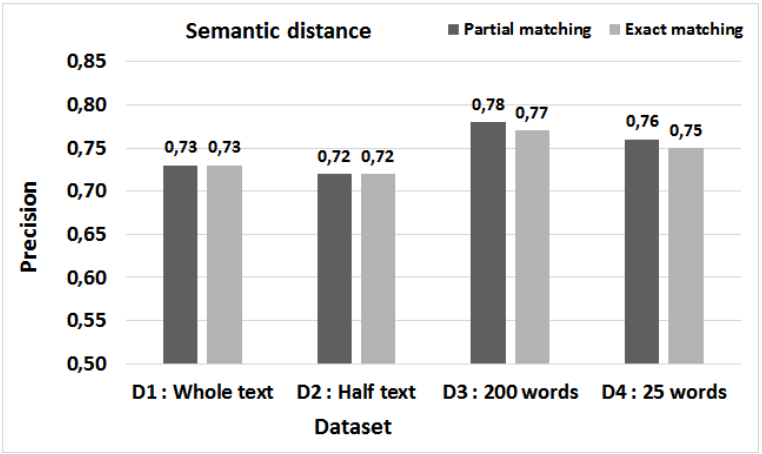

Figure 6.14: Average precision of m_2 when we use partial and exact matching

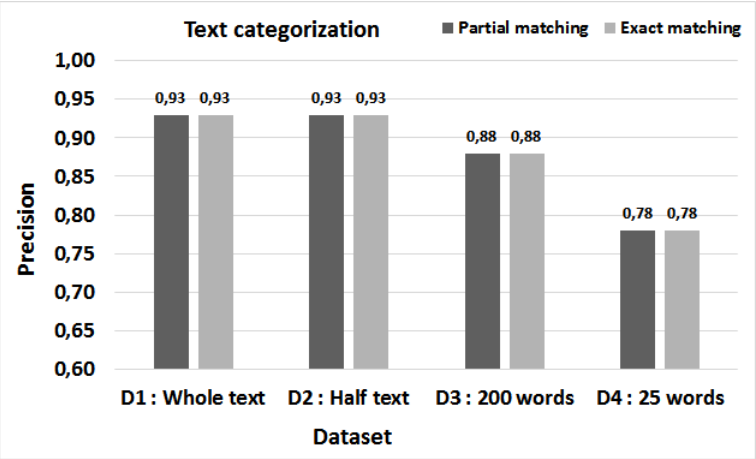

Figure 6.15: Average precision of m_3 when we use partial and exact matching

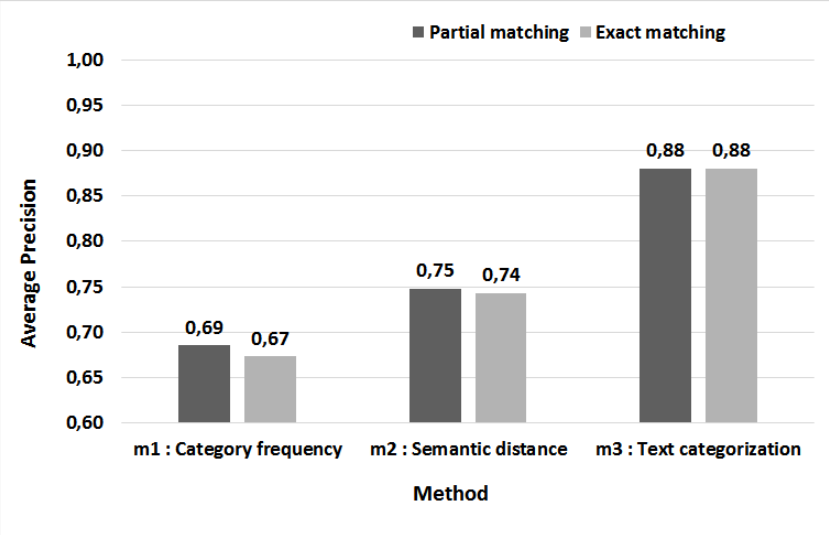

Figure 6.16: Average precision of our methods when we use partial and exact matching

with the minimum distance.

On the contrary, the m_1 and m_3 require 0.11 and 0.34 seconds in both cases. This happens because m_1 estimates only the occurrence frequency of each supported category, whereas m_3 classifies a new text sample based on its extracted features, where both tasks have a minor cost. However, m_3 requires a train set, whose creation according to its size can last from several minutes to hours, even days.

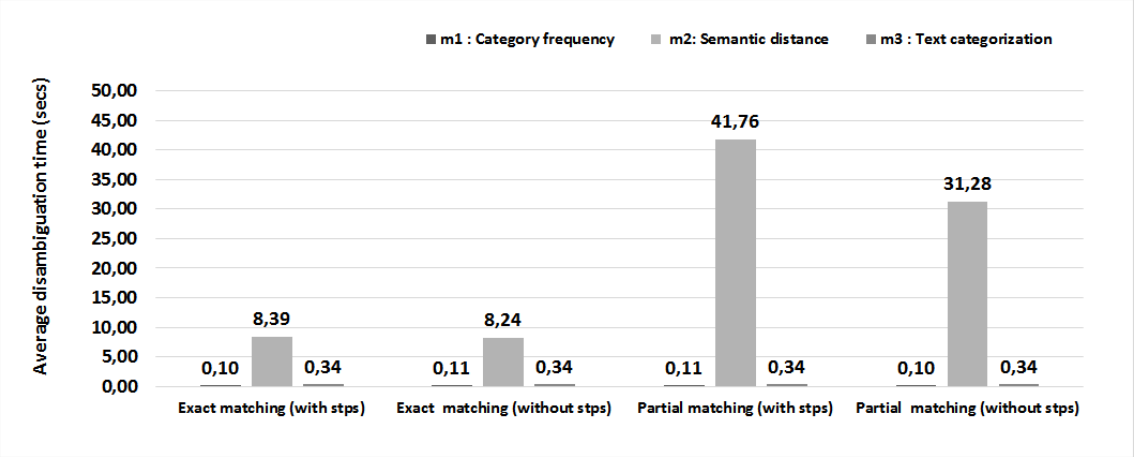

Figure 6.17: Average disambiguation time of our methods

Figure 6.18 depicts the effect of supporting the *Rest* category in m_2 from the perspective of execution time. As we can see, especially in the case that partial matching is enabled, it takes 3 times longer to disambiguate the same document. Whereas, this gap is reduced when we use exact matching. In any case, it is a cost we have to pay since it improves the achieved precision by 10.7% as we mentioned above.

6.3.5 Synopsis

The above results showed us that the achieved precision of method m_1 is ranged from 58% up to 73%, the m_2 from 72% up to 77% and m_3 between 78% and 93%, whereas their average precision (in both datasets) is 67% for m_1, 74% for m_2, and 88% for m_3, as depicted in Figure 6.7. From the corresponding execution times we see that the most time consuming method is the m_2, since it requires up to 36 seconds (on average) when we use the *partial matching* functionality and 8.3 seconds in the case of *exact matching*. The rest methods are much faster, since m_3 requires 0.34 second and m_1 only 0.1 second.

Hence, it is evident that m_3 succeed the best trade-off between effectiveness

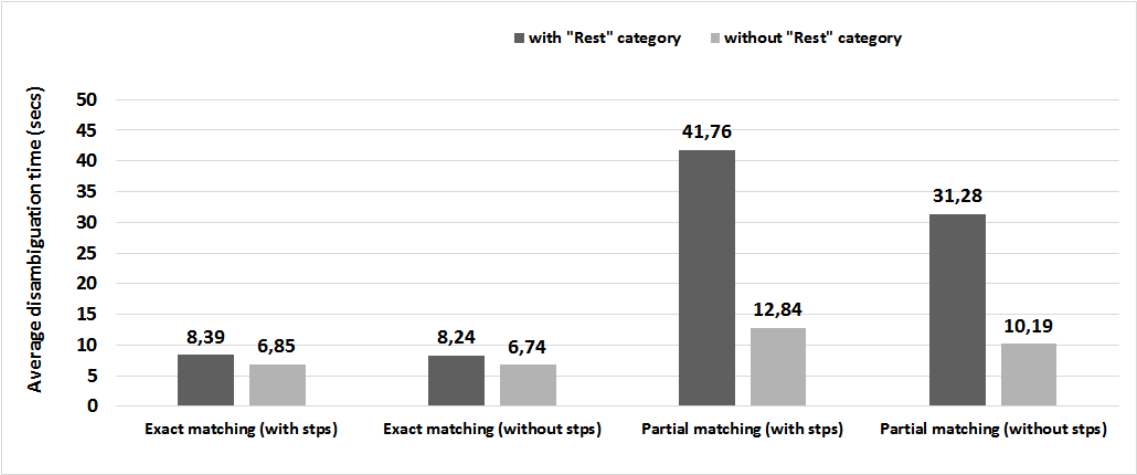

Figure 6.18: Average disambiguating time of m_2 when includes or not the *Rest* category

and performance. However, it requires a large corpus of documents for the training phase, which must reproduced every time we want to support a new category of interest. In addition, the quality of results highly depends on train set, so we must be very careful when we select the classifiers features because this task is vulnerable to underfitting and overfitting problems. Moreover, for poor structured and illegible snippets, m_3 achieve almost the same precision with m_2, which ignores the textual content and exploits the semantic relations between mined entities.

Concerning m_2, it could be nice if we could also parse the textual content of given documents and find their part of speech. This could be useful in the sense that we can disambiguate an entity occurrence, basis on whether it appear as subject or object. For example *Mustang* can act (e.g. gallop, neigh) and appear as subject almost only in the case that it refers to the *Mammal* and not the *Automobile*. For this approach we can use a dictionary, e.g. WordNet.

As regards the *partial matching* functionality, our results failed to highlight its value. Therefore we plan to conduct further experiments in the future and evaluate our methods over different datasets for more reliable results.

Although the related systems are evaluated over different datasets and we cannot compare them with our approach, however it is worth mentioning their evaluation results just to be aware of their accuracy. Specifically, in AIDA Hoffart et al. use as dataset documents from CoNLL 2003[3], while the achieved precision is 81.91%. In that work the precision is defined as the fraction of mention-entity assignments that match the ground-truth assignment (as in our case). In DBpedia Spotlight Mendes et al. performed an annotation and a disambiguation evaluation. For the annotation evaluation authors used 35 manually annotated paragraphs

[3] `http://www.cnts.ua.ac.be/conll2003/ner`

from New York Times articles from 8 different domains, where each linkable phrase of these paragraphs is matched with a DBpedia resource. The achieved F_1 score was reached up to 56%. In the other hand, for the disambiguation evaluation they used 155.000 randomly selected wikilink samples, and the accuracy was reached up to 80.52%. In AGDISTIS Usbeck et al. used data from benchmark dataset `N3` [56], from AIDA evaluation corpus[4], and from Cornolti et al. [23] benchmark, where the achieved F-measure was varied between 31% and 87%. Finally, in Babelfy Moro et al. used data from Senseval-3 for English Word Sense Disambiguation, SemEval-2007 task 7 and 17 and SemEval-2013 task 12 datasets. The achieved F_1 score was varied between 71.6% and 87.4%.

As we can see, most of these systems use the F_1 score (F-score or F-measure) to measure their accuracy, where $F_1 = 2 * (precision * recall)/(precision + recall)$. The precision and the recall in that case are defined as follows: *"In a classification task, the precision for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class). Recall in this context is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labeled as belonging to the positive class but should have been)."* [5]

From the above results we conclude that the performance of related systems highly depends on the format of documents that are used during their evaluation (document size, provenance), for this reason there are significant variations in their performance, as in our case. Specifically, the greatest difference is observed in AGDISTIS where the minimum (31%) and maximum (87%) F-measure differ by 95%. Similarly, in Babelfy the achieved F_1 scores differ by 20%, since is evaluated over multilingual datasets. In our work the highest difference is noticed in m_1 where the succeed precisions differ by 23%, whereas in m_3 the difference is 17.5%, and in m_2 they differ by 6.7%.

All experiments are carried out in an ordinary laptop with processor Intel i7 @ 2.50 GHz CPU, 16GB RAM and running Windows 10 (64 bit). The implementation of all approaches was in Java 1.7, using Weka ML library for m_3 and OpenLink Virtuoso Server 7.1 for m_2.

[4] `https://www.mpi-inf.mpg.de/departments/databases-and-informationsystems/research/yago-naga/aida/downloads`

[5] `https://en.wikipedia.org/wiki/Precision_and_recall`

Chapter 7

Conclusion

We have proposed a method that exploits Linked Data for *configuring* dynamically and handily a NEE system. For tackling the configuration requirements we have defined a generic configuration model, while for being able to exchange a supported configuration we have proposed an RDF/S vocabulary, called *Open NEE Configuration Model*. By publishing the configurations supported by one or more NEE services using the proposed model, an application can dynamically discover and use the NEE services that best satisfy its annotation needs. In addition, a NEE service that is able to read such configurations can dynamically (at request-time) use a given configuration for annotating a set of documents.

We should stress that it would be beneficial for the community if every NEE system supported the configuration model that we propose for making them LOD-aware, and also if every NEE system published the configurations supported by its services using the Open NEE Configuration Model.

Furthermore, we have presented the design and functionality of `X-Link`, a fully configurable (LOD-based) NEE framework that realizes the proposed configuration model. `X-Link` allows the user/administrator to easily define the categories of entities that are interesting for the application at hand, as well as to update a category and specify how to link and enrich the identified entities, by exploiting one or more online semantic KBs. This enhanced configurability allows `X-Link` to be used for building and dynamically configuring domain-specific applications (e.g. for identifying *drugs* in a medical search system, for annotating and exploring *fish species* in a marine-related web page, etc.).

In addition, we proposed a set of methods for *disambiguating* ambiguous named entities, that are appropriate for our setting. Primarily we consider that the best category is this with the highest occurrence frequency. However this method is unable to utilize the content of documents, which is very important in our case. For this reason in the next method we exploit the semantics of mined entities and we investigate their relations. This method assumes that the proper sense corresponds to the semantic resource (URI) of the ambiguous entity that is closer (semantically) to rest entities and concepts. Finally, the third method classifies

a document into a specific category, using machine learning algorithms, such as Support Vector Machine, Decision Tree, Simple Naive Bayes, etc.

Then, we evaluate the proposed methods over documents of different size using search results of Bing. In our experiments we measure the achieved precision and the required disambiguation time for each of them. The results showed that our methods achieve precision up to 88% (on average). We conclude that the *Text Categorization* method achieves the best trade-off in large and middle size documents, whereas in small documents, e.g. tweets, e-mails, where the textual content is restricted it achieves almost the same precision with the *Semantic Distance* method.

Regarding future work and research, there are several aspects that are worth investigating. One is to extend the proposed configuration model to allow modeling also non-functional aspects of the NEE service like the average annotation time, the average linking time, etc. Our long term vision is to offer a model that can wholly describe the functionality, the API (i.e. how to use it) and the configurability of a NEE service. This would allow a client application to dynamically discover and use NEE services by exploiting only standard Web protocols, without needing to set up a corresponding service. Moreover, we would like to improve our NED methods according to evaluation results, and especially extend the *Semantic Distance* method in order to parse and exploit the textual content of documents (e.g. part of speech) except from their semantics, i.e relations between contained entities. As regards `X-Link`, a future direction is to incorporate our proposed methods for automated entity disambiguation, evaluate them over a wide set of ambiguous entities, and compare them with other well-known NED tools.

Bibliography

[1] AlchemyAPI. http://www.alchemyapi.com/.

[2] FAO Fisheries Linked Open Data. http://www.fao.org/figis/flod/.

[3] Lupedia Enrichment Service, Ontotext. http://lupedia.ontotext.com/.

[4] OpenCalais, Thomson Reuters. http://www.opencalais.com/.

[5] RDF Schema 1.1. http://www.w3.org/TR/rdf-schema/.

[6] Resource Description Framework (RDF). http://www.w3.org/RDF/.

[7] SPARQL 1.1 Federated Query, W3C Recommendation, 21 March 2013. http://www.w3.org/TR/sparql11-federated-query/.

[8] SPARQL endpoint. http://semanticweb.org/wiki/SPARQL_endpoint.

[9] Universal Resource Identifier (URI). http://www.w3.org/Addressing/URL/URI_Overview.html.

[10] Wikimeta. http://www.wikimeta.com/.

[11] Yahoo! content analysis api. http://developer.yahoo.com/contentanalysis/.

[12] Oszkár Ambrus, Knud Möller, and Siegfried Handschuh. Konduit VQB: a Visual Query Builder for SPARQL on the Social Semantic Desktop. In *Workshop on Visual Interfaces to the Social and Semantic Web*, 2010.

[13] Teresa K Attwood, Douglas B Kell, Philip McDermott, James Marsh, SR Pettifer, and David Thorne. Utopia documents: linking scholarly literature with research data. *Bioinformatics*, 26(18):i568–i574, 2010.

[14] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*. Springer, 2007.

[15] Dave Beckett and Brian McBride. Rdf/xml syntax specification (revised). *W3C recommendation*, 10, 2004.

[16] Barry Bishop, Atanas Kiryakov, Damyan Ognyanov, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. Factforge: A Fast Track to the Web of Data. *Semantic Web*, 2(2):157–166, 2011.

[17] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story so Far. *International Journal on Semantic Web and Information Systems*, 5(3), 2009.

[18] Kalina Bontcheva, Valentin Tablan, Diana Maynard, and Hamish Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3-4):349–373, 2004.

[19] David Carmel, Ming-Wei Chang, Evgeniy Gabrilovich, Bo-June (Paul) Hsu, and Kuansan Wang. Erd'14: Entity recognition and disambiguation challenge. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '14, pages 1292–1292. ACM, 2014.

[20] Eric Charton, Michel Gagnon, and Benoit Ozell. Automatic Semantic Web Annotation of Named Entities. In *Advances in Artificial Intelligence*, pages 74–85. Springer, 2011.

[21] Paolo Ciccarese, Marco Ocana, Leyla Jael Garcia Castro, Sudeshna Das, and Tim Clark. An open annotation ontology for science on web 3.0. *Journal of Biomedical Semantics*, 2, 2011.

[22] Paolo Ciccarese, Marco Ocana, and Tim Clark. Open semantic annotation of scientific publications using domeo. *Journal of biomedical semantics*, 3:1–14, 2012.

[23] Marco Cornolti, Paolo Ferragina, and Massimiliano Ciaramita. A framework for benchmarking entity-annotation systems. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 249–260, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[24] Silviu Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *EMNLP-CoNLL*, volume 7, pages 708–716. Citeseer, 2007.

[25] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[26] Marco Trevisan Enrico Franconi, Paolo Guagliardo. Quelo: a NL-based Intelligent Query Interface. In *2nd Workshop on Controlled Natural Languages (CNL 2010)*, 2010.

[27] P. Fafalios, I. Kitsos, Y. Marketakis, C. Baldassarre, M. Salampasis, and Y. Tzitzikas. Web Searching with Entity Mining at Query Time. In *Proceedings of the 5th Information Retrieval Facility Conference*, 2012.

[28] P. Fafalios, P. Papadakos, and Y. Tzitzikas. Enriching textual search results at query time using entity mining, linked data and link analysis. *International Journal of Semantic Computing*, 2015.

[29] P. Fafalios, M. Salampasis, and Y. Tzitzikas. Exploratory Patent Search with Faceted Search and Configurable Entity Mining. In *1st International Workshop on Integrating IR technologies for Professional Search (ECIR'13 Workshop)*, 2013.

[30] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2013.

[31] P. Fafalios and Y. Tzitzikas. Exploratory Professional Search through Semantic Post-Analysis of Search Results. In *Professional Search in the Modern World*, volume 8830 of *Lecture Notes in Computer Science*, pages 166–192. Springer, 2014.

[32] P. Fafalios and Y. Tzitzikas. Post-analysis of keyword-based search results using entity mining, linked data and link analysis at query time. In *2014 IEEE Eighth International Conference on Semantic Computing (ICSC 2014)*, Newport Beach, California, USA, June 16-18 2014. IEEE.

[33] Pavlos Fafalios, Manolis Baritakis, and Yannis Tzitzikas. Configuring named entity extraction through real-time exploitation of linked data. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, page 10. ACM, 2014.

[34] Pavlos Fafalios, Manolis Baritakis, and Yannis Tzitzikas. Exploiting linked data for open and configurable named entity extraction. *International Journal on Artificial Intelligence Tools*, 24(2):1540012, 2015.

[35] Pavlos Fafalios and Panagiotis Papadakos. Theophrastus: On demand and real-time automatic annotation and exploration of (web) documents using open linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2014.

[36] Paolo Ferragina and Ugo Scaiella. Fast and accurate annotation of short texts with wikipedia pages. *arXiv preprint arXiv:1006.3498*, 2010.

[37] Sébastien Ferré and Alice Hermann. Semantic search: Reconciling expressive querying and exploratory search. In *The Semantic Web–ISWC 2011*, pages 177–192. Springer, 2011.

[38] Michel Gagnon, Amal Zouaq, and Ludovic Jean-Louis. Can We Use Linked Data Semantic Annotators for the Extraction of Domain-Relevant Expressions? In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 1239–1246. International World Wide Web Conferences Steering Committee, 2013.

[39] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[40] Xianpei Han and Jun Zhao. Named entity disambiguation by leveraging wikipedia semantic knowledge. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 215–224. ACM, 2009.

[41] Joseph Hassell, Boanerges Aleman-Meza, and I Budak Arpinar. *Ontology-driven automatic entity disambiguation in unstructured text*. Springer, 2006.

[42] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard De Melo, and Gerhard Weikum. YAGO2: Exploring and Querying World Knowledge in Time, Space, Context, and Many Languages. In *Proceedings of the 20th international conference companion on World wide web*, pages 229–232. ACM, 2011.

[43] Clement Jonquet, Nigam Shah, Cherie Youn, Chris Callendar, Margaret-Anne Storey, and M Musen. Ncbo annotator: semantic annotation of biomedical data. In *International Semantic Web Conference*, 2009.

[44] Guyonvarch Joris and Sébastien Ferré. Scalewelis: a scalable query-based faceted search system on top of sparql endpoints. In *Work. Multilingual Question Answering over Linked Data (QALD-3)*, 2013.

[45] Sayali Kulkarni, Amit Singh, Ganesh Ramakrishnan, and Soumen Chakrabarti. Collective Annotation of Wikipedia Entities in Web Text. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009.

[46] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia Spotlight: Shedding Light on the Web of Documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8. ACM, 2011.

[47] David Milne and Ian H Witten. An open-source toolkit for mining wikipedia. *Artificial Intelligence*, 194:222–239, 2013.

[48] Diego Mollá, Menno Van Zaanen, and Daniel Smith. Named Entity Recognition for Question Answering. *Proceedings of ALTW*, pages 51–58, 2006.

[49] Andrea Moro, Alessandro Raganato, and Roberto Navigli. Entity linking meets word sense disambiguation: A unified approach. *Transactions of the Association for Computational Linguistics*, 2, 2014.

[50] Gonzalo Navarro. A Guided Tour to Approximate String Matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[51] Bruno Woltzenlogel Paleo. An approximate gazetteer for gate based on levenshtein distance. In *Student Session of the European Summer School of Logic, Language and Information*, 2007.

[52] Dietrich Rebholz-Schuhmann, Miguel Arregui, Sylvain Gaudan, Harald Kirsch, and Antonio Jimeno. Text processing through web services: calling whatizit. *Bioinformatics*, 24(2), 2008.

[53] Giuseppe Rizzo and Raphaël Troncy. NERD: Evaluating Named Entity Recognition Tools in the Web of Data. In *ISWC 2011, Workshop on Web Scale Knowledge Extraction (WEKEX'11), October 23-27, 2011*, Bonn, GERMANY, 2011.

[54] Giuseppe Rizzo and Raphaël Troncy. NERD: A Framework for Unifying Named Entity Recognition and Disambiguation Extraction Tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–76. Association for Computational Linguistics, 2012.

[55] Giuseppe Rizzo, Raphaël Troncy, Sebastian Hellmann, and Martin Bruemmer. NERD meets NIF: Lifting NLP Extraction Results to the Linked Data Cloud. *LDOW*, 937, 2012.

[56] Michael Röder, Ricardo Usbeck, Sebastian Hellmann, Daniel Gerber, and Andreas Both. N3-a collection of datasets for named entity recognition and disambiguation in the nlp interchange format. *9th LREC*, 2014.

[57] Alistair Russell, Paul R. Smart, Dave Braines, and Nigel R. Shadbolt. NITELIGHT: A Graphical Tool for Semantic Query Construction. In *Semantic Web User Interaction Workshop (SWUI 2008)*, April 2008.

[58] Robert Sanderson, Paolo Ciccarese, and Herbert Van de Sompel. Open annotation data model. *W3C Community Draft*, 2013.

[59] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. A string metric for ontology alignment. In *The Semantic Web–ISWC 2005*, pages 624–637. Springer, 2005.

[60] Y. Tzitzikas, C. Alloca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating Heterogeneous and

Distributed Information about Marine Species through a Top Level Ontology. In *Proceedings of the 7th Metadata and Semantic Research Conference (MTSR'13)*, November 2013.

[61] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. Agdistis - graph-based disambiguation of named entities using linked data. In *The Semantic Web–ISWC 2014*. Springer, 2014.

[62] Mohamed Amir Yosef, Johannes Hoffart, Ilaria Bordino, Marc Spaniol, and Gerhard Weikum. AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. *Proceedings of the VLDB Endowment*, 4(12):1450–1453, 2011.