International Journal on Artificial Intelligence Tools
© World Scientific Publishing Company

Exploiting Linked Data for Open and Configurable Named Entity Extraction

PAVLOS FAFALIOS, MANOLIS BARITAKIS and YANNIS TZITZIKAS

Institute of Computer Science, Foundation for Research and Technology - Hellas, and Computer Science Department, University of Crete, GREECE
{fafalios,mbaritak,tzitzik}@ics.forth.gr

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

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, post-processing of search results, etc. However, existing NEE 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 paper, we show how we can exploit semantic information (Linked Data) at real-time for configuring (handily) a NEE system and we propose a generic model for configuring such services. To explicitly define the semantics of the proposed model, 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. To allow relating the output of a NEE process with an applied configuration, we propose an extension of the Open Annotation Data Model which also enables an application to run advanced queries over the annotated data. As a proof of concept, we present `X-Link`, a fully-configurable NEE framework that realizes this approach. Contrary to the existing tools, `X-Link` allows the user to easily define the categories of entities that are interesting for the application at hand by exploiting one or more semantic Knowledge Bases. The user is also able to update a category and specify how to semantically link and enrich the identified entities. This enhanced configurability allows `X-Link` to be easily configured for different contexts for building domain-specific applications. To test the approach, we conducted a task-based evaluation with users that demonstrates its *usability*, and a case study that demonstrates its *feasibility*.

Keywords: Named Entity Extraction, Named Entity Recognition, Semantic Annotation, Linked Data, Entity Mining, Entity Linking

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)[a]. 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. NEE is useful in several tasks, e.g. for question answering[1], post-processing of search results[2,3], annotating (Web) documents[4,5]. 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, e.g. DBpedia Spotlight[6], AlchemyAPI[7] and OpenCalais[8]. 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) or to *extend* an existing category with additional entities coming from a new KB. 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 Open Data (LOD)[9], 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.

To tackle this lack of functionality, in this paper:

- 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[10] vocabulary which allows a NEE system to describe (and publish as Linked Data) the entity mining capabilities of its services.

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

- We present `X-Link`, a fully configurable (LOD-based) NEE framework that we have designed and implemented which realizes the proposed configuration model.
- We report the results of a task-based *user study* that demonstrate the *usability* of the proposed approach.
- We report the results of a *case study* that demonstrate the *feasibility* of the proposed approach, and we discuss how we can achieve reliability and scalability.

In addition, to enable relating the output of a NEE process with an applied configuration, we propose an extension of the *Open Annotation Data Model*[11]. This extension allows also an application to run advanced (SPARQL[12]) queries over an annotated set of documents.

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

2. Motivation

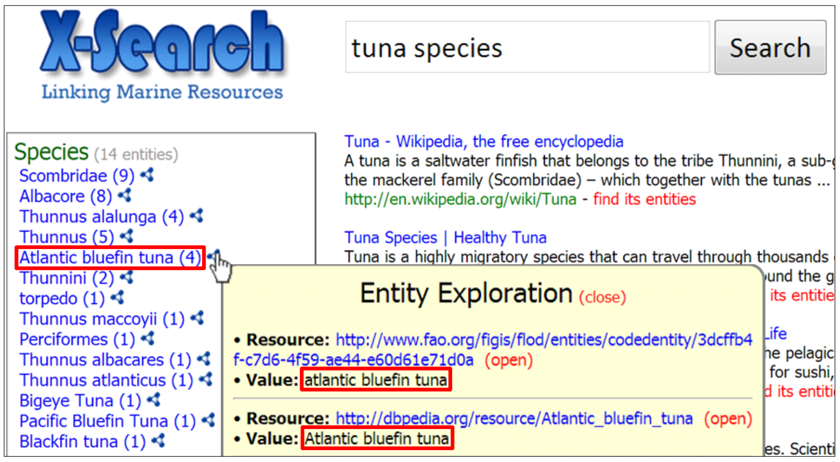
For justifying the value of the proposed approach, below we first present a vertical search scenario that stresses that different communities have different and ever-changing requirements, and then we discuss several benefits of adopting an open and exchangeable configuration model.

2.1. *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[b]:

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 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).* ◇

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



Species (14 entities)
Scombridae (9)
Albacore (8)
Thunnus alalunga (4)
Thunnus (5)
Atlantic bluefin tuna (4)
Thunnini (2)
torpedo (1)
Thunnus maccoyii (1)
Perciformes (1)
Thunnus albacares (1)
Thunnus atlanticus (1)
Bigeye Tuna (1)
Pacific Bluefin Tuna (1)
Blackfin tuna (1)

Tuna - Wikipedia, the free encyclopedia
A tuna is a saltwater finfish that belongs to the tribe Thunnini, a sub-
the mackerel family (Scombridae) – which together with the tunas ...
http://en.wikipedia.org/wiki/Tuna - find its entities

Tuna Species | Healthy Tuna
Tuna is a highly migratory species that can travel through thousands

Entity Exploration (close)

• **Resource:** http://www.fao.org/figis/flod/entities/codedentity/3dcffb4f-c7d6-4f59-ae44-e60d61e71d0a (open)
• **Value:** atlantic bluefin tuna

• **Resource:** http://dbpedia.org/resource/Atlantic_bluefin_tuna (open)
• **Value:** Atlantic bluefin tuna

Fig. 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. 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`[13], 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[14], `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 paper we present one method to accomplish this scenario.

2.2. *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 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.

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 introduce the Open NEE Configuration Model (§3.5), and finally we present the extension of the Open Annotation Data Model (§3.6).

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[15]. Consider an infinite set U of RDF URI[16] references, an infinite set B of blank nodes[17] 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(Country) = {Afghanistan, Albania, Algeria, …}. Inversely, let $ctg(e) \in \mathcal{C}$ denote the category of an entity name (e.g. ctg(Algeria) = 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(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 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 §4.2.2).

3.2. *The Proposed Configuration Model*

Figure 2 shows the configuration model that we propose. Each `Category` has a name and can be associated with one or more `Knowledge Base Mirrors` (KBMs).

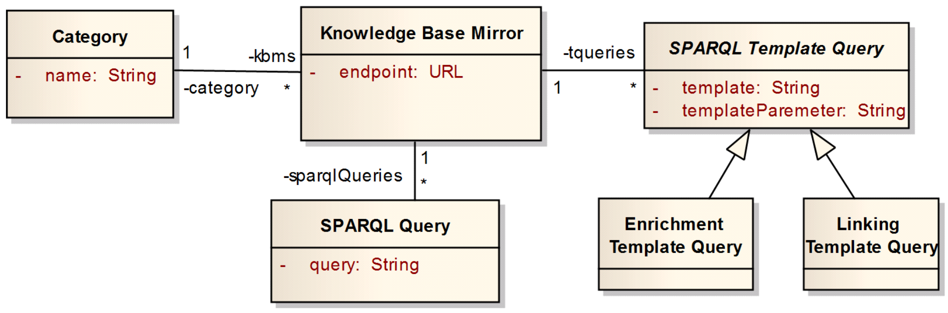


Fig. 2. A generic (abstract) model for configuring a NEE system.

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[18]).

The category Country is associated with one KBM:

- KBM_3 = `http://factforge.net/sparql` (SPARQL endpoint of FactForge[19]).

For the KBM_1, we can set the SPARQL query of Figure 3 for specifying the fish species of interest, or the one shown in Figure 4 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 }
```

Fig. 3. 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") }
```

Fig. 4. 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 5 which aims at returning URIs of type Fish whose label contains the name of an entity (ignoring case)[c]. 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 6, focusing on bigger precision.

For *Entity Enrichment*, KBM_1 can be associated with the template query shown in Figure 7 which retrieves the *outgoing* properties of a URI[d]. Notice 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)".

[c]The results of this task are shown in the pop-up window "Entity Exploration" in Figure 1.
[d]The retrieved triples are those shown if the user clicks the link of a resource in the pop-up window of Figure 1.

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

Fig. 5. 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]")) }
```

Fig. 6. 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 }
```

Fig. 7. SPARQL template query for retrieving the outgoing properties of resource.

By collecting the RDF triples that correspond to a set of entity URIs, we can form an RDF graph from which we can infer whether and how these entity URIs are connected. For example, Figure 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.

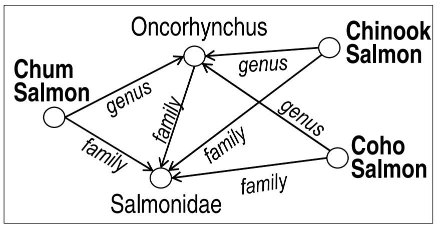


Fig. 8. An example of an RDF graph.

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[20]. 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 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 5 and 6). 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[6]), 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 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:

$$Graph(D,r) = \cup_{u \in U(D)} Descr(u,r) \tag{1}$$

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:

$$score(e) = \frac{count(e)}{\sum_{e' \in Ent(D)} count(e')} \tag{2}$$

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 paper.

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[21] 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:

$$sim(a, b) = \frac{\max(l(a), l(b)) - edt(a, b)}{\max(l(a), l(b))} \tag{3}$$

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

$$URIscore(e, u) = \max(sim(e, label(u)), sim(e, suffix(u))) \tag{4}$$

Instead of comparing the strings using Edit Distance, we can use the distance function proposed by Stoilos et al.[22], 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 $sim(e, label(u)) = 0$. In §5.3.3, we report indicative experimental results regarding the effectiveness of these ranking approaches in a specific domain.

3.5. *The Open NEE Configuration Model*

Here we introduce an RDF/S vocabulary for describing a configuration based on the proposed model. This vocabulary, apart from defining explicitly the semantics of a configuration, allows a NEE system to describe (and publish as Linked Data) its "entity mining" capabilities. It also allows to better handle the provenance of the outcome of a NEE process by explicitly describing the configuration used during the NEE process.

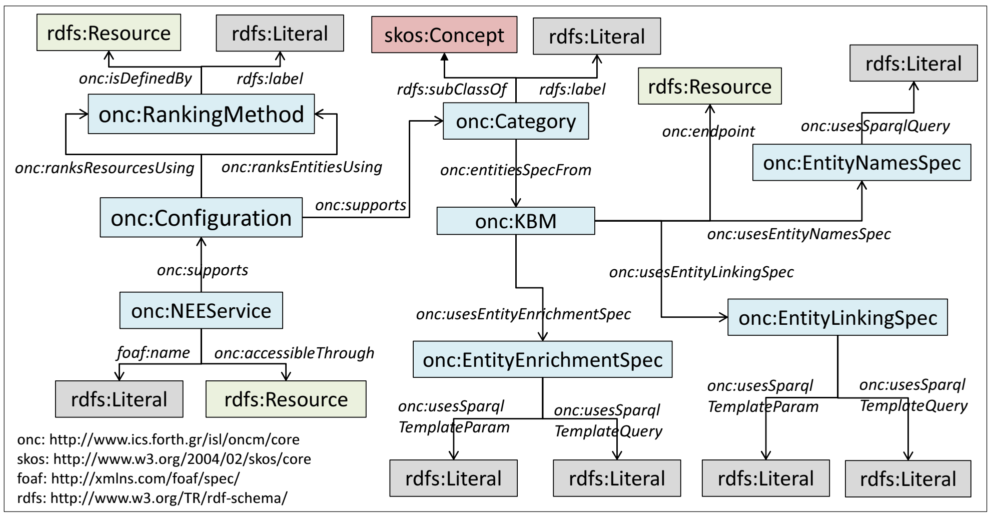


Fig. 9. The Open NEE Configuration Model.

Figure 9 depicts the proposed model, which we call "Open NEE Configuration Model". The vocabulary is accessible through: `http://www.ics.forth.gr/isl/oncm`. We have defined 8 classes and 13 properties (they are briefly described in Table 1). The model, apart from allowing the description of the supported categories and the KMBs, also enables specifying the method that it is used for ranking the detected entities and/or the matched resources. Although the model allows relating a configuration with a NEE service, we can describe a configuration without providing information about the service (i.e. without connecting an instance of `onc:Configuration` with an instance of `onc:NEEService`) because, for example, we want to create a general configuration which will be used for configuring one or more NEE services. Moreover, the model exploits the SKOS[23] vocabulary for indicating that `onc:Category` is subclass of `skos:Concept`. Thereby, we can interrelate the supported categories exploiting the SKOS properties. For example, we can define that the category Species is a broader concept of the categories Fish Species and

Table 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.

Bird Species. This information can be then exploited by an application for offering a more advanced visualization, e.g. by using nesting if a category is narrower that another, etc. We can also provide domain information by relating a category with concepts/classes from a widely used taxonomy or thesaurus. For instance, if there is a well-known thesaurus related to the marine domain, we can define that the category Fish Species is related to a concept of that thesaurus. We should also stress that we can exploit the provenance data model (PROV[24]) and include provenance information, e.g. who created a configuration, when, etc.

Figure 10 depicts an instantiation example of this model, while Figure 11 shows the corresponding RDF triples. In this example, the NEE system "X-Link" supports a configuration which can identify entities of type Fish Species, while for ranking the matched resources it uses the `Stoilos` distance function. We can also see the KBM that is used for linking, enriching and updating the entities of this category.

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 "Fish Species" (Figure 12).
- Give me all categories supported by the NEE service "X-Link" (Figure 13).
- Give me the SPARQL template queries (together with the endpoints) that are used for *linking* entities of the category "Fish Species" (Figure 14).
- Give me the NEE services supporting categories related to the marine domain (Figure 15).

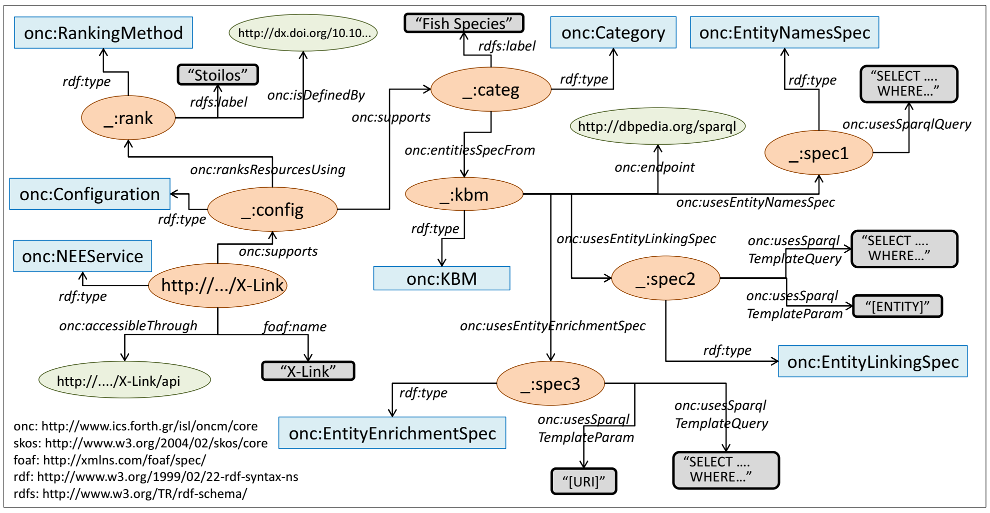


Fig. 10. Instantiation Example of the Open NEE Configuration Model.

```
PREFIX onc:  <http://www.ics.forth.gr/isl/oncm/core>
PREFIX skos: <http://www.w3.org/2004/02/skos/core>
PREFIX foaf: <http://xmlns.com/foaf/spec/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs: <http://www.w3.org/TR/rdf-schema/>

<http://.../X-Link>   rdf:type                        onc:NEEService ;
                      foaf:name                       "X-Link" ;
                      onc:accessibleThrough           <http://.../X-Link/api> ;
                      onc:supports                    _:config .

_:config              rdf:type                        onc:Configuration ;
                      onc:supports                    _:categ ;
                      onc:ranksResourcesUsing         _:rank .

_:rank                rdf:type                        onc:RankingMethod ;
                      rdfs:label                      "Similarity using Stoilos function." ;
                      onc:isDefinedBy                 <http://dx.doi.org/10.1007/11574620_45> .

_:categ               rdf:type                        onc:Category ;
                      rdfs:label                      "Fish Species" ;
                      onc:entitiesSpecFrom            _:kbm .

_:kbm                 rdf:type                        onc:KBM ;
                      onc:endpoint                    <http://dbpedia.org/sparql> ;
                      onc:usesEntityNamesSpec         _:spec1 ;
                      onc:usesEntityLinkingSpec       _:spec2 ;
                      onc:usesEntityEnrichmentSpec    _:spec3 .

_:spec1               rdf:type                        onc:EntityNamesSpec ;
                      onc:usesSparqlQuery             "SELECT ... FROM ..." .

_:spec2               rdf:type                        onc:EntityLinkingSpec ;
                      onc:usesSparqlTemplateQuery     "SELECT ... FROM ..." ;
                      onc:usesSparqlTemplateParam     "[ENTITY]" .

_:spec3               rdf:type                        onc:EntityEnrichmentSpec ;
                      onc:usesSparqlTemplateQuery     "SELECT ... FROM ..." ;
                      onc:usesSparqlTemplateParam     "[URI]" .
```

Fig. 11. Example of RDF triples describing a configuration.

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

Fig. 12. SPARQL query for retrieving the name and the URL of all services that support the category “Fish Species”.

```
SELECT ?name WHERE {
 <http://../X-Link> onc:supports ?config .
 ?config onc:supports ?categ . ?categ rdfs:label ?name }
```

Fig. 13. SPARQL query for retrieving the categories supported by a NEE system.

```
SELECT ?endpoint ?template WHERE {
  ?categ a onc:Category ; rdfs:label "Fish Species" ; onc:entitiesSpecFrom ?kbm .
  ?kbm onc:endpoint ?endpoint ; onc:usesEntityLinkingSpec ?linkspec .
  ?linkspec onc:usesSparqlTemplateQuery ?template }
```

Fig. 14. SPARQL query for retrieving the template queries and the corresponding endpoints that are used for linking entities of the category “Fish Species”.

```
SELECT ?tool ?name WHERE {
  ?tool a onc:NEEService ; foaf:name ?name ; onc:supports ?config .
  ?config onc:supports ?categ . ?categ skos:related example:marine }
```

Fig. 15. SPARQL query for retrieving the name and the URL of all services that support categories related to the marine domain.

3.6. *Exporting/Exchanging the Annotation Results*

It is often useful to know the *provenance* of a NEE process. Provenance also concerns the configuration under which an annotation was applied. Towards this direction, we propose an extension of the Open Annotation Data Model[11]. The Open Annotation Data Model specifies an RDF-based framework for creating associations (annotations) between related resources, allowing annotations to be easily shared between platforms. The extension model (which is an RDF/S vocabulary) is depicted in Figure 16 and comprises 1 new class, 8 new properties and 1 new instance.

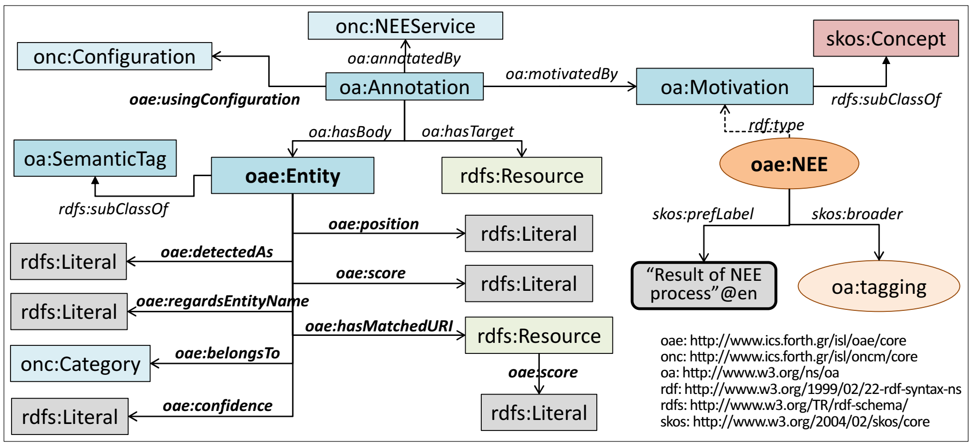


Fig. 16. The extension of the Open Annotation Data Model.

The class `oae:Entity` is subclass of `oa:SemanticTag` and is used for representing a detected entity. The instance `oae:NEE` is a `oa:Motivation` representing the result of a NEE process and can be considered narrower than the `oa:tagging` motivation. The property `oae:usingConfiguration` is used for associating the annotation process with a configuration. The property `oae:detectedAs` is used for representing the string in the document that was detected and considered an entity, while the property `oae:regardsEntityName` represents the actual entity name that exists in a gazetteer (text file containing a list of sorted entity names) of the NEE system. The property `oae:position` is used for representing the positions in the document in which the entity name was detected, `oae:score` represents the score of an entity or of an entity URI, while `oae:confidence` can represent the confidence of an ambiguous entity. The property `oae:belongsTo` is used for representing the category of the detected entity. Finally, the property `oae:hasMatchedURI` is used for representing the URIs that match an entity name.

The extension is available at `http://www.ics.forth.gr/isl/oae`. Figure 17 depicts an instantiation example in which the fish name "catfish" was detected in a Wikipedia page by the X-Link NEE system. Figure 18 depicts the RDF triples of a similar example. In this example, the NEE system detected two entities of type fish species. Each entity is accompanied by a matched URI which means that the entity linking process was applied. We notice also that the entity enrichment process was applied and two properties were retrieved from DBpedia (the properties `dbp-owl:order` and `dbp-owl:phylum`).

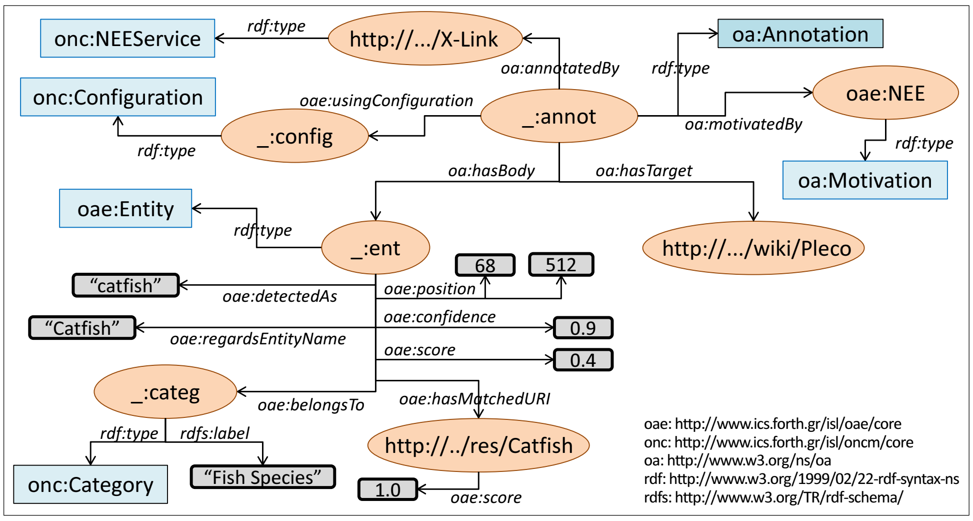


Fig. 17. Instantiation Example of the Open Annotation Extension.

```
PREFIX oae:     <http://www.ics.forth.gr/isl/oae/core>
PREFIX oa:      <http://www.w3.org/ns/oa>
PREFIX onc:     <http://www.ics.forth.gr/isl/oncm/core>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs:    <http://www.w3.org/TR/rdf-schema/>
PREFIX dbp-owl: <http://dbpedia.org/ontology/>
PREFIX dbp:     <http://dbpedia.org/resource/>

_:annot                    rdf:type                  oa:Annotation ;
                           oa:annotatedBy            <http://.../X-Link> ;
                           oa:motivatedBy            oae:NEE ;
                           oae:usingConfiguration    _:config ;
                           oa:hasBody                _:ent1 ;
                           oa:hasBody                _:ent2 ;
                           oa:hasTarget              <http://.../wiki/Pleco> .

_:ent1                     rdf:type                  oae:Entity ;
                           oae:detectedAs            "loricariidae" ;
                           oae:regardsEntityName     "Loricariidae" ;
                           oae:confidence            0.95 ;
                           oae:position              42, 115, 312 ;
                           oae:score                 0.6 ;
                           oae:belongsTo             _:categ ;
                           oae:hasMatchedURI         <http://.../Loricariidae> .

<http://../Loricariidae>   dbp-owl:order             dbp:Catfish ;
                           dbp-owl:phylum            dbp:Chordate ;
                           oae:score                 1.0 .

_:ent2                     rdf:type                  oae:Entity ;
                           oae:detectedAs            "catfish" ;
                           oae:regardsEntityName     "Catfish" ;
                           oae:confidence            0.9 ;
                           oae:position              68, 512 ;
                           oae:score                 0.4 ;
                           oae:belongsTo             _:categ ;
                           oae:hasMatchedURI        <http://../Catfish> .

<http://../Catfish>        dbp-owl:order             dbp:Ostariophysi ;
                           dbp-owl:phylum            dbp:Chordate ;
                           oae:score                 1.0 .

_:categ                    rdf:type                  onc:Category ;
                           rdfs:label                "Fish Species" .
```

Fig. 18. RDF triples describing the result of a NEE process using the Open Annotation Extension.

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 the fish species “Catfish” (Figure 19).
- Give me documents referring entities of type “Fish Species” (Figure 20).
- Give me documents containing information about fish species of phylum “Chordate” (Figure 21).
- Give me fish species of order “Ostariophysi” detected in the web page “http://example.html” (Figure 22).

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[25, 26].

```
SELECT ?doc WHERE { ?annot a oa:Annotation ; oa:hasTarget ?doc ; oa:hasBody ?ent .
  ?ent a oae:Entity ; oae:regardsEntityName "Catfish" ; oae:belongsTo ?cat .
  ?cat a onc:Category ; rdfs:label "Fish Species" }
```

Fig. 19. SPARQL query for retrieving the documents referring the fish species “Catfish”.

```
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:label "Fish Species" }
```

Fig. 20. SPARQL query for retrieving the documents referring entities of type “Fish Species”.

```
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 "Fish Species" .
  ?uri dbp-owl:phylum dbp:Chordate }
```

Fig. 21. SPARQL query for retrieving the documents containing information about fish species of phylum “Chordate”.

```
SELECT ?entName WHERE {
  ?annot a oa:Annotation ; oa:hasTarget <http://example.html> ; oa:hasBody ?ent .
  ?ent a oae:Entity ; oae:belongsTo ?cat .
  ?cat a onc:category ; rdfs:label "Fish Species" .
  ?ent oae:regardsEntityName ?entName ; oae:hasMatchedURI ?uri .
  ?uri dbp-owl:order dbp:Ostariophysi }
```

Fig. 22. SPARQL query for retrieving fish species of order “Ostariophysi” detected in the web page “http://example.html”.

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), and two applications that currently use `X-Link` (§4.4).

4.1. *Architecture*

`X-Link` is based on the Gate ANNIE[27,28] 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 23 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.

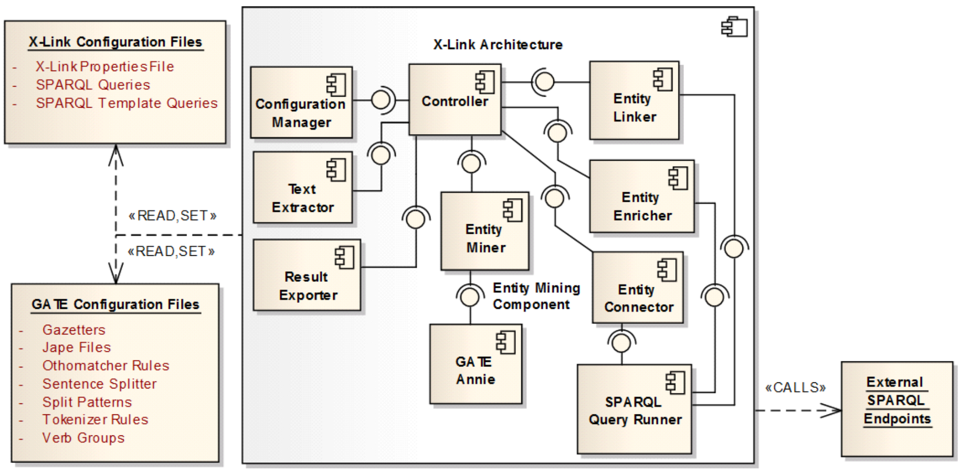


Fig. 23. The architecture of `X-Link`.

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 and Disambiguation*

`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 an entity in a category's gazetteer (using the Edit - Levenshtein - distance[21]). 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. However, although in that case 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.

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. This is the well-known entity disambiguation (or *word-sense disambiguation*) problem whose solution stills an open challenge[29]. Several approaches have been proposed in the literature, e.g. exploiting Wikipedia data[30, 31], using statistical methods[7], exploiting ontologies[32], or graph-based approaches[33, 34]. `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. This allows the application that uses it to disambiguate afterwards the identified entities, e.g. by exploiting context information or user feedback. For instance, in `Theophrastus` system[4] 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[35, 36] 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.

A thorough evaluation of word-sense disambiguation approaches that are appropriate for our setting is out of the scope of this paper but an important direction for future research.

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 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.[22]

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 24 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 $ConnectGraph(D, 1)$.

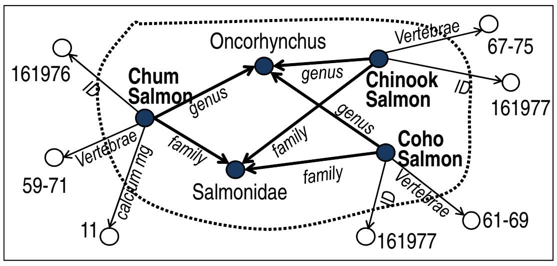


Fig. 24. An example of a *ConnectGraph*.

4.2.5. *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 §3.6) and supports the formats RDF/XML, N-Triples, and Notation3 (N3).

4.2.6. *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 25. 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 matching* is allowed with $p = 0.2$ (lines 11-12).

```
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
```

Fig. 25. A part of `X-Link`'s properties file configured for the marine domain.

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 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`[e] and `Theophrastus`[f].

`X-Search`[2, 37] 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`[g] project) and in *patent search* (in the context of the `PerFedPat`[h] 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[38]. 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[39].

`Theophrastus`[4] 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.

5. Evaluation

We first (§5.1) report the results of a *user study* that demonstrate the usability of `X-Link`. Then (§5.2), we report the results of a *case study* regarding the efficiency of the functions described in §3. Methods for achieving scalability are also discussed. Other aspects, as well as indicative experimental results regarding the effectiveness of the ranking approaches described in §3.4, are discussed in §5.3.

[e]`http://www.ics.forth.gr/isl/X-Search`
[f]`http://www.ics.forth.gr/isl/Theophrastus`
[g]`http://www.i-marine.eu/`
[h]`http://www.perfedpat.eu/`

5.1. *Task-based User Study*

The purpose of the user study is 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.

5.1.1. *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. Note that 11 participants are enough for revealing severe usability problems. Specifically, according to Robert Virzi[40], in a usability evaluation, 80% of the usability problems are detected with four or five subjects, additional subjects are less and less likely to reveal new information, while the most severe usability problems are likely to have been detected by the first few subjects. Furthermore, according to Laura Faulkner[41], at least 10 subjects are needed to reduce the risk of not revealing usability problems.

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

The tasks are based on the following scenario:

"Consider that you are the administrator of an application that can identify Fish *names (currently supporting only the English language) in Web pages. You have been asked to perform some changes. Specifically, by exploiting DBpedia, the application must also identify* European Countries *(T1) as well as fish names in Spanish (T2) (because the application will be used mainly by Spaniards). Also, the identified fishes must be linked with resources from DBpedia (T3) and must be enriched with all their outgoing properties (T4). Finally, in order to test that the system has been properly configured, perform entity mining in the Spanish version of Salmon's Wikipedia page and then inspect the connectivity of the identified entities (T5)"* ◇

We also provided the participants the following data:

- DBpedia's SPARQL endpoint (required for T1, T2, T3 and T4): `http://dbpedia.org/sparql`
- URI of the resource class `European Country` (required for T1): `http://dbpedia.org/class/yago/EuropeanCountries`
- Language code of Spanish (required for T2): `es`
- URI of the resource class `Fish Species` (required for T2 and T3): `http://dbpedia.org/ontology/Fish`
- URI of the property 'label' (required for T2 and T3): `http://www.w3.org/2000/01/rdf-schema#label`
- Spanish version of Tuna's Wikipedia page (required for T5): `http://es.wikipedia.org/wiki/Thunnus`

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 of the above scenario, as well as the time to successfully accomplish each task. In addition, at the end we asked them to complete a questionnaire. Specifically, they had to answer the following questions:

(Q0) How easy was to configure the system according to the scenario?
(Q1) How easy was to add the new category of entities?
(Q2) How easy was to update the existing category?
(Q3) How easy was to specify how to link the identified entities?
(Q4) How easy was to specify how to enrich the identified entities?
(Q5) How easy was to inspect the connectivity of the identified entities?
(Q6) What was difficult for you during the execution of the scenario?
(Q7) How familiar are you with SPARQL?

Regarding the questions Q0 to Q5, the user could select one of the following answers: `very easy`, `easy`, `normal`, `difficult`, `very difficult`, `impossible`. As regards the question Q6 the user could write free text, while for the question Q7 the user could select a value between 1 (*I don't know SPARQL*) and 5 (*I am expert in SPARQL*).

5.1.2. *Results*

Figure 26 (left) depicts the success rate of each task. 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 was also made evident by the responses in Q6).

Figure 26 (right) illustrates the average time for completing (successfully) each task. The reported times include the actual processing time, i.e. the time for running the corresponding SPARQL queries in tasks T1, T2 and T5, and the time for performing entity mining in T5. We notice that the most time consuming task was

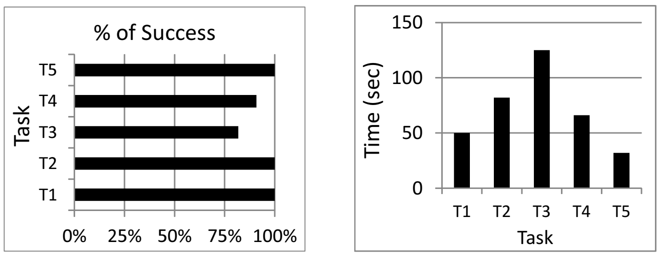


Fig. 26. Success rate (left) and average time (right) for completing each task (results from 11 users).

T3 which required about two minutes in average. This is a predictable result because T3 asked participants to construct (for first time) a SPARQL template query. In addition, the participants managed to totally configure the system according to the scenario (T1 to T4) in less than 6 minutes in average.

As regards the questionnaire, Table 2 depicts the results of the first 6 questions which refer to the difficulty in performing the tasks. None of the participants considered one of the tasks "difficult", "very difficult" or "impossible". 82% of the participants found the overall configuration (Q0) an "easy" task, while 18% found it "very easy". Regarding T3, which according to the success rates of Figure 26 was the most difficult task, 37% answered "normal", 45% answered "easy", while 27% answered "very easy". As regards T4, which was the second most difficult task according to the success rates, 27% answered "normal", 55% answered "easy", and 18% answered "very easy". Furthermore, all participants considered "very easy" the creation of a new category (T1). We notice that although some of the participants failed to complete T3 and T4, none of them found these tasks difficult. This is probably justified by the fact that during the evaluation, when a participant completed unsuccessfully a task we explained them their errors. Maybe, they then understood their errors and considered the task of "normal" difficulty.

Table 2. Evaluation of the difficulty in performing the scenario (results from 11 users).

Q	Very easy	Easy	Normal	Difficult	Very Difficult	Impossible
Q0	18%	82%	0%	0%	0%	0%
Q1	100%	0%	0%	0%	0%	0%
Q2	55%	27%	18%	0%	0%	0%
Q3	27%	45%	27%	0%	0%	0%
Q4	18%	55%	27%	0%	0%	0%
Q5	45%	45%	9%	0%	0%	0%

Regarding Q6, a few participants mentioned a difficulty in understanding the notion of the template queries (one also suggested to provide a user-friendly interface for constructing them). This can be justified by the fact that we did not explain it with many examples during the initial (5-minute) demonstration of `X-Link`. In addition, a participant commented that he/she would like to get informed with more details about the result of each action. For example, when updating a category it would be nice if the system reported the number of the added entity names. Finally, regarding Q7, 18% selected the answer "2", 36% selected "3", 36% selected

"4", and 9% selected "5", meaning that about half of the participants were not very experienced with SPARQL, however most of them managed to configure the system.

Synopsis. Concluding the above results, we can say that by adopting the LOD-based approach that we propose and understanding the notion of the SPARQL template queries, one can easily configure a NEE system within a few minutes. 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.

5.2. *Case Study: Querying Online DBpedia*

We 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.

For improving the accuracy of the results, and since we were querying an online KB at *real-time*, we repeated the experiments 20 times (specifically, about 2 times per day for 10 days) and here we report the average values (including the network's delay time). This case study can be also considered an evaluation of a publicly available KB, since we ran many queries at DBpedia's SPARQL endpoint. The experiments were carried out using an ordinary computer with processor Intel Core i7 @ 3.4Ghz CPU, 8GB RAM and running Windows 7 (64 bit). The implementation is in Java 1.7.[i]

5.2.1. *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 measure 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, like the query in Figure 3), and b) the time for reading the answer and creating the category in `X-Link`.

Figure 27 depicts the average times for each set of resource classes. As expected, the 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 see 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. A limitation regarding DBpedia's SPARQL endpoint is that it does not return more than 50,000 results at once,

[i]The data used in the experiments (queries, resource classes, entity names, URIs, etc.) are accessible at `http://www.ics.forth.gr/isl/X-Link/files/exper_data.zip`.

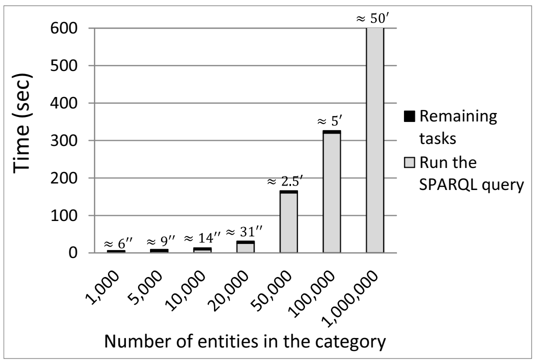


Fig. 27. Time for adding a new category.

thus we had to run multiple queries for resource classes with more than 50,000 entities (using SPARQL's `LIMIT` and `OFFSET`). For this reason, adding a category from DBpedia's endpoint with one million entities costs about 50 minutes. However note that this task is performed once (in a preprocessing step) or every time we want to update the entities of the corresponding category.

5.2.2. *Time for Linking an Identified Entity*

The time highly depends on the total number of entities belonging to the corresponding category. 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 shown in Figure 28 (`[URI_OF_RES_CLASS]` corresponds to the URI of the resource class, while `[ENTITY]` corresponds to the randomly selected label).

```
SELECT DISTINCT ?URI WHERE { ?URI rdf:type <[URI_OF_RES_CLASS]> .
 ?URI rdfs:label ?Name FILTER(regex(str(?Name),"[ENTITY]","i")) }
```

Fig. 28. The SPARQL template query used in the experiments for linking the entities with semantic resources.

Figure 29 depicts the average times. We notice 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.

We should stress here that, in some application scenarios, this functionality can be offered *on-demand*. For example, in the scenario of Figure 1, the user can request to inspect the semantic resources that match an entity by clicking the small icon next to the entity's name.

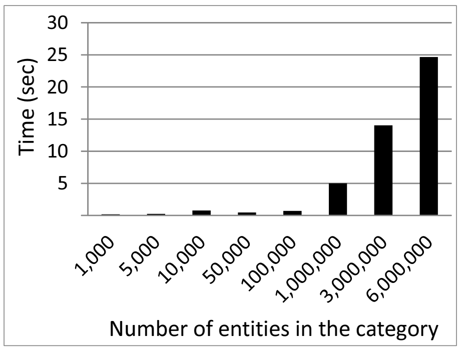


Fig. 29. Time for linking an identified entity.

5.2.3. *Time for Enriching an Entity URI*

The time highly depends on the properties that we retrieve. 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 expect that the time will be very low 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.

We randomly selected 160 URIs from DBpedia and measured the average time required for retrieving the properties. Figures 30-33 show the corresponding SPARQL queries that we ran for each type of properties, while Figure 34 depicts the results. As expected, the time is very low (less than 300 ms) for all types of properties. Like in the case of entity linking, in some application scenarios this functionality can be offered on-demand. For instance, in the example of Figure 1, the user can explore the properties of a resource by clicking on its URI.

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

Fig. 30. SPARQL query for retrieving the incoming properties of a URI.

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

Fig. 31. SPARQL query for retrieving the outgoing properties of a URI.

```
SELECT DISTINCT ?propertyName ?propertyValue
WHERE { { <[URI]> ?propertyName ?propertyValue FILTER(!isLiteral(?propertyValue)) }
UNION { <[URI]> ?propertyName ?propertyValue FILTER(lang(?propertyValue)="en") } }
```

Fig. 32. SPARQL query for retrieving the outgoing properties of a URI, filtered by language.

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

Fig. 33. SPARQL query for retrieving both the incoming and the outgoing properties of a URI.

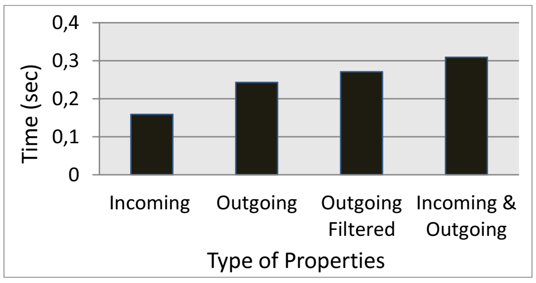


Fig. 34. Time for enriching an entity URI.

5.2.4. *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. We repeated the experiments for 5 different resource classes and we report the average values.

Figure 35 depicts the results. We notice 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. Nevertheless, this is often acceptable in professional systems and the users may desire to pay the cost. For example, persons working in *patent* offices spend many hours for a particular patent search request and the same is true in bibliographic and medical search. Of course, for $r = 1$ and in case we have already retrieved the properties of the entity URIs, the time will be very low.

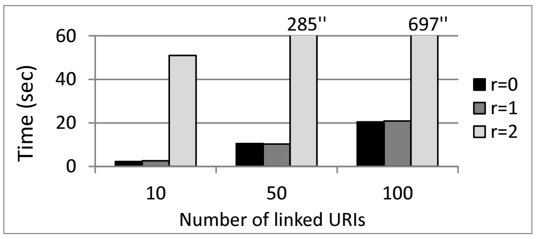


Fig. 35. Time for inspecting the connectivity of a set of entity URIs.

5.2.5. *Reliability and Scalability*

From our experimentation with LOD, we have noticed that the existing publicly available online KBs (like DBpedia) are not optimized for efficiency. The fact that everyone can query them affects their efficiency and availability (this is the reason for repeating the experiments many times). They also do not serve multiple concurrent requests in order to avoid overloading their systems. Nevertheless, the aforementioned experimental results showed that even if we query an online KB at real-time we can support the exploitation of LOD. This is very important since the "real-time" approach exploits the dynamic and "open" nature of LOD. In addition, we have seen that if an entity belongs to a category with millions of entities then the linking time can be high. The same is true in case the application requires to retrieve semantic information for numerous entities at once, i.e. when this functionality is not offered on-demand. In the near future, as such technologies mature and get used in applications, we expect that the aforementioned problems will be handled.

In the meantime, to increase the efficiency and the reliability of such services, we could adopt a *caching mechanism* or we could *index* a part of the underlying KB. Furthermore, as proposed by Umbrich et al.[42], we could keep a local copy of data that hardly changes and offer a hybrid query execution approach for improving the response time and reducing the load on the endpoints, while keeping the results fresh. Of course, the underlying KBs may not be publicly available, or a *dedicated Warehouse* can be constructed that will only serve a particular application, like the marineTLO-based warehouse[38] for the marine domain. The KBs (or the Warehouse) could also be *distributed* in many servers, taking advantage of load balancing techniques[43]. Such approaches can highly improve the performance and the served throughput, however with the cost of loosing the freshness of the results.

5.3. *Other Aspects*

5.3.1. *Formulation of SPARQL Queries*

There are many tools that can facilitate the construction of SPARQL queries, without requiring any advanced knowledge in SPARQL[44, 45]. Furthermore, there are natural language approaches that guide users in formulating queries in a language seemingly akin to English and translate them to SPARQL[46]. In this paper, we consider that the administrator of the underlying application knows the SPARQL query language.

5.3.2. *Effectiveness of NEE*

There are various papers that aim at evaluating the effectiveness of NEE tools[47–50]. 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 this paper 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 or disambiguation method (the proposed approach can be also applied by existing NEE systems). `X-Link` is a framework that realizes the proposed configuration model and it currently relies on Gate ANNIE. Therefore, the quality of the identified entities is out of the scope of this paper.

5.3.3. *Effectiveness of the URI Ranking Approaches*

In order to get a first feedback about the effectiveness of the URI ranking approaches described in §3.4, 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.[j] 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 5. Then, we computed the top-ranked URI using a) the Edit Distance function, b) the `Stoilos` function[22], 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.

[j]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`.

6. Related Work

There is a plethora of non LOD-based NEE tools like `Wikipedia Miner`[51], `Yahoo! Content Analysis API`[52], and `TagMe`[53]. Since the approach that we propose is based on LOD, below we first discuss the most relevant LOD-based NEE tools of general purpose (§6.1), we then report some semantic annotation systems tailored for the *life sciences* domain (§6.2), and finally we discuss the main differences of our approach (§6.3).

6.1. *LOD-based NEE Tools of General Purpose*

DBpedia Spotlight[6] 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[7] 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[k]). 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.
OpenCalais: Calais[8] 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[l]), Microformats, N3 or simple text. In addition, it supports automatic connection to the LOD.
AIDA[54] 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[55]. It accepts plain text, HMTL as well as semi-

[k]`http://rdf.alchemyapi.com/rdf/v1/s/aapi-schema#`
[l]`http://www.opencalais.com/files/owl.opencalais-4.3a.xml`

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[56] 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.[57]. 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[58] 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.

6.2. *Life Sciences-tailored Annotation Tools*

Domeo Annotation Toolkit[59] 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[60]. However, Domeo is currently being extended to also support the Open Annotation Data Model[11]. 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[61] 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**[62] 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[63] 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*.

6.3. *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:

- The proposed approach does not index semantic information (e.g. RDF triples or URIs); it 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.

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. To enable relating the output of a NEE process with an applied configuration, we have proposed an extension of the *Open Annotation Data Model*. By accessing the annotation results in this format, an application can offer advanced query services over the annotated data but also integrate external semantic information coming from the LOD.

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.).

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.

We evaluated the proposed approach in terms of *usability* and *feasibility*. As regards usability, we performed a task-based user study. The results showed that by adopting the proposed approach, one can configure a NEE system within a few minutes. In addition, the majority (80%) of the participants managed to successfully configure the system according to the specified scenario and also found it an easy task. Regarding feasibility, the results of a case study over online DBpedia demonstrated that even if we query a publicly available KB we can support the exploitation of LOD at real-time. We also discussed how we can achieve scalability which highly depends on the application context and the reliability of the underlying KBs. For example, querying a dedicated Warehouse which applies a load balancing technique, or adopting a hybrid query execution approach, can highly improve system's throughput and performance. Finally, we evaluated approaches for ranking the matched resources. The results showed that the similarity between the name of the detected entity and the label or the suffix of the matched URI can be efficiently used for finding the correct matched resource with precision more than 90%.

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. As regards `X-Link`, a future direction is to elaborate on methods for entity disambiguation that are appropriate for our setting, e.g. in cases where entity mining is based only on gazetteers.

Acknowledgments

We thankfully acknowledge the support of iMarine (FP7 Research Infrastructures, 2011-2014), NoE APARSEN (FP7, Proj. No 269977, 2011-2014) and MUMIA COST action (IC1002, 2010-2014).

References

1. D. Mollá, M. Van Zaanen and D. Smith, Named Entity Recognition for Question Answering, *Proceedings of ALTW* (2006) 51–58.
2. 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.
3. 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.
4. P. Fafalios and P. 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).
5. S. Kulkarni, A. Singh, G. Ramakrishnan and S. Chakrabarti, Collective Annotation of Wikipedia Entities in Web Text, in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* ACM2009.
6. P. N. Mendes, M. Jakob, A. García-Silva and C. Bizer, DBpedia Spotlight: Shedding Light on the Web of Documents, in *Proceedings of the 7th International Conference on Semantic Systems* ACM2011, pp. 1–8.
7. AlchemyAPI `http://www.alchemyapi.com/`.
8. OpenCalais, Thomson Reuters `http://www.opencalais.com/`.
9. C. Bizer, T. Heath and T. Berners-Lee, Linked Data - The Story so Far, *International Journal on Semantic Web and Information Systems* **5**(3) (2009).
10. RDF Schema 1.1 `http://www.w3.org/TR/rdf-schema/`.
11. R. Sanderson, P. Ciccarese and H. Van de Sompel, Open annotation data model, *W3C Community Draft* (2013).
12. SPARQL Query Language for RDF `http://www.w3.org/TR/rdf-sparql-query/`.
13. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, Dbpedia: A Nucleus for a Web of Open Data, in *The Semantic Web* (Springer, 2007)
14. Resource Description Framework (RDF) `http://www.w3.org/RDF/`.
15. SPARQL endpoint `http://semanticweb.org/wiki/SPARQL_endpoint`.

16. Universal Resource Identifier (URI) `http://www.w3.org/Addressing/URL/URI_Overview.html`.
17. D. Beckett and B. McBride, Rdf/xml syntax specification (revised), *W3C recommendation* **10** (2004).
18. FAO Fisheries Linked Open Data `http://www.fao.org/figis/flod/`.
19. B. Bishop, A. Kiryakov, D. Ognyanov, I. Peikov, Z. Tashev and R. Velkov, Factforge: A Fast Track to the Web of Data, *Semantic Web* **2**(2) (2011) 157–166.
20. SPARQL 1.1 Federated Query, W3C Recommendation, 21 March 2013 `http://www.w3.org/TR/sparql11-federated-query/`.
21. G. Navarro, A Guided Tour to Approximate String Matching, *ACM computing surveys (CSUR)* **33**(1) (2001) 31–88.
22. G. Stoilos, G. Stamou and S. Kollias, A string metric for ontology alignment, in *The Semantic Web–ISWC 2005* (Springer, 2005) pp. 624–637.
23. Simple Knowledge Organization System `http://www.w3.org/2004/02/skos/`.
24. W3C Provenance Data Model (PROV) `http://www.w3.org/TR/prov-overview/`.
25. S. Ferré and A. Hermann, Semantic search: Reconciling expressive querying and exploratory search, in *The Semantic Web–ISWC 2011* (Springer, 2011) pp. 177–192.
26. G. Joris and S. 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.
27. H. Cunningham, D. Maynard, K. Bontcheva and V. 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.
28. K. Bontcheva, V. Tablan, D. Maynard and H. Cunningham, Evolving GATE to Meet New Challenges in Language Engineering, *Natural Language Engineering* **10**(3-4) (2004) 349–373.
29. D. Carmel, M.-W. Chang, E. Gabrilovich, B.-J. P. Hsu and K. 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*, (ACM, 2014), pp. 1292–1292.
30. S. Cucerzan, Large-scale named entity disambiguation based on wikipedia data., in *EMNLP-CoNLL* **7**, Citeseer2007, pp. 708–716.
31. X. Han and J. Zhao, Named entity disambiguation by leveraging wikipedia semantic knowledge, in *Proceedings of the 18th ACM conference on Information and knowledge management* ACM2009, pp. 215–224.
32. J. Hassell, B. Aleman-Meza and I. B. Arpinar, *Ontology-driven automatic entity disambiguation in unstructured text* (Springer, 2006).
33. R. Usbeck, A.-C. N. Ngomo, M. Röder, D. Gerber, S. A. Coelho, S. Auer and A. Both, Agdistis - graph-based disambiguation of named entities using linked data, in *The Semantic Web–ISWC 2014* (Springer, 2014)
34. A. Moro, A. Raganato and R. Navigli, Entity linking meets word sense disambiguation: A unified approach, *Transactions of the Association for Computational Linguistics* **2** (2014).
35. 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)* IEEE, (Newport Beach, California, USA, 2014).
36. 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 Se-*

mantic Computing (2015).
37. P. Fafalios and Y. Tzitzikas, Exploratory Professional Search through Semantic Post-Analysis of Search Results, in *Professional Search in the Modern World Lecture Notes in Computer Science* **8830** (Springer, 2014) pp. 166–192.
38. 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.
39. 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.
40. R. A. Virzi, Refining the Test Phase of Usability Evaluation: How Many Subjects is Enough?, *Human Factors: The Journal of the Human Factors and Ergonomics Society* **34**(4) (1992) 457–468.
41. L. Faulkner, Beyond the Five-User Assumption: Benefits of Increased Sample Sizes in Usability Testing, *Behavior Research Methods, Instruments, & Computers* **35**(3) (2003) 379–383.
42. J. Umbrich, M. Karnstedt, A. Hogan and J. X. Parreira, Hybrid SPARQL Queries: Fresh vs. Fast Results, in *The Semantic Web–ISWC 2012* 2012 pp. 608–624.
43. V. Cardellini, M. Colajanni and P. S. Yu, Dynamic Load Balancing on Web-Server Systems, *Internet Computing, IEEE* **3**(3) (1999) 28–39.
44. O. Ambrus, K. Möller and S. 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.
45. A. Russell, P. R. Smart, D. Braines and N. R. Shadbolt, NITELIGHT: A Graphical Tool for Semantic Query Construction, in *Semantic Web User Interaction Workshop (SWUI 2008)*April 2008.
46. M. T. Enrico Franconi, Paolo Guagliardo, Quelo: a NL-based Intelligent Query Interface, in *2nd Workshop on Controlled Natural Languages (CNL 2010)*2010.
47. M. Gagnon, A. Zouaq and L. 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* International World Wide Web Conferences Steering Committee2013, pp. 1239–1246.
48. G. Rizzo and R. 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).
49. G. Rizzo and R. 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* Association for Computational Linguistics2012, pp. 73–76.
50. G. Rizzo, R. Troncy, S. Hellmann and M. Bruemmer, NERD meets NIF: Lifting NLP Extraction Results to the Linked Data Cloud, *LDOW* **937** (2012).
51. D. Milne and I. H. Witten, An open-source toolkit for mining wikipedia, *Artificial Intelligence* **194** (2013) 222–239.
52. Yahoo! content analysis api `http://developer.yahoo.com/contentanalysis/`.
53. P. Ferragina and U. Scaiella, Fast and accurate annotation of short texts with wikipedia pages, *arXiv preprint arXiv:1006.3498* (2010).
54. M. A. Yosef, J. Hoffart, I. Bordino, M. Spaniol and G. Weikum, AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables, *Proceedings of the VLDB Endowment* **4**(12) (2011) 1450–1453.

55. J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. De Melo and G. 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* ACM2011, pp. 229–232.
56. Wikimeta `http://www.wikimeta.com/`.
57. E. Charton, M. Gagnon and B. Ozell, Automatic Semantic Web Annotation of Named Entities, in *Advances in Artificial Intelligence* (Springer, 2011) pp. 74–85.
58. Lupedia Enrichment Service, Ontotext `http://lupedia.ontotext.com/`.
59. P. Ciccarese, M. Ocana and T. Clark, Open semantic annotation of scientific publications using domeo, *Journal of biomedical semantics* **3** (2012) 1–14.
60. P. Ciccarese, M. Ocana, L. J. G. Castro, S. Das and T. Clark, An open annotation ontology for science on web 3.0, *Journal of Biomedical Semantics* **2** (2011).
61. T. K. Attwood, D. B. Kell, P. McDermott, J. Marsh, S. Pettifer and D. Thorne, Utopia documents: linking scholarly literature with research data, *Bioinformatics* **26**(18) (2010) i568–i574.
62. C. Jonquet, N. Shah, C. Youn, C. Callendar, M.-A. Storey and M. Musen, Ncbo annotator: semantic annotation of biomedical data, in *International Semantic Web Conference*2009.
63. D. Rebholz-Schuhmann, M. Arregui, S. Gaudan, H. Kirsch and A. Jimeno, Text processing through web services: calling whatizit, *Bioinformatics* **24**(2) (2008).