# Configuring Named Entity Extraction through Real-Time Exploitation of Linked Data

Pavlos Fafalios, Manolis Baritakis and Yannis Tzitzikas
Institute of Computer Science, FORTH-ICS, GREECE, and
Computer Science Department, University of Crete, GREECE
{fafalios|mbaritak|tzitzik}@ics.forth.gr

## ABSTRACT

Named Entity Extraction is the process of identifying entities (like persons, locations, organizations, etc.) in texts and linking them to related semantic resources. This task is useful in several applications, e.g. for question answering, annotating documents, post-processing of search results, etc. However, existing named entity extraction 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 updating an existing category with additional entities, 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 named entity extraction system. We also present `X-Link`, a fully configurable named entity extraction tool 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 (on-line) 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 configured for different contexts, for building domain-specific applications (e.g. for identifying *drugs* in a medical search system or for annotating and exploring *fish species* in a marine-related web page). To test the approach, we conducted a task-based evaluation with users that demonstrates the *usability* of the proposed approach, and a case study that demonstrates its *feasibility*.

## Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; I.7 [**Document and Text Processing**]: Miscellaneous

## General Terms

Design, Management, Experimentation

## Keywords

Named Entity Extraction, Linked Data, Entity Mining

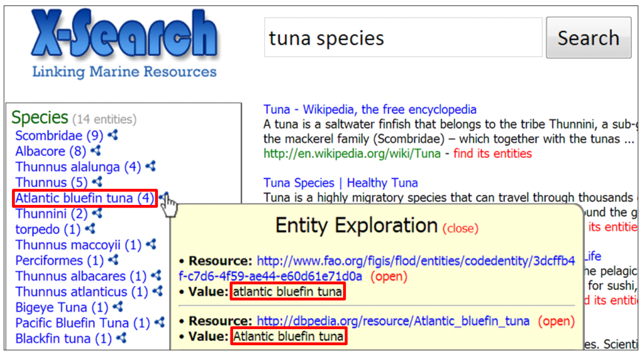
## 1. INTRODUCTION

Named Entity Extraction (NEE), also referred as *semantic annotation*, is the process of identifying entities in texts and linking them to relevant semantic resources. NEE often consists of two main sub-processes: *named entity recognition* (NER) which is the task of identifying entities belonging to a set of class labels (such as Person, Location, Organization, etc.), and *entity linking* which tries to link a named entity with a resource in a Knowledge Base. NEE is useful in several tasks, e.g. for question answering [24], post-processing of search results [16, 18], annotating (Web) documents [22]. 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 automatic NEE tools.

There are already several tools that support NEE, e.g. DBpedia Spotlight [23], AlchemyAPI [1] and OpenCalais [4]. 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 Knowledge Base. Hence, it is quite difficult to configure them for building domain specific applications.

Since a lot of information about *named entities* is already available as Linked Open Data (LOD) [10], 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 Knowledge Base 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 Knowledge Base appears which contains plenty of information for entities belonging to X. It would be useful if one could somehow "plug" the new Knowledge Base in a NEE system (with the less possible effort), enabling thereby the linkage of the identified entities with resources in the new Knowledge Base.

Furthermore, 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.


X-Search
Linking Marine Resources

tuna species
Search

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

**Figure 1: Semantic post-processing of search results (for the query *tuna species*) and exploration of the entity *Atlantic bluefin tuna* in X-Search.**

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.
- We propose a *generic model* for configuring a LOD-based NEE system.
- We present `X-Link`, a fully configurable (LOD-based) NEE tool that we have designed and implemented.
- We report the results of a task-based *user study* that demonstrates the *usability* of the proposed approach.
- We report the results of a *case study* that demonstrate the *feasibility* of the proposed approach, and we also discuss how we can achieve scalability.

The motivation for enhancing *configurability* can be made evident from the following scenario, which is a real scenario related to the ongoing `iMarine` project[1]:

*Semantic post-processing of search results: 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 NEE tool for identifying (at real time) Fish Species in the snippets or the full contents of the top results. You think that it would also be useful to link (on demand) the identified species with related semantic resources, as well as, retrieve more information (e.g. a short description of the species, an image, its taxonomy, etc.) by querying (at real-time) online Semantic Knowledge Bases. Figure 1 depicts a screen shot of X-Search for the query "tuna species". The user can see (in a left bar) the fish species that have been identified in the search results, and can also explore an identified species at real-time (the species "Atlantic bluefin tuna" in this example).* ◇

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 to link and enrich the identified species with resources from different sources; one may want images from DBpedia [8], 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 Knowledge Base, etc.).

In this paper we present one method to accomplish this scenario. The rest of this paper is organized as follows: Section 2 discusses related works. Section 3 analyzes how we can exploit the LOD for configuring a NEE system and proposes a generic configuration model. Section 4 describes `X-Link`, a fully configurable (LOD-based) NEE system, that supports the model described in Section 3. Section 5 reports the results of a task-based evaluation and of a case study. Finally, Section 6 concludes and identifies directions for future research.

## 2. RELATED WORKS AND SYSTEMS

Below we initially discuss the most relevant, LOD-based, NEE tools and systems, and then we report the main differences of our approach.

**DBpedia Spotlight** [23] 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 with the text content [23], and links them with URIs from DBpedia [8]. The results of entity extraction 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 Knowledge Base), update a category or specify how to link and enrich the identified entities.

**AlchemyAPI** [1] is a Natural Language Processing (NLP) service which provides a scalable platform for analyzing web pages, documents and tweets along with APIs for integration. The retrieved entities are ranked based on their importance in the given text and the results can be stored as JSON, Microformats, XML and RDF. 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 [4] is a toolkit that allows incorporating state-of-the-art 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,

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

Microformats, N3 or simple format text. In addition, it supports automatic connection to the LOD.

**AIDA** [33] is a framework and online tool for entity detection and disambiguation. Given a natural-language text, AIDA maps mentions of ambiguous names onto entities registered in the YAGO2 Knowledge Base [21]. It accepts plain text, HMTL as well as semi-structured inputs like tables, lists, or short XML files. AIDA is centered around collective disambiguation exploiting the prominence of entities, similarity between the context of the mention and its candidates, and the coherence among candidate entities for all mentions.

**Wikimeta** [6] 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 [13]. Wikimeta API is compliant with REST and the responses are formatted in XML and JSON. The datasets used to train the NLP tools of Wikimeta are derived from Wikipedia.

**Lupedia** [3] uses a gazetteer which is a list of surface forms that are 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 Knowledge Base. The results can be stored in HTML, JSON, RDF or XML.

## 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 (i.e. the supported categories of entities), as well as for linking and enriching 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 NEE 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); 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, existing systems return only the corresponding URIs and maybe some related Web pages.

## 3. THE PROPOSED APPROACH

At first we introduce 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) and describe the semantics of such configurations (§3.4).

### 3.1 Notions and Notations

Let $\mathcal{C}$ be a set of *entity categories*, e.g. $\mathcal{C} = \{\textsf{Fish Species}, \textsf{Country}, \textsf{Water Area}\}$ 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. $E(\textsf{Country}) = \{\textsf{Afghanistan}, \textsf{Albania}, \textsf{Algeria}, \ldots\}$. Inversely, let $ctg(e) \in \mathcal{C}$ denote the category of an entity name (e.g. $ctg(\textsf{Algeria}) = \textsf{Country}$). For an entity name $e$, let $U(e)$ denote the URIs that are related to entity $e$, e.g. $U(\textsf{Chum Salmon}) = \{$`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 of the form (`s`, `p`, `o`) (where `s` is the subject, `p` the predicate and `o` the object) that express information about $u$.

For an input document, say *doc*, we define as $Ent(doc, c)$ the set of entities identified in *doc* 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 the $D$,
- $U(D) = \cup_{e \in Ent(D)} U(e)$ 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, e.g. one entity *argentina* of type Country and one entity *argentina* of type Fish Genus. Therefore, each of these entities will have one category (i.e. $|ctg(e)| = 1$). This choice enables to apply afterwards disambiguation methods.

### 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).

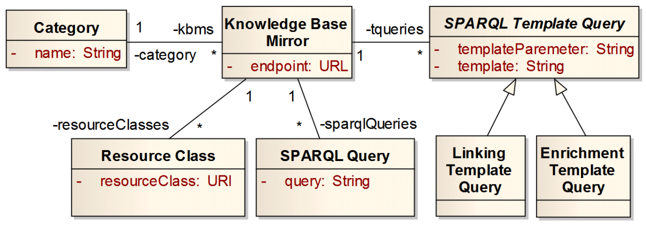


**Figure 2: A generic model for configuring a NEE system.**

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

The elements of type (a) are used for specifying the *entity names* of interest by providing their RDF Class in the KBM.

The elements of type (b) have the same objective, but instead of specifying an RDF class, a SPARQL query is provided.

The elements of type (c) allow specifying how entity names correspond to *entity URIs*, by providing a KBM-answerable SPARQL query.

The elements of type (d) 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 $\mathcal{C} = \{$Fish Species, Country$\}$. The category Fish Species is associated with two KBMs:

- $\text{KBM}_1$ = `http://dbpedia.org/sparql` (DBPedia's SPARQL endpoint)
- $\text{KBM}_2$ = `http://www.fao.org/figis/flod/endpoint` (SPARQL endpoint of FAO FLOD [2])

The category Country is associated with one KBM:

- $\text{KBM}_3$ = `http://factforge.net/sparql` (FactForge's [9] SPARQL endpoint)

For the $\text{KBM}_1$, we can set the *resource class* `http://dbpedia.org/ontology/Fish` (which actually corresponds to the query of Figure 3), or the *SPARQL query* 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> .
  ?uri rdfs:label ?label }
```

**Figure 3: Example of a SPARQL query for retrieving a list of fish names from DBpedia for a given resource class.**

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

**Figure 4: Example of a SPARQL query for retrieving a list of English fish names from DBpedia.**

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

**Figure 5: Example of a SPARQL template query for linking an identified Fish name with resources in DBpedia.**

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

**Figure 6: Example of a SPARQL template query for retrieving the outgoing properties of resource.**

For *Entity Linking*, $\text{KBM}_1$ 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)[2]. 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 following URI: `http://dbpedia.org/resource/Chum_salmon`.

For *Entity Enrichment*, $\text{KBM}_1$ can be associated with the template query shown in Figure 6 which retrieves the *outgoing* properties of a URI[3]. 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 the following:

```
SUBJECT: http://dbpedia.org/resource/Chum_salmon
PREDICATE: http://dbpedia.org/ontology/genus
OBJECT: http://dbpedia.org/resource/Oncorhynchus
```

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 7 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.

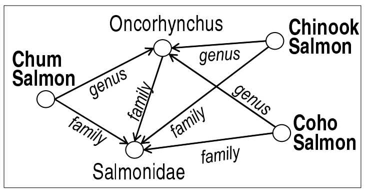


**Figure 7: An example of an RDF graph.**

Analogously, one can specify Resource Classes, SPARQL Queries and SPARQL 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 [5]. 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 defined 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).

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 $G$ the RDF graph of the underlying Knowledge Base. We define $In(S)$ and $Out(S)$ as follows:

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

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

[3]The retrieved triples are those shown if the user clicks the link of a resource in the pop-up window of Figure 1.

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:

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

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

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

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

## 4. THE SYSTEM X-LINK

X-Link is a LOD-based NEE tool that we have designed and implemented which realizes the functionality and the configuration model described in the previous sections. Below, we describe its architecture (§4.1), its functionality (§4.2), ways to use it (§4.3), and the supported configurability (§4.4).

### 4.1 Architecture

X-Link is based on the Gate ANNIE system and supports both gazetteers and NLP functions. Gate ANNIE [14, 11] 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.

Figure 8 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 (using the SPARQL Query Runner component) external SPARQL endpoints. Finally, the results are exported using the Result Exporter component.

### 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 Cleaning and Entity Mining

X-Link at first reads the contents of the requested document and performs a "cleaning" task, i.e. it removes useless text. For example, it removes HTML tags in a Web page or Meta elements in a Microsoft Word file.

After cleaning, X-Link applies NEE over the cleaned content. Currently, X-Link does not apply any disambiguation method, meaning that if an entity name exists in two supported categories, then this entity is returned twice, one for each supported category. For instance, consider that the supported categories are two: Country and Fish Species, and that the entity name "argentina" exists in both categories. If the document that we analyze contains the string "argentina" then the entities that match this string (and which are returned) are two: argentina (of type Fish Species) and argentina (of type Country). This allows the underlying application to disambiguate afterwards the identified entities, e.g. by exploiting the $Graph(D)$ or context information.

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 [25]). The allowed edit distance value depends on the length of the matching 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.

#### 4.2.3 Entity Linking

As regards the *entity linking* process, for each detected entity X-Link returns all the matching URIs and lets the underlying application 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.

An issue here is how to rank the URIs in case they are numerous, i.e. for an entity name $e$, how to rank the elements in $U(e)$. For example, for the entity name "salmon", and the template query of Figure 5, DBpedia returns 16 URIs that correspond to several species in the salmon family. However, an application may want to select only one URI (i.e. the top ranked) which must be the most relevant to the identified entity (this is also related to the *entity disambiguation* problem). Various methods can be applied. For the time being, X-Link returns all the matching URIs and lets the underlying application to define the desired ranking or selection criteria.

#### 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 which 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 template queries or by selecting to retrieve one of the following (common) types of properties: a) outgoing (the entity URI is the subject in the RDF triple), b) incoming (the entity URI 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.

As regards the connectivity of the entity URIs, X-Link

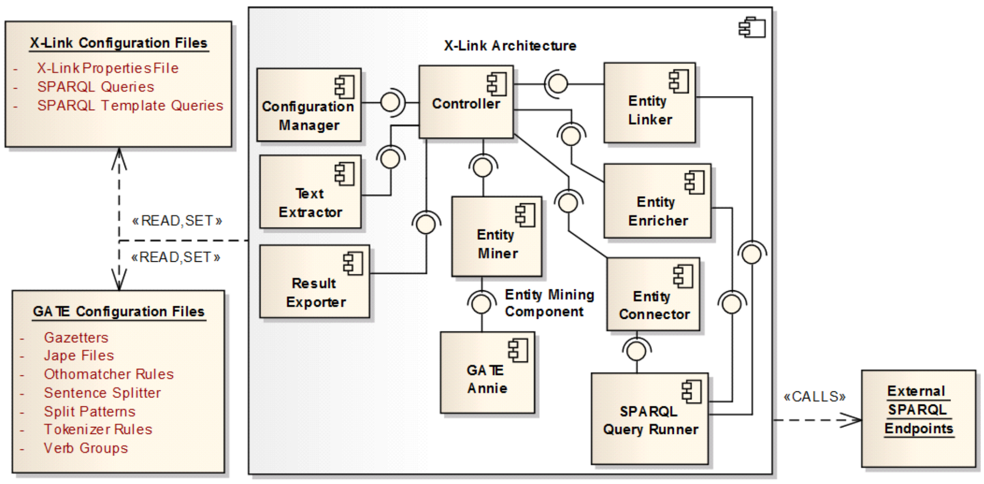


**Figure 8: The architecture of X-Link.**

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 2 URIs are omitted).

Figure 9 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* which are in bold (for simplicity, we have omitted the namespaces). The graph enclosed in the dashed shape, containing the black nodes, is the $ConnectGraph(D, 1)$.

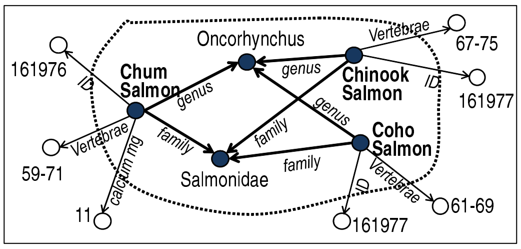


**Figure 9: An example of a** $ConnectGraph$**.**

### 4.2.5 *Output*

Currently, `X-Link` exports the results in XML and CSV.

## 4.3 Ways to Use

`X-Link`[4] 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.

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

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 underlying application.

## 4.4 Configurability

`X-Link` supports the configuration model described in Section 3 in two ways: (a) it can read such a configuration from a *properties file*, and (b) it offers a configuration API. The last can be used in a preprocessing step or even while a corresponding service is running.

### 4.4.1 *File-based Configuration*

An indicative part of the properties file (configured for the *marine* domain) is shown in Figure 10. 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 (*Disease*, *Drug*, *Protein* and *Chemical Substance*) are inactive. The set of *active categories* allows us to define which of the supported categories are interesting for an application, so `X-Link` can identify only entities of these categories. The category Fish uses one KBM (line 3), which is actually the SPARQL endpoint of DBpedia (line 4), and for the update of the category `X-Link` can use 2 resource classes (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).

### 4.4.2 *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, one or more resource classes and/or one or more SPARQL queries).
- Update an existing category (using one or more lists of entities, one or more resource classes and/or one or more SPARQL queries).

```
xlink.categories.supported = Fish;Country;Water_Area;Disease;Drug;Protein;Chemical_Substance
xlink.categories.active = Fish;Country;Water_Area
xlink.categories.Fish.kbms = dbpedia_fish
xlink.categories.Fish.kbms.dbpedia_fish.endpoint = http://dbpedia.org/sparql
xlink.categories.Fish.kbms.dbpedia_fish.resourceclasses = http://dbpedia.org/ontology/Fish;http://umbel.org/umbel/rc/Fish
xlink.categories.Fish.kbms.dbpedia_fish.templatequeries.linking = C:/xlink/templates/dbpedia_fish_linking.sparql
xlink.categories.Fish.kbms.dbpedia_fish.templatequeries.linking.parameter = [ENTITY]
xlink.categories.Fish.kbms.dbpedia_fish.templatequeries.enriching = C:/xlink/templates/dbpedia_fish_enriching.sparql
xlink.categories.Fish.kbms.dbpedia_fish.templatequeries.enriching.parameter = [URI]
xlink.connect.radius = 1
xlink.fuzzy = true
xlink.fuzzy.value = 0.2
```

**Figure 10: A part of `X-Link`'s properties file configured for the marine domain.**

- Remove a category.
- Change the displayed name of a category (i.e. rename).
- Set/change the KBMs of a category.
- Set/change the resource classes, the SPARQL queries and the SPARQL 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 $p$.

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.

#### 4.4.3 *Portability of Configurations*

The configurations can be exchanged. For instance, 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 the supported categories and on the number of the 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 of entities and several files required by Gate ANNIE. We should note here 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.

## 4.5 Current Applications of X-Link

`X-Link` library is currently used by the X-Search system[5] [16, 17] in two different contexts: in the *marine* domain (in the context of the iMarine[6] project) and in *Patent Search* (in the context of the `PerFedPat`[7] project). X-Search is a meta-search engine that reads the description of an underlying search source, queries that source, analyzes the returned results in various ways and also exploits the availability of semantic repositories. For instance, in iMarine, `X-Link` has been configured to identify Fish Species, Water Areas, Countries and Regional Fisheries Bodies, while the Knowledge Base that is exploited is the MarineTLO-based Warehouse [30].

## 5. EVALUATION

Here 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 are discussed in brief in §5.3.

## 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 system. 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. In addition, the administrator of the system can change the configuration through an administration page. Specifically, the administrator 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 also they have a basic knowledge of Linked Data and the SPARQL query language. Note that 11 participants are enough for revealing sever usability problems. Specifically, according to [32], 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 sever usability problems are likely to have been detected in the first few subjects. Furthermore, according to [19], 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

[5]`http://wiki.i-marine.eu/index.php/XSearch`
[6]`http:/www.i-marine.eu/`
[7]`http://www.perfedpat.eu/`

**(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 to 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 11 depicts the success rate for each task. All participants managed to complete the tasks T1, T2 and T5. However, 18% of the participants (in particular 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 inside the query and how it is used for constructing the template query (this was also made evident by the responses in Q6).

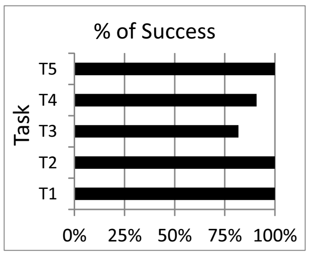


**Figure 11: Success rate for each task (results from 11 users).**

Figure 12 illustrates the average time for completing (successfully) each task. The most time consuming task was 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).

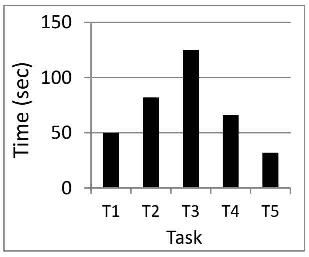


**Figure 12: Average time for completing each task.**

As regards the questionnaire, Table 1 depicts the results of the first 6 questions which correspond 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 11 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).

Regarding Q6, a few participants mentioned a difficulty in understanding the notion of the SPARQL 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 (e.g. when updating a category it would be nice if the system reported the number of the added entity names).

**Table 1: 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% |

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 DBpedia as the underlying Knowledge Base and we measured the time for:

1) Creating a new category
2) Linking an identified entity with semantic resources
3) Enriching an entity URI
4) Inferring the connectivity of the entity URIs

For improving the accuracy of the results, and since we were querying an *online* Knowledge Base, we repeated the experiments 20 times (specifically, about 2 times per day for 10 days) and here we report the average values[8]. This case study can be also considered an evaluation of a publicly available Knowledge Base, since we run 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.

### *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 13 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

[8]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`.

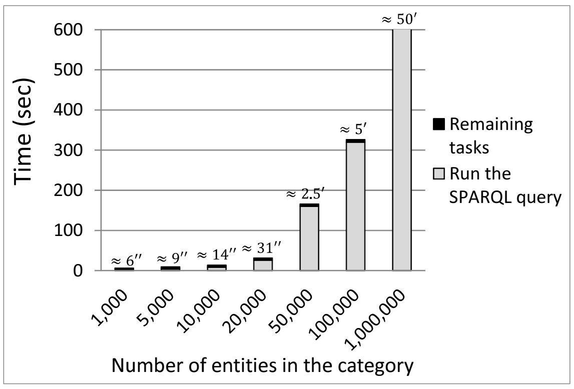


**Figure 13: Time for adding a new category.**

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

**Figure 14: The SPARQL template query used in the experiments for linking the entities with semantic resources.**

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, so 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 14 (`[URI_OF_ RES_CLASS]` corresponds to the URI of the resource class, while `[ENTITY]` corresponds to the randomly selected label).

Figure 15 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 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.

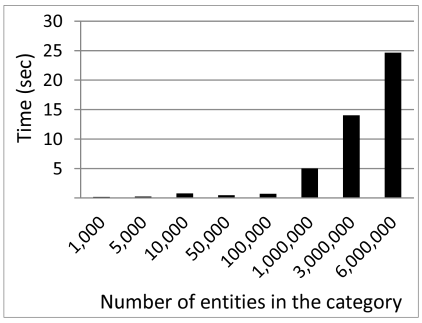


**Figure 15: Time for linking an identified entity.**

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

**Figure 16: Query for retrieving the incoming properties of a URI.**

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

**Figure 17: 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') } }
```

**Figure 18: Query for retrieving the outgoing properties of a URI, filtered by language.**

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

**Figure 19: Query for retrieving both the incoming and the outgoing properties of a URI.**

### 5.2.3 *Time for Enriching an Entity URI*

The time highly depends on the properties that we want to retrieve. We run experiments for the following types of properties: i) incoming, ii) outgoing, iii) outgoing of a specific language, and iv) union of incoming and outgoing properties. We expect that the time will be very low since the URI for which we want to retrieve properties is known (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 16-19 show the corresponding SPARQL queries that we run for each type of properties, while Figure 20 depicts the results. As expected, the time is very low for all types of properties (less than 300 ms).

Like in the case of entity linking, in some application scenario 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 the desired resource in the pop-up window.

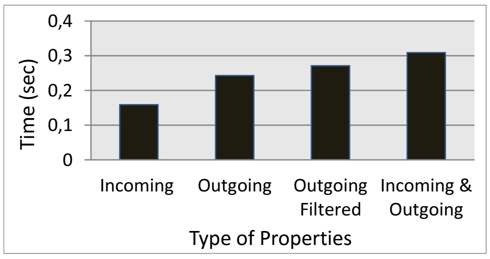


**Figure 20: Time for enriching an identified entity.**

### 5.2.4 *Time for Inspecting the Connectivity of the Entity URIs*

We run indicative 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 run 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 21 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 and is increased exponential to the number of URIs (e.g. for 100 URIs about 12 minutes are required). This is a predictable result since each URI may contain many related URIs. Nevertheless, this is often acceptable in professional search and the users may desire to pay the cost. For example, persons working in *patent* offices spend many hours for a particular patent search 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.

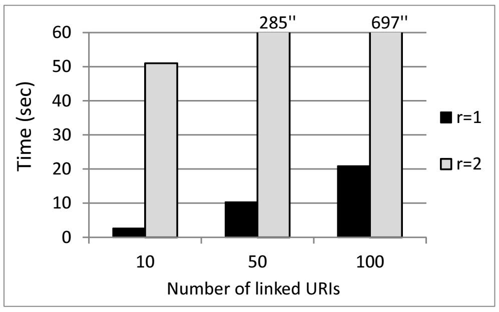


**Figure 21: Time for inspecting the connectivity.**

### 5.2.5 *Reliability and Scalability*

From our experimentation with LOD, we have noticed that the existing publicly available online Knowledge Bases (like DBpedia) are not reliable since they mainly serve demonstration purposes. 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 Knowledge Base at real-time we can support the exploitation of LOD. This is very important since it 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 underlying application requires to retrieve semantic information for numerous entities at once, i.e. when this functionality is not offered on-demand. In such cases, adopting a *caching mechanism* or *indexing* a part of the underlying Knowledge Base (with the cost of loosing the freshness of the results) will highly improve the response times and the throughput that can be served.

Of course, in a real application the underlying Knowledge Bases may not be publicly available, or a *dedicated Warehouse* can be constructed that will only serve a particular application (like the marineTLO-based warehouse [30] for the marine domain). The Knowledge Bases (or the Warehouse) could also be *distributed* in many servers, so the system can apply a load balancing technique [12] for serving the requests. Furthermore, as proposed in [31], 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. All the above can highly improve the performance and the scalability of the proposed approach.

## 5.3 Other Aspects

*Formulation of SPARQL Queries.* There are many tools that can facilitate the construction of the SPARQL queries, without requiring any advanced knowledge in SPARQL (like [7] and [29]). Furthermore, there are natural language approaches (e.g. [15]) that guide users in formulating queries in a language seemingly akin to English and translate them to SPARQL. In this paper, we consider that the administrator of the underlying application knows the SPARQL query language.

*Effectiveness of the NEE Tool.* There are various papers that aim at evaluating the effectiveness of NEE tools, e.g. [20], [26], [27] and [28]. The effectiveness of `X-Link` (which currently does not apply any disambiguation method) highly depends on how the user/developer has configured it (i.e. on the completeness of the specified categories, the quality of the underlying Knowledge Bases, the given 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 algorithm or disambiguation method. `X-Link` currently relies on Gate ANNIE, but that can be changed. Therefore, the quality of the identified entities is out of the scope of this paper.

## 6. CONCLUSION

We have proposed a method that exploits Linked Data for *configuring* (dynamically and handily) a Named Entity Extraction system. For tackling the configuration requirements, we have defined a generic configuration model and we have presented the design and implementation of `X-Link`, a fully configurable (LOD-based) NEE tool that supports this 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 Knowledge Bases. This enhanced configurability allows `X-Link` to be used for building and dynamically configuring domain-specific applications. 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. In addition, `X-Link` can infer if and how the identified entities are associated.

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 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 DBpedia demonstrated that even if we query a publicly available Knowledge Base 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 Knowledge Bases). 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.

Regarding future work and research, there are several aspects that are worth investigating. One is to elaborate on methods for ranking the matching URIs in case they are numerous. Another related issue is to evaluate approaches for entity disambiguation that are appropriate for our setting (i.e. cases in which an identified entity belongs to more than one supported categories).

## Acknowledgments

We thankfully acknowledge the support of the *iMarine* (FP7 Research Infrastructures, 2011-2014) and *MUMIA* COST action (IC1002, 2010-2014). We would also like to thank the members of the Information Systems Laboratory at FORTH-ICS who willingly took part in the user study.

## 7. REFERENCES

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

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

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

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

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

[6] Wikimeta. `http://www.wikimeta.com/`.

[7] 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.

[8] 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*, pages 722–735. Springer, 2007.

[9] 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):157–166, 2011.

[10] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story so Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[11] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3-4):349–373, 2004.

[12] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic Load Balancing on Web-Server Systems. *Internet Computing, IEEE*, 3(3):28–39, 1999.

[13] E. Charton, M. Gagnon, and B. Ozell. Automatic Semantic Web Annotation of Named Entities. In *Advances in Artificial Intelligence*, pages 74–85. Springer, 2011.

[14] 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.

[15] M. T. Enrico Franconi, Paolo Guagliardo. Quelo: a NL-based Intelligent Query Interface. In *Procs of the Second Workshop on Controlled Natural Languages (CNL 2010)*, 2010.

[16] 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.

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

[18] 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*, Dublin, Ireland, July 28 - August 01 2013.

[19] L. Faulkner. Beyond the Five-User Assumption: Benefits of Increased Sample Sizes in Usability Testing. *Behavior Research Methods, Instruments, & Computers*, 35(3):379–383, 2003.

[20] 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*, pages 1239–1246. International World Wide Web Conferences Steering Committee, 2013.

[21] 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*, pages 229–232. ACM, 2011.

[22] 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*, pages 457–466. ACM, 2009.

[23] 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*, pages 1–8. ACM, 2011.

[24] D. Mollá, M. Van Zaanen, and D. Smith. Named Entity Recognition for Question Answering. *Proceedings of ALTW*, pages 51–58, 2006.

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

[26] 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*, Bonn, GERMANY, 10 2011.

[27] 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*, pages 73–76. Association for Computational Linguistics, 2012.

[28] G. Rizzo, R. Troncy, S. Hellmann, and M. Bruemmer. NERD meets NIF: Lifting NLP Extraction Results to the Linked Data Cloud. *LDOW*, 937, 2012.

[29] 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.

[30] 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)*, Thessaloniki, Greece, November 2013.

[31] J. Umbrich, M. Karnstedt, A. Hogan, and J. X. Parreira. Hybrid SPARQL Queries: Fresh vs. Fast Results. In *The Semantic Web–ISWC 2012*, pages 608–624. Springer, 2012.

[32] 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):457–468, 1992.

[33] 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):1450–1453, 2011.