Universidad Zaragoza

1542

Trabajo Fin de Grado
Grado en Ingeniería Informática

Desarrollo de un sistema para la población de bases de conocimiento en la Web de datos

Development of a system to populate Knowledge Bases on the Web of Data

Autor

Ismael Rodríguez Hernández

Directores

Raquel Trillo Lado
Roberto Yus Peirote

Universidad de Zaragoza
Escuela de Ingeniería y Arquitectura

Junio 2016

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Ismael Rodríguez Hernández,

con nº de DNI 26498844S en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Desarrollo de un sistema para la población de bases de conocimiento en la Web de datos.,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 23 de Junio de 2016.

[illegible]

Fdo: Ismael Rodríguez Hernández

Agradecimientos

En primer lugar, me gustaría dar las gracias a los directores de mi proyecto, Raquel y Roberto, por su completa dedicación y apoyo, contestando siempre a los cientos de correos que les he enviado e incluso realizando videoconferencia con nueve horas de diferencia horaria. Me gustaría reconocer el esfuerzo desinteresado que han hecho manteniéndome informado en todo momento de becas, concursos y oportunidades, permitiéndome así el haber participado en la elaboración de dos artículos de investigación.

También quiero dar las gracias a mis compañeros y amigos de clase, especialmente a David, Luis, Raúl y Sergio, con los que he compartido los momentos más duros de la carrera, pero también los mejores. Me han hecho darme cuenta de lo importante que es el trabajo en grupo; incluso en los momentos más estresantes siempre estaban dispuestos a ayudarme.

Quiero también agradecer a los organizadores y participantes de los eventos “Editatón por la visibilidad de las mujeres de Aragón” y “Wikinformática 2016” por usar mi prototipo, proporcionándome así mucha información útil.

Por último, y no por ello menos importante, quisiera dar las gracias a mis padres y mi hermano, a mi novia Marta y a mis amigos, por su apoyo incondicional, su paciencia cuando estaba más agobiado (casi cada dia) y por darme ánimos cada dia desde que empecé la carrera. Sin ellos, no habría llegado hasta aquí; es por eso por lo que quiero dedicarles todo el esfuerzo depositado en este proyecto.

¡Muchas gracias!

Resumen

Durante las últimas décadas, el uso de la World Wide Web ha estado creciendo de forma exponencial, en gran parte gracias a la capacidad de los usuarios de aportar contenidos. Esta expansión ha convertido a la Web en una gran fuente de datos heterogénea. Sin embargo, la Web estaba orientada a las personas y no al procesado automático de la información por parte de agentes software. Para facilitar esto, han surgido diferentes iniciativas, metodologías y tecnologías agrupadas bajo las denominaciones de Web Semántica (*Semantic Web*), y Web de datos enlazados (*Web of Linked Data*). Sus pilares fundamentales son las ontologías, definidas como especificaciones explícitas formales de acuerdo a una conceptualización, y las bases de conocimiento (Knowledge Bases), repositorios con datos modelados según una ontología. Muchas de estas bases de conocimiento son pobladas con datos de forma manual, mientras que otras usan como fuente páginas web de las que se extrae la información mediante técnicas automáticas. Un ejemplo de esto último es DBpedia, cuyos datos son obtenidos de los infoboxes, pequeñas cajas de información estructurada que acompañan a cada artículo de Wikipedia.

Actualmente, uno de los grandes problemas de estas bases de conocimiento es la gran cantidad de errores e inconsistencias en los datos, la falta de precisión y la ausencia de enlaces o relaciones entre datos que deberían estar relacionados. Estos problemas son, en parte, debidos al desconocimiento de los usuarios sobre los procesos de inserción de datos. La falta de información sobre la estructura de las bases de conocimiento provoca que no sepan qué pueden o deben introducir, ni en qué forma deben hacerlo. Por otra parte, aunque existen técnicas automáticas de inserción de datos, suelen tener un rendimiento más bajo que usuarios especialistas, sobre todo si las fuentes usadas son de baja calidad.

Este proyecto plantea el análisis, diseño y desarrollo de un sistema que ayuda a los usuarios a crear contenido para poblar bases de conocimiento. Dicho sistema proporciona al usuario información sobre qué datos y metadatos pueden introducirse y qué formato deben emplear, sugiriéndoles posibles valores para diferentes campos, y ayudándoles a relacionar los nuevos datos con datos ya existentes cuando sea posible. Para ello, el sistema hace uso tanto de técnicas estadísticas sobre datos ya introducidos, como de técnicas semánticas sobre las posibles relaciones y restricciones definidas en la base de conocimiento con la que se trabaja. Además, el sistema desarrollado está accesible como aplicación web (http://sid.cps.unizar.es/Infoboxer), es adaptable a distintas bases de conocimiento y permite exportar el contenido creado en diferentes formatos, incluyendo RDF e infobox de Wikipedia.

Por último señalar que el sistema ha sido probado en tres evaluaciones con usuarios, en las que ha demostrado su efectividad y sencillez para crear contenido de mayor calidad que sin su uso, y que se han escrito dos artículos de investigación sobre este trabajo; uno de ellos aceptado para su exposición y publicación en las XXI Jornadas de Ingeniería del Software y Bases de Datos (JISBD), y el otro en proceso de revisión en la *15th International Semantic Web Conference* (ISWC).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since its definition in 1989 by Tim Berners-Lee [3], the World Wide Web has been constantly evolving. With the Web 1.0, users could only visualize content. Later, Web 2.0 [23, 22] allowed them to interact with the contents and create theirs. Nowadays, more than 3,366 million people[1] use the Web. The enormous quantity of heterogeneous information present on the Web that could be interesting not only for people, but also for computers, motivated the proposal of the Semantic Web in the early 2000s [28]. It was defined as an evolution of the Web where data is semantically annotated to make it easily processable by computers. As annotating all the Web is a difficult task, the Semantic Web is still far from being widely adopted. However, some projects like Google's Knowledge Graph[2] or Siri[3] already show its possibilities. These tools are capable of exploiting *Knowledge Bases* and making inferences about their contents to answer human questions.

Knowledge Bases (KBs) contain data that is modelled according to an ontology ("[...] an explicit specification of conceptualizations" [15]) and are populated (data is inserted on them) from diverse sources; some of them manually, while others are populated automatically or semi-automatically from existing websites. For example, DBpedia[4] is a KB that obtains its data from Wikipedia *infoboxes*[5]. Nowadays, KB have many problems, such as *errors* in the information, *inconsistencies* (e.g., different users represent dates in a different format), *inaccuracies* (e.g. saying that the birth place of someone is a country instead of a city or town) and *unlinked data* (i.e, inserting new data that makes allusion to existing values or entities in the KB without linking to them, without following one of the principles of Linked Data [4]). One cause is the difficult process of manually populating a KB; the user has to know the structure of the associated ontology to discern what data can be introduced and its format. Another reason is that automatic techniques have lower performance populating KBs than users, specially if the data sources are of low quality.

Although there are some tools (such as Protègè [30] or Populous [19]) that try

[1]`http://www.internetworldstats.com/stats.htm`. All the URLs in this document have been last accessed on June 2016.

[2]`https://goo.gl/x72FVy`

[3]`http://www.apple.com/ios/siri/`

[4]`http://wiki.dbpedia.org/`

[5]Tables present on Wikipedia articles that summarize their most important information.

to help users to create and insert data, none of them suggest what kind of data can be introduced (if it is more appropriate a number, text, a date,...), how is data represented (e.g., how to format dates or person names) or what values already exist in the KB to be linked. Because of that, it is really difficult for people to introduce data correctly, therefore lowering the quality of content of the KB.

This TFG[6] is focused on the analysis and development of a system that helps users to create data to populate KBs. With this system, users do not need to know either the inner structure or the semantics of the KBs, as the system provides them with dynamic templates according to their needs. Statistic and semantic methods (such as inductive reasoning) are used to generate the templates, determining the properties and values that the user should introduce, and linking values to existing entities whenever possible. For that, a functional web prototype has been designed, implemented and highly optimized.

1.1 Goals

The main goal of this TFG is to develop a system which allows any user to populate a KB with high quality data: semantically correct, consistent, precise and linked to other data when possible. Therefore, the TFG is divided into two main parts:

1. **Analysis and design of the solution**

    - Research about ontologies, Linked Data, structure of KBs, and technologies to store them.
    - Design the operations needed to identify relevant attributes, types of values, and rank them according to its importance, using that information to generate templates (referred to as *semantic templates*) that help the user to create content for a KB.
    - Extend the design for being capable of integrating values from multiple categories.

2. **Development of a prototype**

    - Develop a web server that obtains, analyses, processes and stores the needed information using the designed operations.
    - Develop a web user interface that allows users to generate data for inserting it on a KB or Wikipedia infobox, using the operations and information generated on the server.
    - Test the created prototype with real users, thus checking if data generated is more accurate and linked than with other systems and if the user experience is good.

[6] *Trabajo Fin de Grado* (Final Degree Project).

1.2 Document structure

The rest of the document is structured as follows. First, Chapter 2 gives an overview of the technological context of the project, describing concepts, used software and related works. In Chapter 3, the approach to generate semantic templates using statistical and semantic information is detailed. Chapter 4 describes aspects of the developed prototype such as its architecture, interface, information flow, and evolution. Then it shows the results of the performed evaluations used to test the generated data quality and user satisfaction. Finally, in Chapter 5, the conclusions and personal opinion are shown, as well as the project schedule and lines of future work.

In addition, a series of appendices to extend the information of the main document are included. Appendix A shows the process of analysis and design through multiple diagrams. Appendix B contains instructions on how to set-up, configure and use the system. It also describes some technical aspects such as the exposed HTTP API and the SPARQL queries used. Appendix C shows the results of the different performed tests. Finally, Appendix D contains a user manual of the web application.

Chapter 2

Technological Overview

In this chapter, several concepts are described in order to illustrate the context of the project, including the technologies used in its development. Then, some related works are described, analysing the differences between their approaches and the proposed one.

2.1 Context of the project

This section first gives an overview of concepts about Semantic Web, so the next sections can be understood. Then, it is told what Wikipedia and an infobox are, as some Knowledge Bases are populated from them. Finally, a brief description of the different technologies and software used in the project is provided.

2.1.1 Semantic Web

The Semantic Web [28] or Web of Data is an extension of the World Wide Web that provides a standardized way of expressing relationships among web pages and their contents, thus allowing computers to process, understand and extract knowledge from the hyperlinked information. This concept is supported by ontologies, resource annotation with metadata and the use of rules and reasoners. In the following subsections these terms will be explained.

Ontologies

An ontology is an "explicit specification of a conceptualization" according to Gruber [15]. In other words, it is a shared vocabulary used to model a domain, that explicitly defines the "types of concepts used and the constrains on their use" [26]. Ontologies are sets of assertions that define:

1. *Classes*, also called *concepts*, are collections of objects, and are identified by a name (e.g., Person or Country).

2. *Properties* define what types of relations can occur between objects of certain classes. Two classes are related through one or more properties; in a relationship, one of the classes is the Domain of the property, and the other

the Range. The Domain class restricts the set of individuals to which the property can be applied, while the Range restricts the type of the value of the property.

3. *Instances* are specific objects, and belong to at least one class. (e.g., Spain is an instance of the class Country). Instances can be related to other instances or values through the properties defined on their classes in the ontology. (e.g. given the axiom between classes “Person birthPlace Place”, a possible relation between instances would be “Obama birthPlace Honolulu”, and given the axiom “Person age Number”, the relation “Obama age 54” would be possible, too).

Some authors consider that the instances are part of the ontology, while others don’t. Among the former, most of them use ontology and Knowledge Base as interchangeable synonyms. In this project, the ontology is considered to specify only the classes and properties of a domain, while the instances and facts about them are defined in Knowledge Bases, later explained.

Representation Languages. In order to create ontologies, three standard languages that provide a reusable vocabulary have been developed, each one serving as a base for the next one:

1. *RDF* (*Resource Description Framework*) [20], a language used for conceptual modelling of information of web resources. The central element of RDF is the statement. A statement is a triple (S,P,O) where:

   - S is a URI that represents the subject of the statement.
   - P is also a URI but it represents a binary relationship between S and O (i.e., a property of the subject).
   - O is either a URI or a literal denominated object that represents the value of the property P for the subject S.

   RDF triples can be stored in triplestores, and compressed using formats like HDT[1] which substantially reduce the dataset size.

2. *RDFS* (*RDF Schema*) [9], a semantic extension of RDF that was created to reduce the lack of semantic expressiveness of RDF and to facilitate the definition of vocabularies, as in RDF there is no way either to define classes or to apply domain and range constraints to properties.

3. *OWL*, the *Web Ontology Language* [31], a vocabulary description language built using RDF and RDFS that provides new terms for describing resources in more detail, thus allowing computers to extract implicit conclusions and relations of data.

[1]`http://www.rdfhdt.org/`

The ontologies, that contain the semantic restrictions and vocabulary, are usually represented using the OWL language, while the Knowledge Bases, that only contain instances, are described with the RDF language.

Query Language. SPARQL [16] is a semantic query language used to retrieve and manipulate data stored in RDF format in a SPARQL server (e.g., Jena Fuseki[2], which provides REST-style SPARQL updates and queries using the SPARQL protocol over HTTP). It was standardized by the *RDF Data Access Working Group (DAWG)*[3] of the World Wide Web Consortium, and is one of the key technologies of the Semantic Web. A modified version of Fuseki that provides read-only queries on HDT files instead of plain RDF is used in the project.

Annotations

To make the Semantic Web a reality, besides having the enormous amount of data on the web available in a standard format, relationships between data have to be available. Linked Data [4] is a collection of interrelated datasets that tries to reach that goal. For that, it makes uses of annotations and standard technologies. An annotation is additional information (metadata) associated to an existing piece of data or resource, whose purpose is to provide a formal representation of it understandable by computers. Linked Data uses RDF to relate and describe data, and URIs are used to identify any kind of concept. There are a set of rules that govern Linked Data:

1. Use URIs as names for every entity (e.g., http://dbpedia.org/resource/Spain).
2. Use HTTP URIs so that people can look up those names.
3. Include links to other URIs, so that they can discover more things.

Publishing content that follows these rules helps to the growth and adoption of the Linked Data.

Knowledge Bases. A Knowledge Base (KB) is a set of data that describes the instances of an ontology, facts modelled according to it, following its restrictions and relations. KBs are centred in a specific domain, and can be seen as a node or dataset of the Linked Data Cloud (see Figure 2.1).

DBpedia [5] is one of the most famous KBs and the central point of the Linked Data Cloud. It pretends to be a general purpose KB with data extracted semi-automatically from Wikipedia, but also being connected with well-known KBs like GeoNames [32], Gutenberg Project[4] or FOAF [7].

Semantic Reasoners

A Semantic Reasoner [35] is a software that makes logical deductions from a set of axioms (i.e., if “YoungPerson“ is equivalent to “hasAge<30” and “Ismael hasAge

[2] `https://jena.apache.org/documentation/serving_data/`

[3] `https://www.w3.org/2001/sw/DataAccess/homepage-20080115`

[4] `https://www.gutenberg.org/`

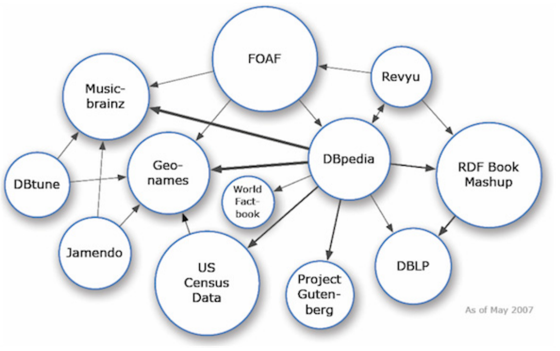


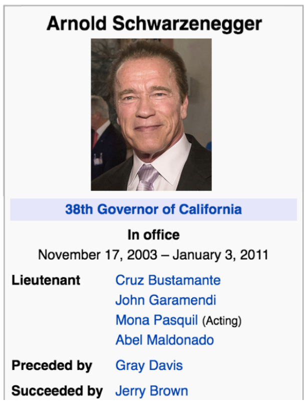
Figure 2.1: Knowledge Bases connected on the Linked Data Cloud. Obtained from http://goo.gl/ivqJsY on May 2016.

21", then "Ismael is a YoungPerson"). Semantic reasoners based on Description Logics (DL)[5] usually perform tasks such as consistency checking, verification of hierarchical relationships, classification of new terms, and instance retrieval.

2.1.2 Wikipedia

Since its creation in 2001, Wikipedia[6], a free, collaborative, and digital encyclopedia, has become the one of the most important sources of reliable information on the Web. The English version of Wikipedia currently contains more than five million English articles[7] and there are almost 300 versions of Wikipedia in other languages with different content.

Wikipedia articles are often split into two parts: a body of unstructured text with details on the article's subject and an optional semi–structured *infobox* (Figure 2.2) that summarizes the most important facts about the article's subject. The addition of an infobox enhances an article in several ways: 1) it summarizes important information in an easy-to-read format and enable the comparison across different pages; 2) the structured format makes it easy to develop tools to consume Wikipedia information; and 3) it provides increasing integration with Wikidata [29], avoiding replicated and non consistent data among Wikipedia versions in different languages.



Arnold Schwarzenegger

38th Governor of California

In office
November 17, 2003 – January 3, 2011

Lieutenant	Cruz Bustamante John Garamendi Mona Pasquil (Acting) Abel Maldonado
Preceded by	Gray Davis
Succeeded by	Jerry Brown

Figure 2.2: Infobox example.

The number and quality of infoboxes have a great impact on the success of projects that use them, such as Google's Knowledge Graph and DBpedia. Therefore, Wikipedia infoboxes are very important in this project, as they are an indirect way of populating multiple KBs.

[5]Description Logics (DL) [2] is "a family of knowledge representation formalism that represent knowledge of an application domain [...]" which provides a logical formalism for ontologies.

[6]`http://wikipedia.org`

[7]`https://en.wikipedia.org/wiki/Wikipedia:Statistics`

2.1.3 Other Technologies

The programming languages, querying languages, and libraries used in the development of this project are briefly described in this section.

- *Java* [14] is a multi-platform, object-oriented, and general-purpose programming language. It has been chosen for developing the web server (Backend) because of the acquired experience with it and the great amount of semantic technologies available for this language.
- *HTML* [17] and *CSS* [6] are the standard languages for web development. HTML is a mark-up language used to describe the structure of a website, and CSS is a style sheet language using for providing style. They have been selected to develop the Frontend of the system because their wide adoption and compatibility among web browsers.
- *Javascript* [10] is an object-oriented, weak typed language mostly used to give dynamism to web pages. It has been chosen to develop the web interface (Frontend) because the needed dynamism and the great amount of available libraries.
- *AngularJS*[8] is a Javascript framework used to create powerful single-page applications. It has been chosen because of the previous experience with it and the easiness it provides compared to Javascript without any framework.
- *Spring Framework*[9] is a framework that eases the development of Java applications, providing services such as RESTful Web Services. It has been chosen because it makes possible a fast and easy set-up and maintenance of the project.
- *Bootstrap*[10] is an open-source Frontend library whose goal is to free the developer of designing all the interface components.
- *OWL API V3.4.4* [18] is a Java library to manage OWL ontologies that gives developers a high level of abstraction. It is widely used and a lot of documentation can be found online.
- *HermiT v1.3.8.1* [25] is a Java reasoner that supports OWL. It became the first DL reasoner that had the capacity of classifying large ontologies efficiently.
- *MySQL*[11] is the most popular[12] open source relational database management system (RDBMS). It is used as an auxiliar storage system to save analytics and caché data of value suggestions. The query language SQL [8] (Structured

[8]`https://angularjs.org/`

[9]`https://projects.spring.io/spring-framework/`

[10]`http://getbootstrap.com/`

[11]`https://www.mysql.com/`

[12]`http://db-engines.com/en/ranking`

Query Language) is used for querying, creating, updating and deleting data stored in it.

- *MongoDB*[13] is a document-oriented NoSQL database that claims to provide flexibility and a big performance. It was used in the preliminary prototype of the system developed.

2.2 State of the Art

Due to the growing use of ontologies and KBs, many research works have focused on their creation and population either manually or with automatic techniques. In this section, the most representative systems focused on the creation and population of KBs and ontologies are presented. As Wikipedia infoboxes are the source of some KBs, tools focused on the creation and management of infoboxes are also commented.

2.2.1 Tools to Create And Populate KBs

Four relevant tools related to KBs and ontologies are now described:

- *RightField* [33] is a tool that "allows data collectors to generate Excel spreadsheet templates embedded with Ontologies [...]" (see Figure 2.3). The main goal of RightField is to control what data is inserted into an Excel spreadsheet using manually selected semantic restrictions. In contrast, the main goal of the system presented in this TFG is to automatically provide the users with semantic and statistic information to create data for being inserted into a KB.

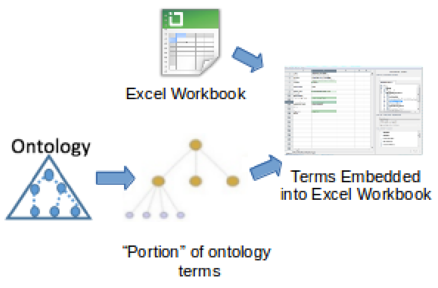

Excel Workbook
Ontology
Terms Embedded into Excel Workbook
"Portion" of ontology terms

Figure 2.3: RightField process schema.

- *Populous* [19] is a tool built on top of RightField that helps users to create new ontologies and add terms to existing ones from repetitive data. It is focused on engaging a wide community of scientist in the mass production of ontological content. This tool can export the content to RDF and check semantic constraints in a similar way to the presented system. However, in Populous, the suggestions of the terms and values to be inserted are based only on semantic information and not on statistic data considering the context.

[13]`https://www.mongodb.com/es`

- *Protégé* [30] is an "open source ontology editor and knowledge management system", used to create and populate ontologies (see Figure 2.4). It also allows to infer knowledge and check the consistency of the loaded ontologies by using a reasoner. Although it can populate ontologies and KBs, it is very time-consuming as it does not provides suggestions at all, so possible properties, values, and ranges have to be looked up manually.

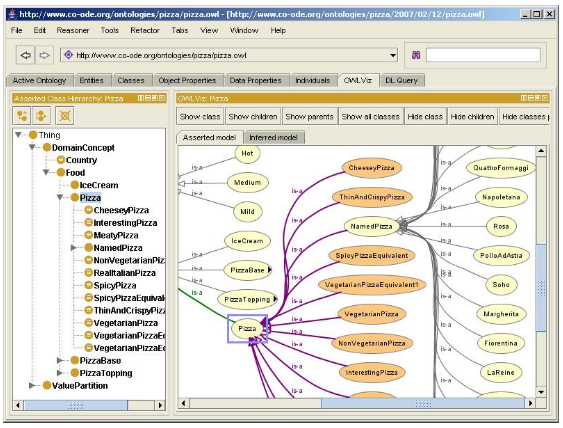


Figure 2.4: Screenshot: Protége. Obtained from http://goo.gl/OP02qQ on June 2016.

- *OntoPop* [1] is a tool that (semi-)automatically inserts new instances to a KB as defined by an ontology, using information from natural language documents. To do so, it maps linguistic extractions with concepts of the ontology. Unlike the developed system, it requires no user interaction, but needs non-ambiguous documents as a source. The presence of a person is usually required to fix errors and inaccuracy in the instances created automatically.

Although these systems are related to KBs and ontologies, none of them is specifically designed to ease the manual process of creating content for KBs by non-expert users.

2.2.2 Tools to Create and Manage Infoboxes

Infoboxes are useful for both humans and information systems as they are the source of multiple KBs. So, in this section, current tools to manage infoboxes are described:

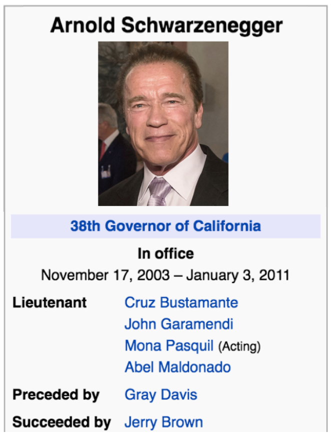
1. *Wikipedia manual editor* is the most used tool to create infoboxes. First, editors have to select an appropriate template by considering the categories related to the article. Then, the template has to be filled with information related to the article. Choosing the proper template can be a source of misunderstandings, as several ones could be selected for an article, and different editors could select different templates for the same type of articles. Besides, templates can have a large number of attributes, multiple attribute names (e.g., date of birth and birthdate) for the same purpose, among other problems that make users difficult to create quality and consistent infoboxes.

Figure 2.5 shows an excerpt of the infobox in the Wikipedia article of Arnold Schwarzenegger on the left and the code to generate it on the right.

2. *Wikidata* [29] is a collaborative KB that provides a common source of data for Wikipedia, infoboxes, and other Wikimedia projects. Although Wikidata rolled out a feature which suggests popular attributes for a given entity[14], it cannot generate templates for entities belonging to multiple categories, nor does it suggest or semantically control the attribute value and their types.

3. Multiple research projects are also focused on developing automatic techniques to enhance infoboxes. Wu and Weld [34] developed *KOG*, an autonomous system that automatically builds an ontology using Wikipedia infoboxes and WordNet to semantically represent the attribute value pairs of each infobox. Sultana et al. [27] developed SVM classifiers to recommend a infobox template type (Soccer Player, Actor) using features based on article content, article category, and related entities. *iPopulator* [21] is a tool that mines the Wikipedia article text to identify additional attribute value pairs for an infobox. Fetahu et al. [13] presented techniques to suggest news articles that could be used to complete Wikipedia entity articles and infoboxes.

In conclusion, although multiple and interesting projects related to infoboxes exist, none of them provides users with so much suggestions and recommendations as the system proposed in this TFG.



Arnold Schwarzenegger

38th Governor of California

In office
November 17, 2003 – January 3, 2011

Lieutenant	Cruz Bustamante John Garamendi Mona Pasquil (Acting) Abel Maldonado
Preceded by	Gray Davis
Succeeded by	Jerry Brown

```
{{Infobox officeholder
|name          = Arnold Schwarzenegger
|image         = Arnold Schwarzenegger February 2015.jpg
|caption       = Schwarzenegger in 2015
|order         = [[List of Governors of California|38th]]
                 [[Governor of California]]
|term_start    = November 17, 2003
|term_end      = January 3, 2011
|predecessor   = [[Gray Davis]]
|successor     = [[Jerry Brown]]

...
}
```

Figure 2.5: Example of a Wikipedia infobox and code to generate it.

[14]http://lists.wikimedia.org/pipermail/wikidata-l/2014-July/004148.html

Chapter 3

Generation of Semantic Templates

In this chapter, the proposed approach to generate semantic templates (information about the relevant attributes, types of values, and values that can be inserted into a KB) using statistical and semantic information is described. First, an overview of the approach is given. Then, it is justified why both semantic and statistical information are used. Finally, the most important processes are detailed.

3.1 Overview of the Approach

The main goal of the approach is to generate *semantic templates* that help users to create instances for an existing KB. These semantic templates contain semantically relevant attributes or properties (e.g., "birth place" for a person) whose expected values are controlled semantically to prevent users from filling them with incorrect information (e.g., the value for the "birth place" property should be a place). The semantic templates use statistical information of an existing KB to rank properties according to their popularity and provide samples of the expected values.

The semantic templates are generated from information extracted from the considered KB and its associated ontology. Thus, to generate a semantic template, a user has to select a set of categories (e.g., "Governor", "Actor", and "Body Builder" in the case of "Arnold Schwarzenegger"), defined in the ontology. Then, the system performs the following steps (see Figure 3.1):

1. Obtain instances in the KB with the same category. For example, if using DBpedia and the selected category is "Actor", it obtains people such as "Tom Hanks" and "Sylvester Stallone".

2. Identify interesting attributes or properties using both statistical information from existing entities in the KB, and semantic information from the associated ontology. For example, attributes such as "birth place" or "award" would be retrieved for the previous example.

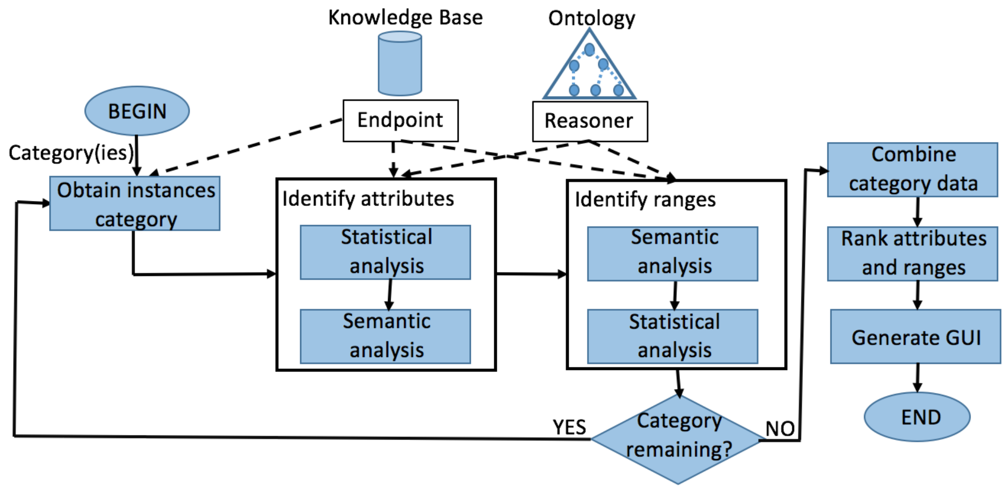


Figure 3.1: Main steps to generate a semantic template.

3. For the identified attributes or properties, obtain the expected value types (ranges) using both statistic and semantic information. For example, for the attribute "birth place", some expected value types are "Country" and "Settlement".

4. Repeat the previous steps for each category selected by the user and combine the obtained information. Attributes with the same name are merged, taking into account their expected value types.

5. Rank the attributes and ranges according to their usage and create the semantic template with this information. For example, properties such as "name" and "birth place" are more popular than "death place" or "siblings".

Finally, a graphical user interface is generated to show the template to users, and suggestions are provided in real time when the user types values for an attribute. After users fill the template the system can export the generated data to RDF so it can be later imported into the desired KB. As some KBs, such as DBpedia [5], are populated automatically from Wikipedia infoboxes, the system can generate infobox code as well. In the following sections more details about how the approach works are provided.

3.2 Combining Statistical and Semantic Information

Statistical and semantic information is used to determine which information should be included in the semantic template. Ideally, this information could be extracted from the associated ontology; in particular from the parts of the ontology that define which attributes or properties can be used with the selected classes or categories, their restrictions such as types of data for each attribute or property (their ranges), etc. However, this can be problematic when the definitions of the domains and ranges of the properties are not accurate. For example, DBpedia 2015-04 (the main KB used while building the prototype) contains 2,863 properties; 356 of them (12%) have no domain defined, 376 (13%) have no range, and 131 (5%) have not either domain or range. Moreover, statistical information about the instances of a category can be also inaccurate depending on the number of instances available. For example, none of the 235 instances of the category “Body Builder” in the previous KB uses properties such as “spouse” or “known for”, which might be relevant for a body builder infobox. Therefore, the best option is to use a combination of statistical and semantic information to generate complete and precise templates and accurate content.

3.3 Identifying Relevant Attributes

Interesting and relevant attributes are obtained by combining semantic and statistical properties for a given category selected by the user. A semantic reasoner and the ontology associated with the KB are used to identify semantically relevant attributes (also referred to as “semantic properties”). The goal is to obtain properties where the given category is a domain. Additional relevant properties are included by considering properties where each of the superclass of the given category is a domain. Therefore, given a category c_i, all the properties $(d, r) : p$ such that the category c_i is subsumed by the domain of p (i.e., $c_i \sqsubseteq d, c_i \in C$) are found. For example, given the category “Actor” using DBpedia 2015-04 , some of the semantic properties for that category and its parent classes are shown in Table 3.1.

Actor	Artist	Person
nationalFilmAward	style	colleague
arielAward	voiceType	bloodGroup
iftaAward	mentor	eyeColour
...	...	...
Total: 9	Total: 23	Total: 272

Table 3.1: Semantic properties obtained for different categories.

Statistically relevant attributes (which are also referred to as “statistical properties”) are also obtained by identifying properties associated with the instances of the given category. These can be used to narrow the semantic property set in cases where the domain/category is too broad. With the list of instances obtained for the given category, a list of attributes used by these instances is generated (along with information about their popularity in terms of number of instances using each attribute). Duplicate counts are avoided by noting distinct attribute for every instance only once (at this point it is interesting how many different instances of the category are using the property to highlight its popularity). For example, the property “award” appears several times with many instances of the actor category (as they have won several awards) but it is only counted once.

Range	Country (statistical)	Settlement (statistical)	owl#Thing (statistical)	Place (semantic)
Uses of range	547	388	129	0
Some suggested values	England (159) Sweden (107)	London (118) Stockholm (73)	Sussex (16) Kent (7)	Tokyo (0) California (0)
	United Kingdom (101)	Paris (39)	Middlesex (7)	Los Angeles (0)

Table 3.2: Types of values for the "Actor" category and the "deathPlace" property.

3.4 Identifying Types of Values

For each relevant property obtained in the previous step, a semantic restriction over its expected values is retrieved, which is referred to as *types of values* or *ranges*. In an analogous way to the previous step, "semantic value types" and "statistical value types" are obtained. The "semantic value types" are the ranges defined in the ontology for each property, while the "statistical value types" are ranges popular among the instances in the KB. For the latter, a list of attribute values for a given category and attribute is first obtained, by identifying the list of triples in the KB whose subject is an instance of the given category and whose property is the given attribute. Based on the attribute, value types are either semantic classes (e.g.,"City" or "Town") or basic datatypes (e.g., a text string or a number). For example, the types of values obtained for the category "Actor" and the property "deathPlace" are shown in Table 3.2. Notice that, "Place" is the range of the property in the ontology while "Country", "Settlement" and "owl:Thing" are types of values from instances of "Actor" in the KB.

Also, for each type of value a list of previously popular used values is generated from the statistical information obtained before. This list is used both as a suggestion for users and as a way of helping users to select an existing entity from the KB to create a linked value. How to generate the suggestions of values and rank them is detailed in Section 3.7.

3.5 Dealing with Multiple Categories

Many instances of a KB could have more than one category associated (for instance, Arnold Schwarzenegger is an actor, a governor, and body builder). Therefore, two approaches have been studied and incorporated in the system to generate semantic template for instances associated with a set of categories $C = \{c_1 ... c_n\}$. When selecting the instances of the KB used to generate statistical information, we can consider instances fulfilling:

1. $\forall c_i \in C \; \exists$ <*instance, rdf:type,* c_i>. Consider instances which belong to all the categories selected by the user.

2. $\exists$ <*instance, rdf:type, c_i*> | $c_i \in C$. Consider instances which belong to at least one category selected by the user.

The first approach provides more precise templates than the second approach, as it can generate a template specialized in all the selected categories. However, the number of instances fulfilling requirements of the first approach might decrease with an increasing number of categories selected by the user. This could be problematic for the system's approach as its model is partially based on statistical data from the existing instances. For instance, in the KB used in the experiments (DBpedia 2015-04), no instance is associated with the three categories that can be used for Arnold Schwarzenegger ("Actor", "Governor", and "Body builder") and not even with two of them. In fact, the most common pair of types[1] associated with instances are "*Artist*" and "*Writer*" (7810 instances have these two types) and the system obtains 62 relevant properties from these instances.

The second approach is obviously less restrictive than the first as it considers instances belonging to any of the categories. Therefore, the number of instances obtained might be greater and thus, more properties might be extracted by the system. However, some of the properties obtained might be repeated. For instance, from the 146 properties extracted for "Writer" and "Artist", 66 are duplicate (e.g., "birthName" and "birthPlace"). Duplicate properties are combined and the computed value types of the different categories are merged.

3.6 Ranking Attributes and Types of Values

The system ranks the list of attributes to be filled in a template in order to improve the users' experience. The attributes are ordered based on their popularity in the KB. Thus, the higher the frequency of the use of an attribute in the instances of the selected categories in the KB, the higher its position in the ranking. The attributes whose domain is one of the selected categories but not used in any instance of those categories, are listed in the lowest positions of the ranking.

The types of values or ranges of each property are also ranked according to its use frequency by the instances of the selected categories for that property. However, in order to not provide the user with too much information, only a maximum of four ranges are shown for every property; if N ranges are obtained, being N $>$ 3, then the (N-3) less frequent ranges are grouped into one, shown with the name of a "semantic range" chosen by taking into account the ontology used.

3.7 Providing Significant Suggestions of Values

The system provides a ranked list of suggested values for each attribute and shown range (type of value) that changes as the user types.

When the range is a class (e.g., "Soccer Team", "City", "Person", ...), the system suggests values from the whole KB whose type is the range of the attribute,

[1]The only types considered are the ones from the DBpedia ontology which can be associated with categories from Wikipedia infoboxes. Other categories (e.g., from Yago) are not used.

thus allowing the selection of values even if they have never been used for the selected categories and property. They are ranked according to their use frequency and similarity with the typed text. In this case, as an exception, the frequency of each value is calculated as the number of uses by instances which belong to *all* the selected categories, for that property. For instance, for the "Actor" and "Governor" categories, the property "birthPlace", and the range "City", all the cities in the KB are suggested, ordered by how many times people that are both actor and governor were born there. This provides more specialized values, and when there are no instances belonging to all the selected categories, values from the whole KB are still suggested.

When the range of the property is a basic datatype (e.g., a number, string, or date), values from all the KB are not suggested, as there are millions of different basic datatypes values (e.g., text descriptions, quotes, dates, ...) that provide no relevant information. It suggests values used by instances which belong to *at least one* selected category, so when there are no instances which belong to all the classes, some values are still suggested. For instance, for the categories "Actor" and "Governor", the property "description", and the range "String", the descriptions of people that are either actors or governors are suggested, ordered by how many times these people use each description.

Chapter 4

Prototype of the System

In this chapter, first aspects about the fully functional system prototype developed, composed by a Frontend and a Backend, are described. For that, the system architecture is defined, the Frontend interface is explained, and some aspects about the Backend (use of different KBs, generation of infobox code, and evolution) are detailed. Finally, the performed evaluations that test the quality of created contents and the user's satisfaction are explained.

4.1 Architecture of the System

The developed system's architecture is divided into two main different components: the Frontend and the Backend of a web application system.

- The Frontend of the web application allows users to visualize data obtained from the Backend (properties, value types...) and create new contents for a KB. It has been developed using AngularJS and Bootstrap. The Graphical User Interface (GUI) has been adapted to the expertise of users by displaying statistical information for expert and non-expert users (see Section 4.1.1).
- The main component of the Backend is a Java application deployed on a web server that processes data, maintains a cache of the results, and makes them available in JSON format through an HTTP interface. It has the following parts or modules:
  - The "Main operations" module. It is responsible for the more important and CPU-intense operations such as identifying relevant attributes or value types, using a KB and its associated ontology. Every operation result of this module is stored in a file cache, so if it has already been calculated previously with the same parameters, it is returned immediately.
  - The "Suggestions provider" module. Its purpose is to provide real-time suggestions of possible values. All the possible values are retrieved from the SPARQL Endpoint and previously generated cache files, and stored in a indexed MySQL database so they can be efficiently queried every time the user types a text.

– The “Analytics handler” module. It registers users’ interactions with the web application (clicks, typed texts, times, etc) in a MySQL database.

– The “Exporter” module. It is in charge of exporting the data generated by the user in semantic format (RDF) or Wikipedia infobox format.

Moreover, as it can be seen in Figure 4.1 other components are also required. The system uses a SPARQL server Apache Jena Fuseki to store the KB that supports the statistical analysis of the system, and to provide access to it. This server is a modified version of Jena that allows to work with the KB stored in the HDT format instead of in a non-binary format (which speeds up the data management). The handling of the ontology to extract semantic properties and ranges is done by using the OWL API along with the HermiT reasoner. The HTTP interface for the Frontend component is created by using the Spring Framework. Finally, the MySQL database is also used to store statistics about the user actions (interactions with the system) to analyse how they work.

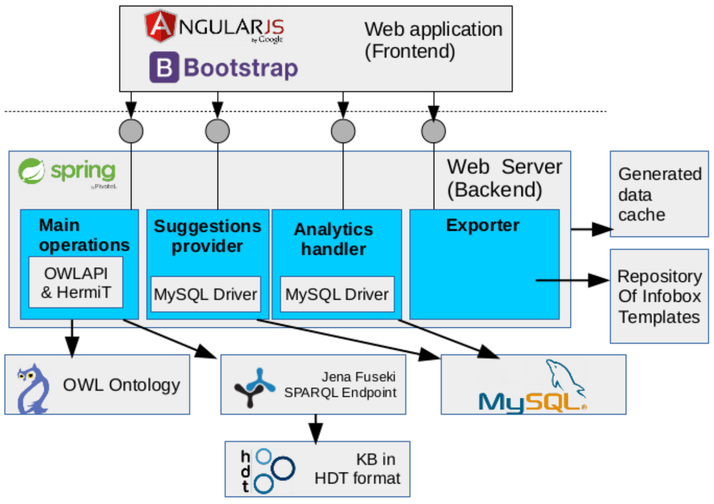


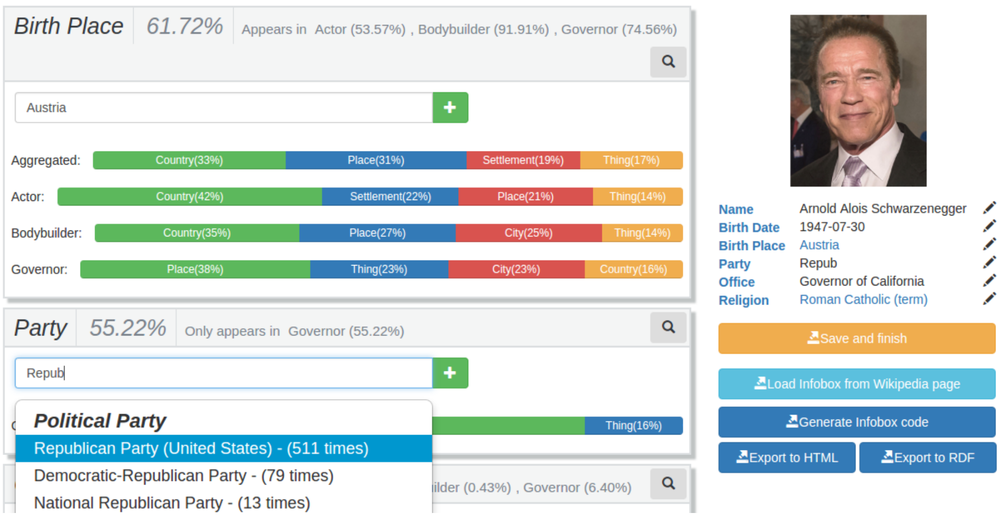
Figure 4.1: Technological system architecture.

4.1.1 Frontend

The Frontend has two different interface modes: the “Expert Mode”, that shows details about statistical information; and a “Basic Mode”, that hides those details for non-experts users. Both modes divide the interface in two parts (see Figures 4.2 and 4.3): the left part shows a list of properties or attributes, where the statistical attributes are showed before the semantic attributes, ranked by frequency. Besides, in this left part users are allowed to introduce values. The right part shows a preview of the entity being created, using a visual format similar to Wikipedia infoboxes, and a button that allows the user to submit the result to the server. A complete user manual of the Frontend can be found on Appendix D.5.

Expert Mode

The "Expert Mode" (see Figure 4.2) shows, for every retrieved property, information such as the frequency of instances from the selected categories that use the property at least once. Also, if multiple categories are selected, the same information is displayed for each individual category. A "Search" button that opens a window with a Google search is also present. Possible ranges or types of values are represented with bars, where at most four ranges are shown. Ranges inside a bar are ranked according to the number of triples whose type of value is the given range, whose subject is an instance of the selected categories and whose property is the given one. Moreover, with multiple categories, a bar combining information of all categories (instances that belong to one or more of the categories) is also shown.


Birth Place 61.72% Appears in Actor (53.57%) , Bodybuilder (91.91%) , Governor (74.56%)

Austria

Aggregated: Country(33%) Place(31%) Settlement(19%) Thing(17%)

Actor: Country(42%) Settlement(22%) Place(21%) Thing(14%)

Bodybuilder: Country(35%) Place(27%) City(25%) Thing(14%)

Governor: Place(38%) Thing(23%) City(23%) Country(16%)

Party 55.22% Only appears in Governor (55.22%)

Repub

Political Party
Republican Party (United States) - (511 times)
Democratic-Republican Party - (79 times)
National Republican Party - (13 times)

Name	Arnold Alois Schwarzenegger
Birth Date	1947-07-30
Birth Place	Austria
Party	Repub
Office	Governor of California
Religion	Roman Catholic (term)

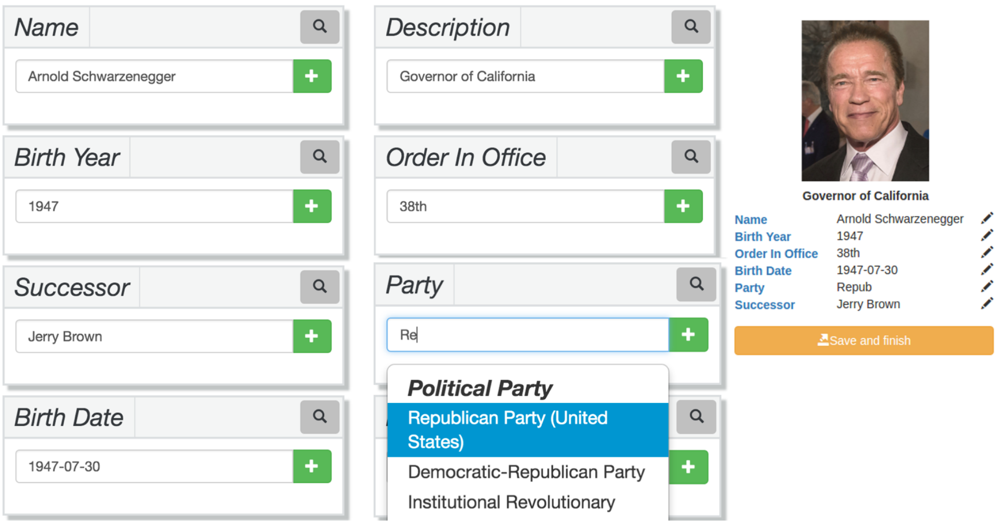
Figure 4.2: Screenshot: prototype's GUI for expert users.

When a user clicks on an input box, a list of significant suggestions is provided; this list changes as the user types. One group of suggestions is provided for each one of the ranges shown in the top bar (either an aggregation bar if more than one category was selected, or an individual bar). These groups have the range name at the top, and for each value, the number of uses is shown (the frequency used for ranking the suggestions, computed as it was explained in Section 3.7).

Regarding the right part where the infobox preview is shown, in this mode some buttons that allow the generation of RDF and Wikipedia code are displayed.

Basic mode

The "Basic Mode" (see Figure 4.3) hides statistical information, and only shows, for each property, a minimal box with its name, the same "Search" button as in the previous mode, and suggestions of values. Although no frequency information is displayed, properties are ranked according to it. No range bars are shown, but the suggestions of the values are grouped by ranges as in the "Expert Mode". The only difference is that the frequency each value is not displayed.


Name: Arnold Schwarzenegger

Description: Governor of California

Birth Year: 1947

Order In Office: 38th

Successor: Jerry Brown

Party: Re

Political Party
- Republican Party (United States)
- Democratic-Republican Party
- Institutional Revolutionary

Birth Date: 1947-07-30

Governor of California

Name	Arnold Schwarzenegger
Birth Year	1947
Order In Office	38th
Birth Date	1947-07-30
Party	Repub
Successor	Jerry Brown

Figure 4.3: Screenshot: prototype's GUI for non-expert users.

4.1.2 Backend

In this section, some aspects of the Backend are detailed, including an overview of the operations it performs, how it was made KB-agnostic, how Wikipedia infobox code is generated and its evolution.

Operations and Information Flow

The information flow between the Frontend and the web server can be seen on Figure 4.4. Firstly, there are three main operations that are repeated for every individual selected category ("Instance Count", "Property List" and "Range List"). Secondly, if several categories have been selected, then operations to aggregate information from those categories are performed.

The first operation, "Instance count", calculates the number of instances that belong to, at least, one of the given categories. The second one, "Property list", returns a list of relevant attributes obtained both semantically and statistically. The third one, "Range list", returns a list of ranges for each statistic property with their frequencies. Besides, as users' interactions are collected, when users click on interface elements, input some text or finish their activities, information about those actions is sent to the web server to be stored. For more details about these operations see the Sequence Diagrams on Appendix A.2.3.

Using Different Knowledge Bases

The system has been designed and developed to be KB agnostic, i.e., to generate templates for different and diverse KBs. To do so, all the parameters specific to a KB such as URI prefixes, how property, range, and instance labels are obtained, and available categories are isolated into configuration files. Thus, in order to load a new KB into the system, the user only has to download it, load it into the SPARQL endpoint and modify these configuration files. Four different KBs have

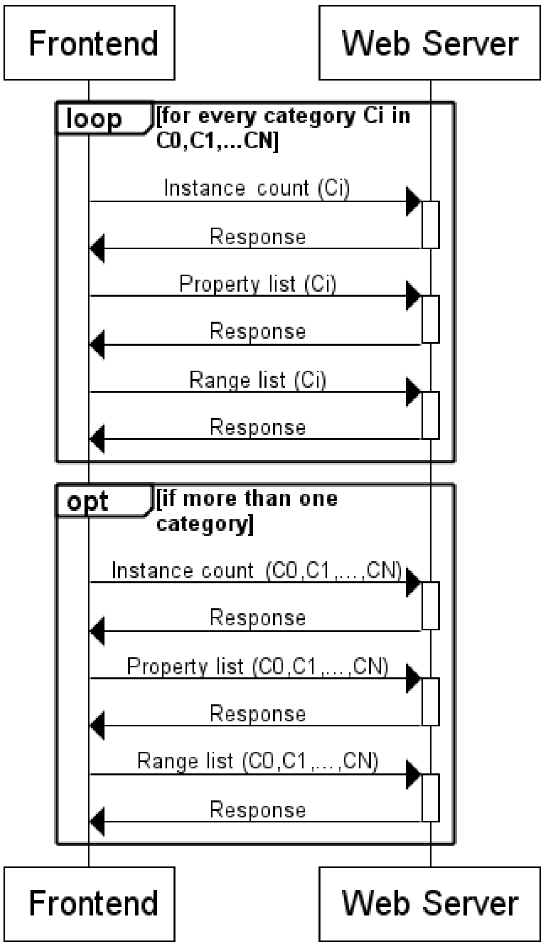

Frontend
Web Server
loop [for every category Ci in C0,C1,...CN]
Instance count (Ci)
Response
Property list (Ci)
Response
Range list (Ci)
Response
opt [if more than one category]
Instance count (C0,C1,...,CN)
Response
Property list (C0,C1,...,CN)
Response
Range list (C0,C1,...,CN)
Response
Frontend
Web Server

Figure 4.4: Sequence diagram: information flow between Frontend and Backend.

been tested, and the process to load them is detailed on Appendix B.3. These KBs are DBpedia (about 6,000,000 instances), GeoSpecies (about 20,000 instances), IIMB Test (about 200 instances), and SwetoDBLP (about 920,000 instances).

Generating Wikipedia Infobox code

As said before, the developed system, besides generating RDF code, also exports the created instance to Wikipedia infoboxes code because their importance to populate KBs such as DBpedia. To do so, when the user has finished filling the semantic template, the system selects from a local repository of Wikipedia infobox templates the most appropriate one depending on the categories the user selected. Then, the system fills the infobox template with the provided information. Finally, the infobox code is returned to the user.

That local repository of Wikipedia Infobox templates consists of one file for each category in the KB used. Those files contain the name of the appropriate infobox template and a series of associations between the property names in the KB and in the properties in the template. The most appropriates Wikipedia infobox templates have to be chosen beforehand by a human, and the associations in each file have to be established manually. Nevertheless, there are works on the area of Ontology Matching [11] that could generate these associations automatically.

Evolution

The system technology and how data was processed has evolved through the analysis and development, being more efficient every time a modification was done. The three main approaches are explained in the following:

The first approach (corresponding to a previous research prototype [37] on which this TFG is based on) used a NodeJS web server and a MongoDB database. To generate a template for a category, a file with the processed data for one unique category had to be manually prepared and stored into MongoDB. That process took about 10 hours and 190MB disk space for the “SoccerPlayer” category of DBpedia 2015-04. Then, the web server queried MongoDB, taking about 30 seconds the first time. This approach did not support templates with more than one category.

The second approach (see Figure 4.5), consisted of rewriting the program that made the category processing. That process was analysed and a Java program 1300% faster was implemented (for the “SoccerPlayer” category previously mentioned, it took about 45 min.) Also, the system was adapted for being able to generate templates with more than one category. Even it was a great improvement, it consumed a lot of time and disk space. The impedance between RDF and MongoDB made queries complicated an inefficient. So the whole process was reconsidered and a new version/approach of the system was designed.

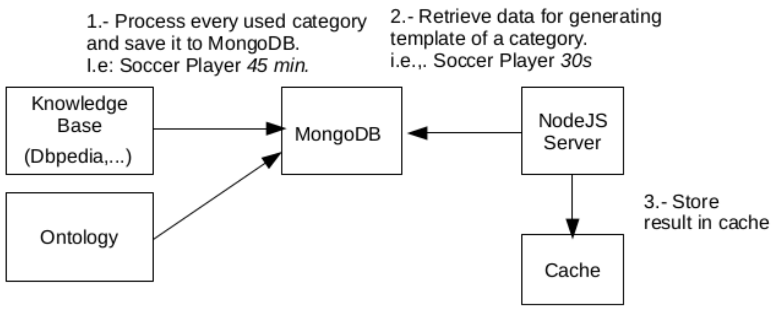

1.- Process every used category and save it to MongoDB. I.e: Soccer Player *45 min.*
2.- Retrieve data for generating template of a category. i.e.,. Soccer Player *30s*
Knowledge Base (Dbpedia,...)
Ontology
MongoDB
NodeJS Server
3.- Store result in cache
Cache

Figure 4.5: Generation of templates with MongoDB and NodeJS using DBpedia 2015-04.

The third and current approach (see Figure 4.6), uses a SPARQL endpoint to perform the needed operations. The previous operations were analysed and SPARQL queries with the same goal as the previous used queries were designed and tested with both Jena Fuseki and a modified version of Jena Fuseki using the RDF HDT format. Results, detailed on Appendix C.1, concluded that using a SPARQL endpoint was much faster (importing all the KB into the Endpoint took 10 minutes, and processing the “SoccerPlayer” category could take 3 minutes). Between the two tested SPARQL endpoints, the one that supports RDF HDT was chosen, as the KB occupies 714 MB, in contrast to 3 GB using plain RDF (full comparison data can be seen in Appendix C.1).

In summary, a new Backend of the system was designed by using Java, Spring, and OWL API. Moreover, the web server and the SPARQL queries have been redesigned as new optimizations were discovered. Thus, currently, when a new template is generated, the web server queries the KB through the Fuseki HDT server and obtains the necessary data in a short time. After that, the system caches the results in a file cache, making the following template generations almost immediately.

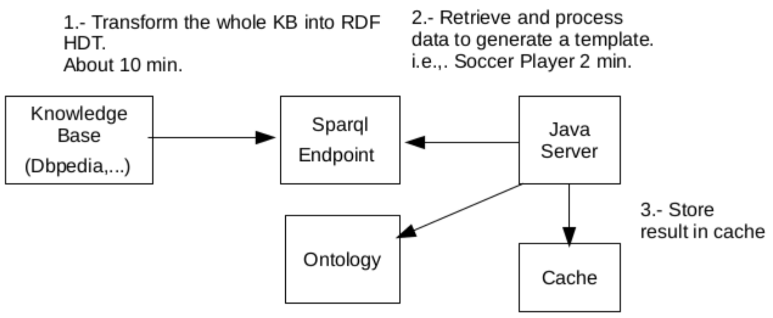

1.- Transform the whole KB into RDF HDT.
About 10 min.
2.- Retrieve and process data to generate a template.
i.e.,. Soccer Player 2 min.
Knowledge Base (Dbpedia,...)
Sparql Endpoint
Java Server
Ontology
Cache
3.- Store result in cache

Figure 4.6: Generation of templates with RDF HDT and Java using DBpedia 2015-04.

4.2 Experimental Evaluation

Three evaluations (two of them in Wikipedia *edit-a-thon* events, and one with Amazon Mechanical Turk users) were made to test the quality of the generated content, the level of satisfaction of the users and the usability of the prototype. As users are more familiar to Wikipedia and its infoboxes than KBs and RDF, they were asked to use the prototype to generate Wikipedia infoboxes.

4.2.1 Evaluating the Quality of the Created Content

In the first Wikipedia *edit-a-thon*, “Editatón por la visibilidad de las mujeres de Aragón” (Edit-a-thon for the visibility of women in Aragon) organized by Wikimedia Spain, the City Council of Zaragoza (Spain), and different associations[1], 7 users, without previous experience in editing Wikipedia, created 13 infoboxes (with an average of 8.30 properties, and 27.7% values linked to existing entities in DBpedia) using the developed prototype. The goal of this preliminary test was simply to obtain feedback about the usage of the prototype. Users highlighted that creating the infobox with the prototype was easy but required additional functionalities such as adding an image to the infobox (later incorporated). Detailed data about the created infoboxes in this event can be found on Appendix C.2.

The second Wikipedia edit-a-thon, “Wikinformática 2016” organized by the University of Zaragoza and Wikimedia[2], was a competition where 32 teams of high-

[1]`https://es.wikipedia.org/wiki/Wikipedia:Encuentros/Editat%C3%B3n_por_la_visibilidad_de_las_mujeres_de_Arag%C3%B3n`

[2]`http://eules.unizar.es/wikinformatica/edicion2016`

school students (187 in total) were asked to create Wikipedia articles about relevant Spanish women working on ICT[3]. The participants created 66 infoboxes using the prototype. In contrast, in the first edition of the same edit-a-thon[4], without the developed system, only one infobox was created. Users needed around 60 seconds on average to fill in each property. The infoboxes created for the scientist category (Figure 4.7) contained on average 9.34 properties, which is a higher number of properties than the average number of DBpedia properties for scientists (6). Besides, 51.18% of the values included in all the infoboxes were linked to existing entities (50.5% in the case of the infoboxes for scientists)[5]. The correctness of values of the infoboxes created in this event was informally verified as well; the results indicated that, on average, only 0.91% of the values were incorrect (e.g., Ada Byron was born in Madrid instead of London) and 6.61% of the values were correct but imprecise (e.g., Ada Byron was born in England instead of London). Given that the subjects were creating infoboxes for the first time, these rates of errors are quite low.

Finally, the ranking of properties or attributed offered to the users was evaluated computing the *generalized Kendall's tau* [12]. Results showed that the order of properties generated by the system is very similar to the order of properties that the users filled in, meaning that they find the ranking appropriate. Detailed data about the ranking evaluation and the created infoboxes in the second edition of "Wikinformática" last event can be found on Appendix C.3.

Analysing the created infoboxes in both events, it was discovered that most of the missing links were caused by entities which do not exist in the DBpedia version used in the test and an error in the prototype. This error was triggered when users copied and pasted the values of the properties instead of inputting them or selecting them from the list of suggestions. The prototype was fixed and improved for the remainder of tests to automatically link a value to an existing entity if the label is the same.

4.2.2 Comparing the System with the Current Wikipedia Mechanism

The goal of this test was to compare the prototype against the current mechanism for creating infoboxes in Wikipedia. To do so, a batch of tests was set in Amazon Mechanical Turk [24], a crowdsourcing platform where requesters can hire users (*turkers*) to perform tasks. Users were required to create the infobox of a given Wikipedia page using the prototype and the Wikipedia mechanism (a simulation developed to log all their interactions). One half of the turkers started with the prototype and the other half of them started with the Wikipedia mechanism. The turkers had 10 minutes to use each system and then had to fill in a survey with questions to compare them. 11 users participated in the tests: 55% of them were between 25 and 40 years old with high–school education (64%). Besides, 64% were

[3]Information and communications technology.

[4]`http://hendrix-http.cps.unizar.es/dokuwiki/doku.php/start`

[5]To compute the percentage of linked values literals as well as values which cannot be matched with any entity in the KB were discarded.

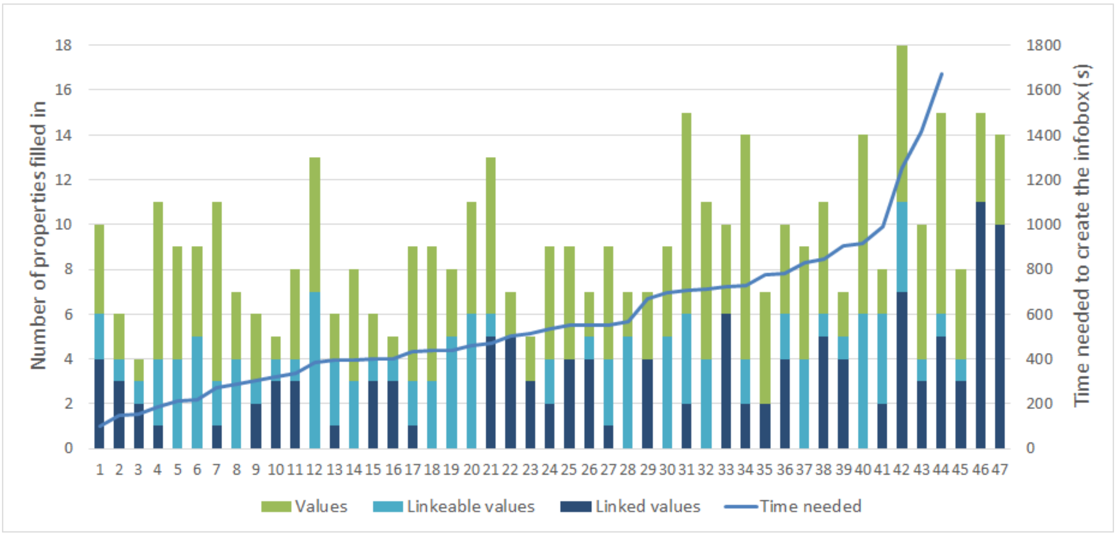


Figure 4.7: Quality of infoboxes for scientist category created with the prototype.

very familiar using Wikipedia and the rest affirm to occasionally use it. None of the users previously had edited a Wikipedia articles or created infoboxes.

The results of the test showed that 73% of the users found that the developed system was easier to use than the traditional Wikipedia interface. Users highlighted benefits of the prototype such as being simpler, helpful in filling and searching information, better looking, and more comfortable. Users also highlighted certain user interface disadvantages such as slightly confusing layout and issues when typed text does not exactly match with a suggested value. Moreover, 64% of the users also considered the developed system faster. Although some users declared that it was slower, this was contradicted by collected data. On average, to fill a property, users required 7 seconds less using the developed system than using the Wikipedia system. With the Wikipedia system, users filled 10.18 properties on average, but none of the values of those properties was linked to existing Wikipedia pages. The average number of properties filled with the prototype was 16.09 (a 58% of improvement over Wikipedia) and half of these values were linked to DBpedia resources. Regarding the precision and correctness of the information, although the developed system had less imprecise data (3.95%, the half of the Wikipedia mechanism), more incorrect values were introduced (2.26%, while this number decreases to a 1.13% using Wikipedia); analysing the data, we can see that those errors must be caused by the desire of the users to finish the test as soon as possible; although they were given the page from which the information had to be extracted, some users did not stop to think what suggested values would be the more appropriates ones (e.g., selecting Austrian Empire instead of Austria for a birth place). So, we can conclude that, besides the turkers had preference on finishing the task quickly, the use of the developed prototype produced quality and linked data, being easier and faster to use than the Wikipedia mechanism. Detailed data about this test can be found on Appendix C.4.

Chapter 5

Conclusions

In this chapter, general conclusions of the project are commented. Information about the planning and schedule of the project is shown by using a Gantt diagram. Besides, some interesting lines of future work are proposed. Finally, a personal assessment of the author is given.

5.1 General Conclusions

As commented before, the population of KBs is a difficult task, and some problems can arise (errors, inconsistencies, ...). A solution to create semantic templates with relevant properties by using semantic and statistical information from a KB, and to enforce semantic constraints on the values of the properties to be filled has been developed. The system allows to export the generated content in RDF or infobox format. Also, some tests have been carried out to evaluate the improvement on generated content, user satisfaction and usability. In more detail, the contributions of this project are:

1. Extraction of relevant attributes, ranges, and suggestions:

    - The needed operations and algorithms for extracting relevant information from a KB and its associated ontology have been analysed and optimized to develop a system that generates semantic templates efficiently, taking into account statistical and semantic information.
    - The presented approach does not depend on any specific KB; it can be configured easily even for people with no programming aptitudes, so it can work with almost every RDF KB. Also, due to the modular structure of its code, it is scalable (i.e., it can be extended or changed easily).
    - The system and algorithms have been tested in different scenarios, showing good results regarding the correctness and relevancy of the information introduced by the user, as well as the easiness of usage and the time required to create information about an entity.
    - A demo paper has been accepted in the *XXI Jornadas de Ingeniería del Software y Bases de Datos* (*JISBD*) with very good evaluations.

Also, a full research paper has been submitted to the 15th International Semantic Web Conference (ISWC 2016), which is under review and the notification is expected by the end of June.

2. Development of a fully functional web application:

   - A complete web application that allows users to use the system through their Internet browsers has been developed. It shows the generated templates in a friendly interface that helps all kind of users to generate data for a certain KB without knowing its inner structure or semantics.
   - The web application has been tested with real users in order to collect suggestions and criticisms from the public, hence improving it later. The evaluation demonstrates that the approach helps no-experts users to create complete and accurate data for a KB.

Therefore, the initial goals for the project have been covered.

5.2 Methodology

For the planning and development of the project, the Scrum[1] agile methodology has been applied. It is based on an incremental strategy in which the development is divided in short iterations or sprints. Instead of doing a complete schedule at the beginning of the project, the tasks for a sprint are defined before it starts. At the end of every sprint (from one week to one month), a review is done and a functional product is ready. Weekly meetings with the advisors to review the progress and solve problems have been done.

5.3 Project Schedule

The project took 15 months, starting in April 2015 and ending in June 2016. This can be explained because the project was performed part time along with some degree classes.

The project was divided in four parts (see Figure 5.2 for details about the tasks planned on the project): the first one was the study of documentation available about the project and the simple existing research prototype developed in the Research Group of Distributed Information Systems (SID) [37]. The second part, the most important, was the analysis and development of a new and better prototype with the acquired information of the previous one. It started with the analysis, design and implementation of a basic system, and then it was progressively improved,

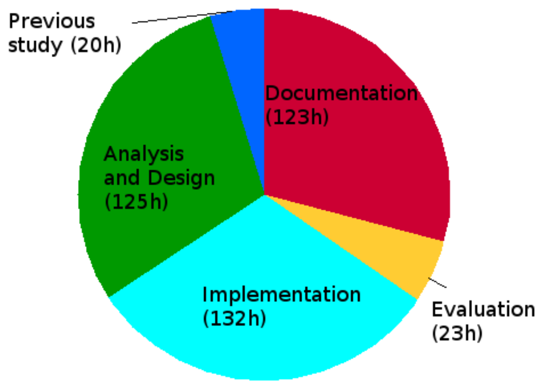
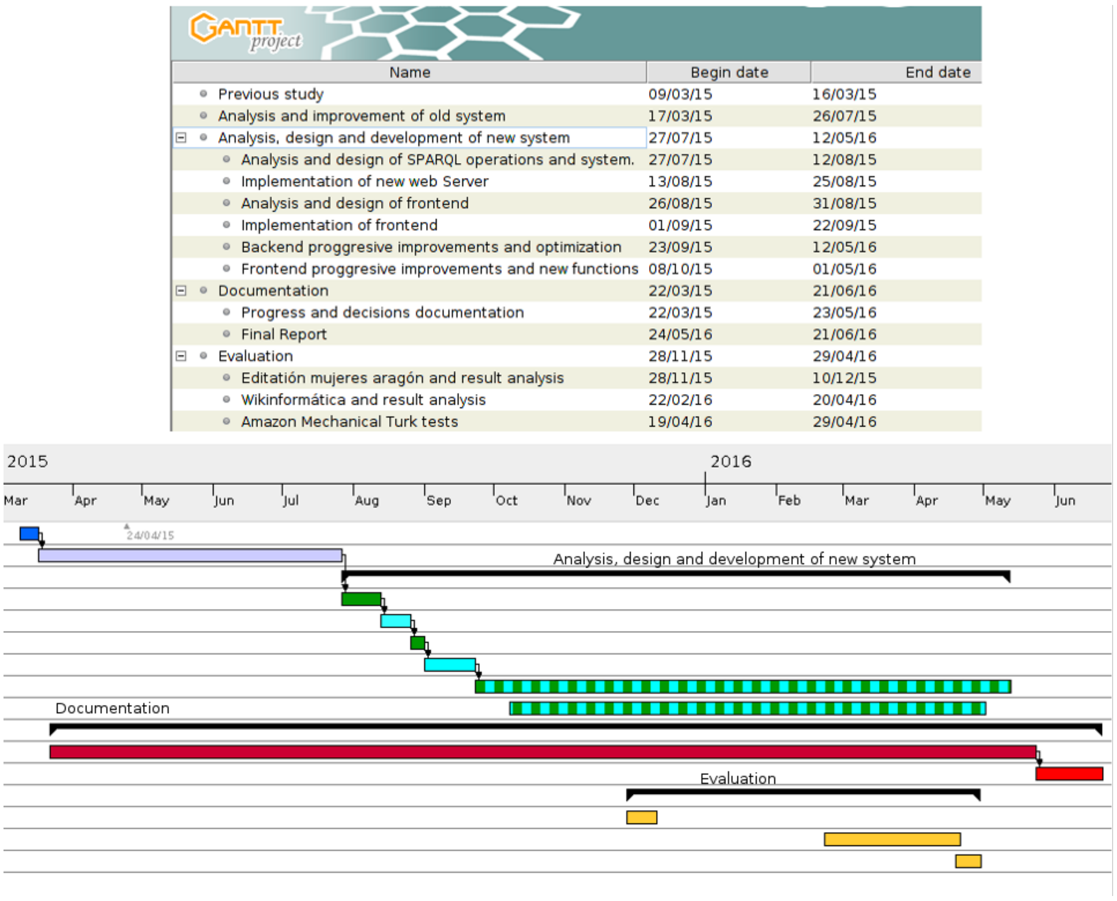


Figure 5.1: Project hours

[1]https://en.wikipedia.org/wiki/Scrum_(software_development)

designing and developing new features and optimizations. Documentation about decisions made, how the system works and technical references were being developed along the entire project, concluding in this Final Report elaboration. Besides, some evaluations and tests were made during the development of the system. A total of 423 hours were spent in this project, as seen in Figure 5.1.

Some problems caused the schedule to be modified. One of them was the lack of structure and bad performance of the old prototype used. It was planned to improve that early prototype and take it as a base, but finally the whole prototype had to be redesigned and developed. As the new prototype was growing, I also realized that some aspects could be changed or improved, thus completely changing some parts of the code. In addition, I needed to understand the context of the project, which involved studying concepts related to the Semantic Web, ontologies, and Wikipedia.



Name	Begin date	End date
Previous study	09/03/15	16/03/15
Analysis and improvement of old system	17/03/15	26/07/15
Analysis, design and development of new system	27/07/15	12/05/16
Analysis and design of SPARQL operations and system.	27/07/15	12/08/15
Implementation of new web Server	13/08/15	25/08/15
Analysis and design of frontend	26/08/15	31/08/15
Implementation of frontend	01/09/15	22/09/15
Backend proggresive improvements and optimization	23/09/15	12/05/16
Frontend proggresive improvements and new functions	08/10/15	01/05/16
Documentation	22/03/15	21/06/16
Progress and decisions documentation	22/03/15	23/05/16
Final Report	24/05/16	21/06/16
Evaluation	28/11/15	29/04/16
Editatión mujeres aragón and result analysis	28/11/15	10/12/15
Wikinformática and result analysis	22/02/16	20/04/16
Amazon Mechanical Turk tests	19/04/16	29/04/16

Figure 5.2: Gantt diagram: Project planning.

5.4 Future Work

As future work, some interesting tasks (out of the scope of this project) are proposed:

1. To improve data processing times by using parallel and distributed processing techniques of large data sets (like the MapReduce technique used in

Hadoop[2]).

2. To use the system to help fixing errors and inconsistencies in existing KB by detecting values that might be semantically or statistically incorrect or inaccurate.

3. To integrate the tool with the Wikipedia and Wikimedia environment to help users to create Wikipedia Infoboxes and to populate WikiData.

Related to the last point, it is worth noting that my advisors and I are in touch with people from Wikimedia, and we have applied for a Wikipedia grant to achieve this task.

5.5 Personal Assessment

This work has introduced me into research, developing a project from the beginning. So, I consider that, thanks to this project, I have improved my skills as a computer scientist and software engineer. I have not only deepened my knowledge about technologies and techniques learnt during the degree, but I have also discovered new fields such as Semantic Web and Linked Data. In the personal level, it has taught me how to manage time in a better way, how to work in a big project and document it. In addition, I had the opportunity of writing research articles with researcher from University of Maryland - Baltimore County, and General Electric. Technically, I have learnt a lot about ontologies, RDF and the SPARQL query language, and I have discovered the great possibilities and future that this technologies have. I have also greatly improved my knowledge of the AngularJS framework, thus making me capable of developing a complex web application.

To sum up, the accomplishment of this project has been a great experience, both academically and personally, making me apply and integrate a lot of knowledge acquired during the degree and convincing me to continue my research formation. I am very satisfied with the developed project because, after a very hard work and difficulties, its result is a very useful, usable and complete system.

[2]`http://hadoop.apache.org/`

Bibliography

[1] Amardeilh, F.: Ontopop or how to annotate documents and populate ontologies from texts. In: Proceedings of the ESWC 2006 Workshop on Mastering the Gap: From Information Extraction to Semantic Representation. pp. 1613–0073 (2006)

[2] Baader, F.: The description logic handbook: theory, implementation, and applications. Cambridge University Press New York (2003)

[3] Berners-Lee, T.: Information management: A proposal (1989)

[4] Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts pp. 205–227 (2009)

[5] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - a crystallization point for the web of data. Web Semantics: Science, Services and Agents on the World Wide Web 7(3), 154–165 (2009)

[6] Bos, B., Celik, T., Hickson, I., Lie, H.W.: Cascading style sheets level 2 revision 1 (css 2.1) specification. w3c candidate recommendation. World Wide Web Consortium (W3C) (2009)

[7] Brickley, D., Miller, L.: Foaf vocabulary specification 0.98. Namespace document 9 (2012)

[8] Date, C.J., Darwen, H.: A Guide To SQL Standard, vol. 3. Addison-Wesley Reading (1997)

[9] D.Brickley, R.G.: RDF vocabulary description language 1.0: Rdf schema. W3C Recommendation (2004)

[10] ECMA International: Standard ECMA-262 - ECMAScript Language Specification. 5.1 edn. (June 2011), `http://www.ecma-international.org/publications/standards/Ecma-262.htm`

[11] Euzenat, J., Shvaiko, P., et al.: Ontology matching, vol. 18. Springer (2007)

[12] Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. In: 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003). pp. 28–36 (2003)

[13] Fetahu, B., Markert, K., Anand, A.: Automated news suggestions for populating Wikipedia entity pages. In: 24th ACM Int. on Conf. on Information and Knowledge Management. pp. 323–332 (2015)

[14] Gosling, J.: The Java language specification. Addison-Wesley Professional (2000)

[15] Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. International journal of human-computer studies 43(5), 907–928 (1995)

[16] Harris, S., Seaborne, A., Prud'hommeaux, E.: Sparql 1.1 query language. W3C Recommendation 21 (2013)

[17] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E., Pfeiffer, S.: Html5. w3c recommentation. World Wide Web Consortium (W3C) (2014)

[18] Horridge, M., Bechhofer, S.: The OWL API: a Java API for working with OWL 2 ontologies. In: Proceedings of the 6th International Conference on OWL: Experiences and Directions-Volume 529. pp. 49–58. CEUR-WS. org (2009)

[19] Jupp, S., Horridge, M., Iannone, L., Klein, J., Owen, S., Schanstra, J., Wolstencroft, K., Stevens, R.: Populous: a tool for building OWL ontologies from templates. BMC bioinformatics 13(1), 1 (2012)

[20] Klyne, G., J.Carroll, J.: Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation (2004)

[21] Lange, D., Böhm, C., Naumann, F.: Extracting structured information from Wikipedia articles to populate infoboxes. In: 19th ACM Int. Conf. on Information and Knowledge Management. pp. 1661–1664 (2010)

[22] O'reilly, T.: What is web 2.0: Design patterns and business models for the next generation of software. Communications & strategies (1), 17 (2007)

[23] O'reilly, T.: Web 2.0: compact definition (2005)

[24] Paolacci, G., Chandler, J., Ipeirotis, P.G.: Running experiments on Amazon Mechanical Turk. Judgment and Decision making 5(5), 411–419 (2010)

[25] Shearer, R., Motik, B., Horrocks, I.: Hermit: A highly-efficient owl reasoner. In: OWLED. vol. 432, p. 91 (2008)

[26] Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: principles and methods. Data & knowledge engineering 25(1), 161–197 (1998)

[27] Sultana, A., Hasan, Q.M., Biswas, A.K., Das, S., Rahman, H., Ding, C., Li, C.: Infobox suggestion for Wikipedia entities. In: 21st ACM Int. Conf. on Information and Knowledge Management. pp. 2307–2310 (2012)

[28] Tim Berners-Lee, J.H., Lassila, O.: The Semantic Web. Scientific american 284.5, 28–37 (2001)

[29] Vrandečić, D., Krötzsch, M.: Wikidata: A free collaborative knowledge base. Communications of the ACM 57(10), 78–85 (2014)

[30] Web, S.: Creating semantic web contents with protege-2000 (2001)

[31] Welty, C., McGuinness, D.L.: OWL web ontology language guide. W3C Recommendation (2004)

[32] Wick, M., Boutreux, C.: Geonames. GeoNames Geographical Database (2011)

[33] Wolstencroft, K., Owen, S., Horridge, M., Krebs, O., Mueller, W., Snoep, J.L., du Preez, F., Goble, C.: Rightfield: embedding ontology annotation in spreadsheets. Bioinformatics 27(14), 2021–2022 (2011)

[34] Wu, F., Weld, D.S.: Automatically refining the Wikipedia infobox ontology. In: 17th Int. Conf. on World Wide Web (WWW 2008). pp. 635–644 (2008)

[35] Yus, R., Bobed, C., Esteban, G., Bobillo, F., Mena, E.: Android goes semantic: Dl reasoners on smartphones. 2nd International Workshop on OWL Reasoner Evaluation (2013)

[36] Yus, R., Ilarri, S., Mena, E.: Real-time selection of video streams for live TV broadcasting based on query-by-example using a 3D model. Multimedia Tools and Applications 74(8), 2659–2685 (2015)

[37] Yus, R., Mulwad, V., Finin, T., Mena, E.: Infoboxer: Using statistical and semantic knowledge to help create wikipedia infoboxes. In: 13th International Semantic Web Conference (ISWC 2014). vol. 1272, pp. 405–408. CEUR-WS (2014)

Appendices

Appendix A

Analysis and Design of the Prototype of the System

In this appendix, the analysis and design process is shown using a list of requirements, use cases, and packages, sequence, and deployment diagrams. Moreover, the data model used in the MySQL database is detailed.

A.1 Analysis of the Prototype

The requirements of the developed system, shown in Table A.1, are divided into Functional Requirements (FR) and Non-functional Requirements (NFR).

Functional Requirements	
Code	Description
1	To allow the user to select one or more category.
2	To allow the user to select simple or expert mode.
3	To generate semantic templates for the selected categories, from a KB and associated ontology.
4	To display the semantic template to the user.
4.1	To display a list of properties ordered by popularity.
4.2	If “Expert mode” selected, to display list of ranges for each property, ordered by popularity.
4.3	To provide value suggestions to the user when he/she types a value.
4.4	If “Expert mode” selected, to show detailed statistical information.
4.5	To allow the user to introduce one or more value for a property.
4.5	To easy the user the process of searching information about a property in Google.
5	To export the generated content in RDF format.
6	To export the generated content in Wikipedia infobox format.
Non-functional Requirements	
Code	Description
1	The “GUI” should be simple and intuitive.
2	The “GUI” texts, properties, and ranges must be available in English and Spanish, allowing the user to change the language, despite of the used KB.
3	The system must maintain a cache of results.
4	The system must gather user interactions data to later analyze it manually.

Table A.1: Requirements of the developed system.

A.2 Design of the Prototype

In the following subsections the system is described using multiple diagrams (use case, package, sequence, deployment, and data model).

A.2.1 General Use Case

The developed prototype only has one global use case, which can be decomposed into several specific use cases (see Figure A.1).

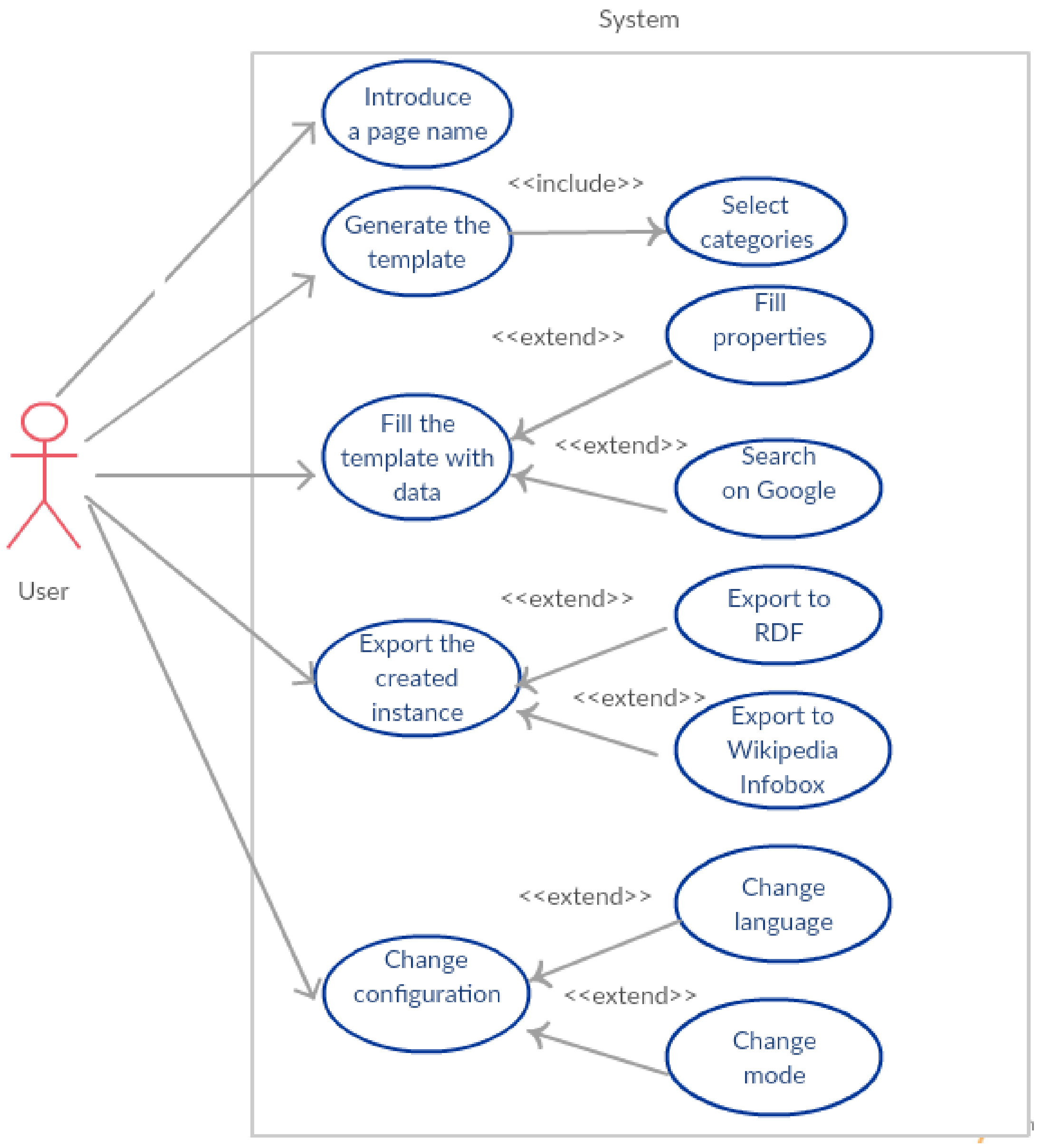


Figure A.1: Use case diagram: General use case.

Use case: Select categories
Description: To select one or more categories from the provided list.
Actors: User and System.
Pre-conditions: -
Normal flow: 1.- The user clicks on the category input. 2.- The user starts typing and the shown categories are filtered. 3.- The user selects one category.
Alternative flow: 4.- The user types again and selects more categories.
Post-conditions: A category or categories have been selected.

Table A.3: Use case: Select categories.

Use case: Generate the template
Description: To generate a semantic template from a list of categories.
Actors: User and System.
Pre-conditions: At least one category has been selected.
Normal flow: 1.- The user clicks on the Load button. 2.- The system generates the template. 3.- The system shows the template to the user.
Alternative flow: -
Post-conditions: A template has been generated and shown.

Table A.5: Use case: Generate the template.

Use case: Fill a property
Description: To type one or more values of a property (for example, the birth place of a soccer player).
Actors: User and System.
Pre-conditions: The template has to be already generated.
Normal flow: 1.- The user clicks on the input box of the desired property. 2.- The user types a value. 3.- The user clicks on a suggestion if available.
Alternative flow: 4.- The user adds another value for the property.
Post-conditions: A value for a property has been typed.

Table A.7: Use case: Fill properties.

Use case: Introduce a page name
Description: To introduce a name for the template/page to fill (for example, if creating data of a person, it would be him/her name).
Actors: User and System.
Pre-conditions: -
Normal flow: 1.- The user clicks on the “Page name” input. 2.- The user types the template/page name.
Alternative flow: -
Post-conditions: The page/template name has been introduced.

Table A.9: Use case: Introduce a page name.

Use case: Search on Google
Description: To search information about a property on Google.
Actors: User and System.
Pre-conditions: The template has to be already generated and a page name has been introduced.
Normal flow: 1.- The user clicks on the search button on the desired property. 2.- A browser windows is open, with a Google search of that property and instance. 3.- The user looks up information about the property.
Alternative flow: -
Post-conditions: Information about the property value for an instance has been searched.

Table A.11: Use case: Search on Google.

Use case: Export to RDF
Description: To export the generated data in RDF format.
Actors: User and System.
Pre-conditions: One value for at least one property and the page name have been introduced.
Normal flow: 1.- The user clicks on the "Export to RDF" button 2.- The RDF code is generated and shown to the user 3.- The user copies the code and inserts it into a KB, for example using a Fuseki Endpoint.
Alternative flow: -
Post-conditions: RDF code has been generated.

Table A.13: Use case: Export to RDF.

Use case: Export to Wikipedia infobox
Description: To export the generated data in Wikipedia infobox format.
Actors: User and System.
Pre-conditions: One value for at least one property and the page name have been introduced.
Normal flow: 1.- The user clicks on the “Generate infobox code” button 2.- The infobox code is generated and shown to the user 3.- The user copies the code and pastes it into a Wikipedia page.
Alternative flow: -
Post-conditions: Wikipedia infobox code has been generated.

Table A.15: Use case: Export to Wikipedia infobox.

Use case: Change language
Description: To change the interface and data language.
Actors: User and System.
Pre-conditions: -
Normal flow: 1.- The user clicks on “Configuration” button at the top bar of the web prototype. 2.- The user selects the new language. 3.- The users clicks on “Save and apply” and the language is changed.
Alternative flow: -
Post-conditions: Language has been changed.

Table A.17: Use case: Change language.

Use case: Change mode
Description: To change the interface mode between simple and expert mode.
Actors: User and System.
Pre-conditions: -
Normal flow: 1.- The user clicks on "Configuration" button at the top bar of the web prototype. 2.- The user selects the new mode. 3.- The users clicks on "Save and apply" and the mode is changed.
Alternative flow: -
Post-conditions: Mode has been changed.

Table A.19: Use case: Change mode.

A.2.2 Package Diagrams

This section uses package diagrams to show the packages and classes of the Backend, Frontend, and the relations between them. Also, a description of the responsibility of each package is presented.

Backend

In the next list and in Figure A.2 the packages, classes and libraries used by the Java Backend (web server) are detailed. The Backend does not offer a Graphical User Interface, as it is a task of the Frontend. The task of the Backend is to perform the needed operations to generate semantic templates, and offer them through an HTTP interface.

1. **Http** is a package that contains the classes that offer the HTTP interface; they are the entry point to the application.

2. **Operations** package is composed of multiple classes, each one responsible of executing one important operation, such as obtaining the list of properties for certain categories, the list of ranges, etc. Those clases are also responsible of maintaining a cache of its results. If not previously calculated, they use the "ontology" and "dataObtaining" packages to do so.

3. **Suggestions** package contains operations needed to provide real time suggestions to the user. As the suggestions are stored in a MySQL database, a class that manages the interaction with it is present; it uses the JDBC driver.

4. **StatsDatabase** package only contains a class responsible for storing actions and statistics from user interactions to a MySQL database. It uses the JDBC drive.

5. **InfoboxCreation** package is responsible of the generation of Wikipedia Infobox code, mapping the received data from the client to a Wikipedia template.

6. **Ontology** package is in charge of the interactions with the ontology or schema associated to the KB, such as obtaining super classes of a certain class, semantic properties,...

7. **DataObtaining** is a package whose tasks are to interact with the SPARQL endpoint, to make queries, and to retrieve results.

8. **Translator** package is responsible for translating the operations result into another language, using the Yandex translator.

9. **Common** package is used by all the other developed packages, and it contains useful operations (string manipulations, commonly used functions,...) in the "utils" package and objects used to transport information along the system in the "dto" package.

10. **Application** class is in charge of the launch of the Server.

11. **Simulation** class is responsible for performing benchmarks and measuring times of the system.

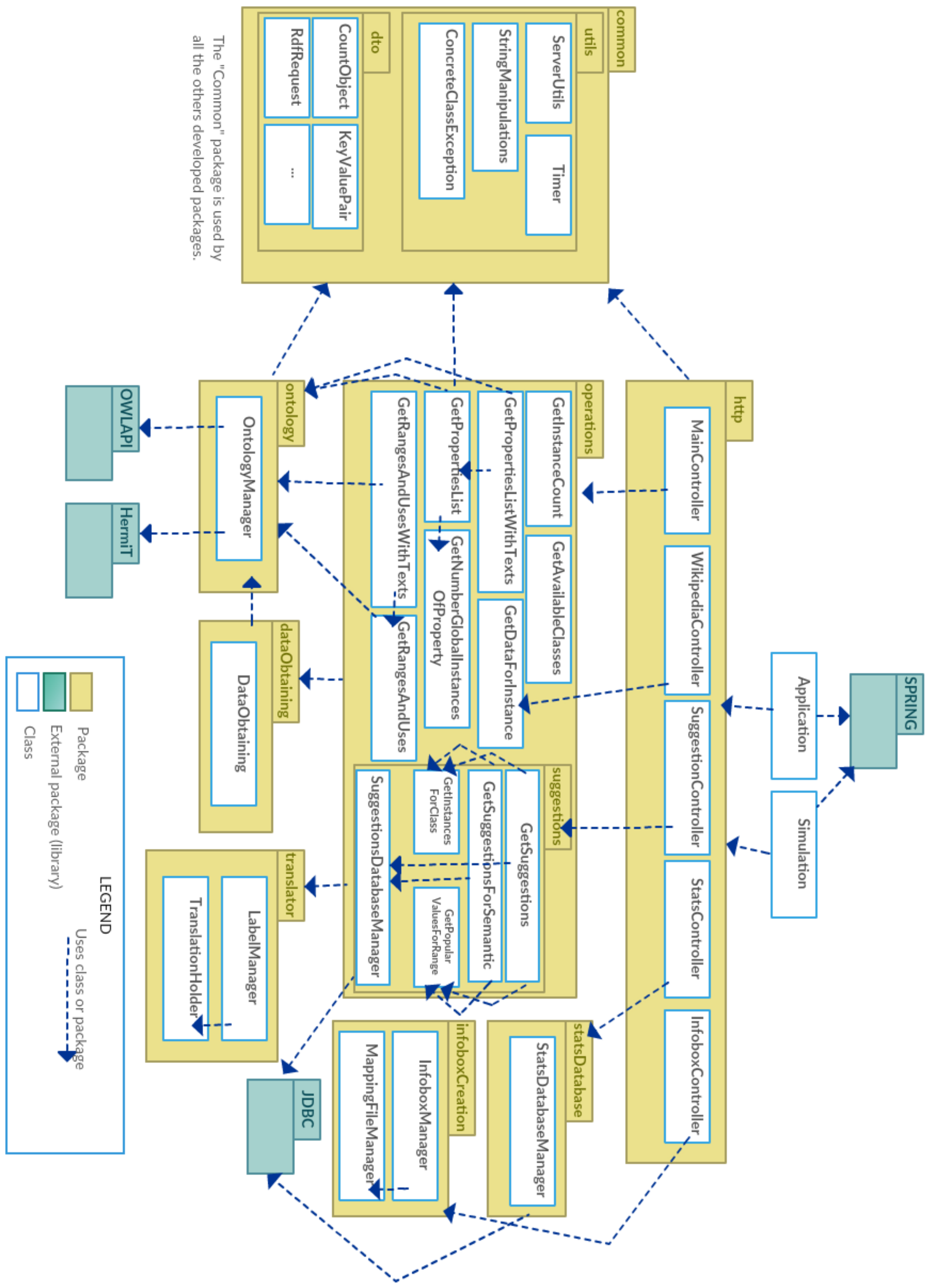


Figure A.2: Package diagram: Backend.

Frontend

In the next list and in Figure A.3 the packages, modules. and libraries used by the AngularJS Frontend are detailed. The main goal of the Frontend is to offer the user a graphical interface he/she can interact with. It interacts with the Backend to obtain the generated template and show it to the user.

1. **Index.html** is the first file executed by the browser, and is responsible for importing the external libraries and passing the control to AngularJS and the "App" module.
2. **Views** package contains all the HTML files rendered by the user Web browser. They also contain AngularJS instructions that modify that HTML before being rendered.
3. **Partials** package contains views that are rendered inside another view, like "modals" (windows that float over another content).
4. **Styles** package contains CSS stylesheets used to give style to the application.
5. **Scripts** package contains all the Javascript code in charge of the presentation logic:
   (a) **App** module is responsible for setting the configuration and initialize the application.
   (b) **Controllers** package contains Javascript files or modules that are in charge of user interactions (clicks, typed text) and modifying the data shown on the view according to it.
   (c) **Directives** package contains "directives", which are modules that encapsulate GUI components and their behaviour. For example, the drop-down list used to give suggestions to the user.
   (d) **Filters** package contains a unique module that implements functions to format data on the views (for example, changing a date format).
   (e) **Service** package contains several modules used to organize and share code across the application. Each module encapsulates some behaviour (such as obtaining the template for a category, saving statistics to the server, access to the Frontend configuration or handling a shown modal) and offers operations to access it.
   (f) **DataStructures** contains data structures and objects definitions in Javascript used in multiple parts of the application.

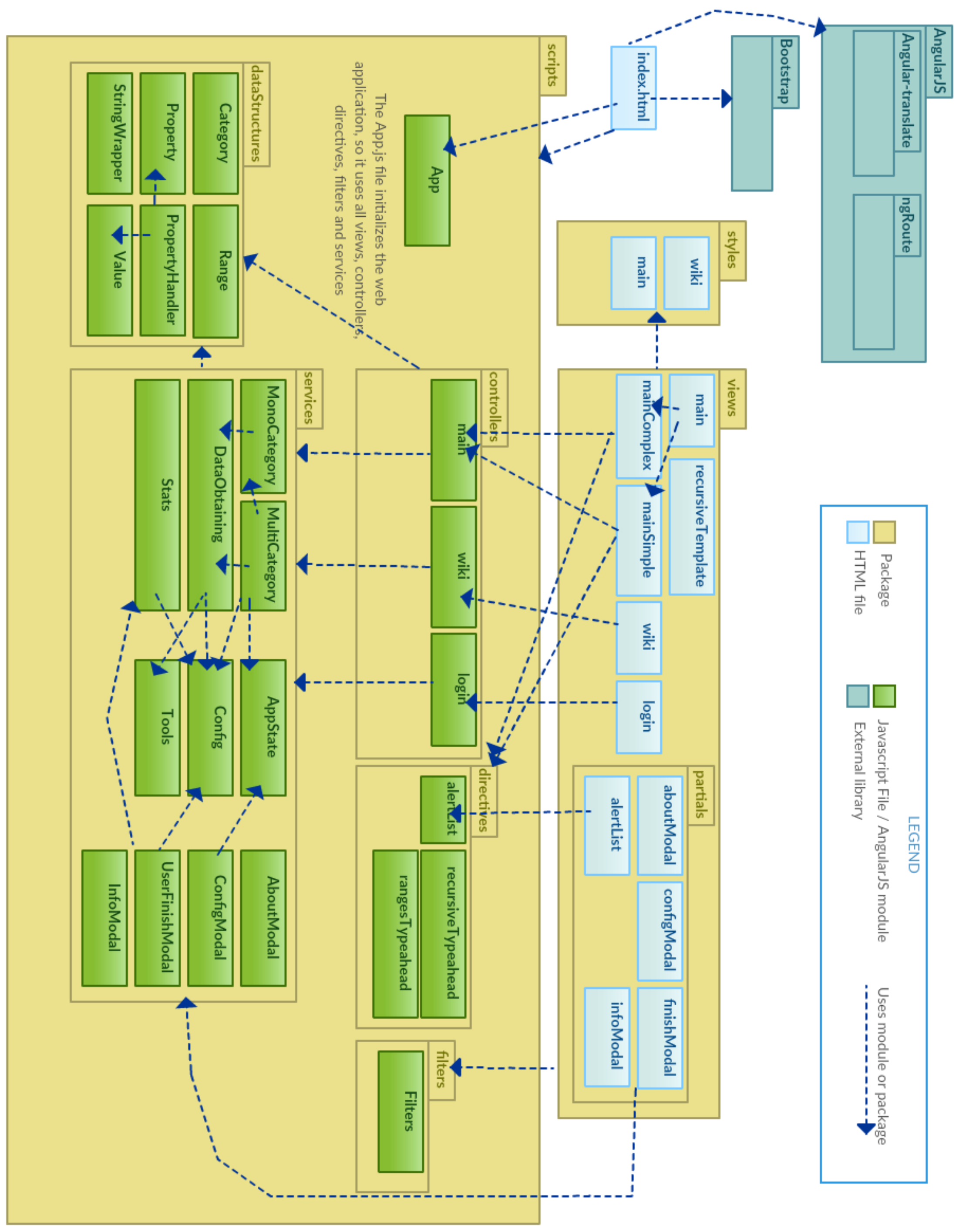


Figure A.3: Package diagram: Frontend.

A.2.3 Sequence Diagrams and Descriptions of Operations

In this section, the most relevant operations performed in the Backend are described, and for each one a sequence diagram is provided to show the interactions between its components.

"Instance count" operation

This operation (Figure A.4) calculates the number of instances that belong to, at least, one of the given categories. For that, it only uses the KB.

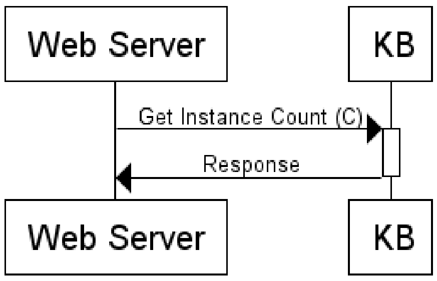


Figure A.4: Sequence diagram: "Instance count" operation.

"Property list" operation

This operation (Figure A.5) returns a list of relevant attributes obtained both semantically and statistically. Each statistical attribute comes with the number of instances of the given categories that use it at least one time. For the semantic attributes, it is provided the number of instances in the whole dataset that use it and the semantic range according to the ontology.

The result is obtained by first querying the KB for the properties and its frequency. Then, the semantic attribute list is obtained from the ontology. The two lists are mixed and for each semantic property that is not in the statistic property list, its count in the whole KB and its semantic range are retrieved. Finally, the label and comment of each property is retrieved from the ontology if available.

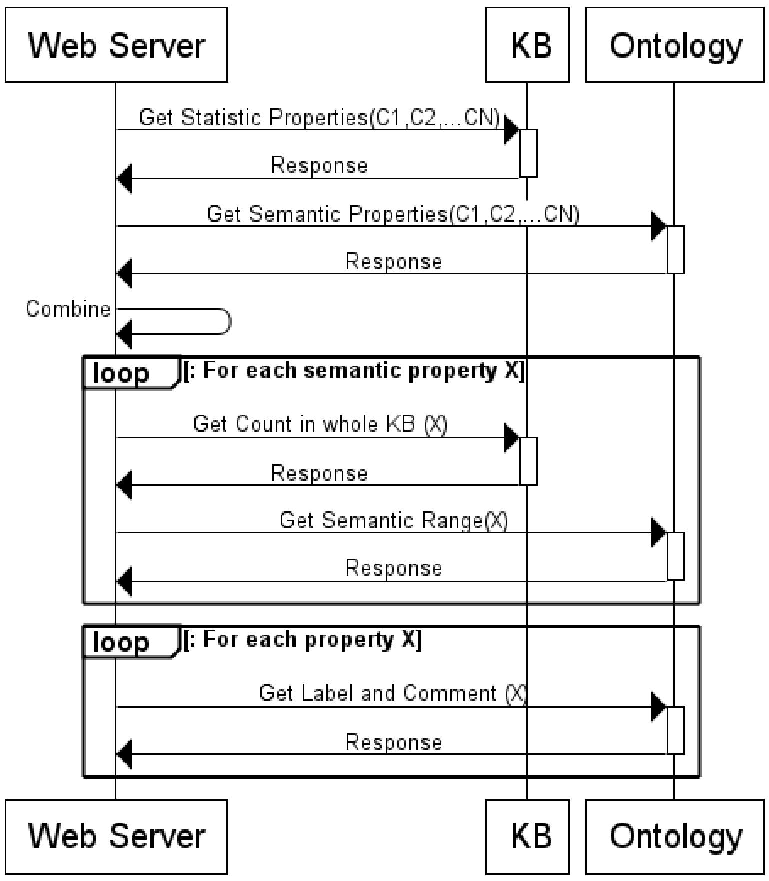

Web Server
KB
Ontology
Get Statistic Properties(C1,C2,...CN)
Response
Get Semantic Properties(C1,C2,...CN)
Response
Combine
loop
[: For each semantic property X]
Get Count in whole KB (X)
Response
Get Semantic Range(X)
Response
loop
[: For each property X]
Get Label and Comment (X)
Response
Web Server
KB
Ontology

Figure A.5: Sequence diagram: "Property list" operation.

"Range list" Operation

This operation (Figure A.6), obtains a list of ranges or type values for each statistical property of the given categories, with its frequency.

It is decomposed in multiple smaller operations, performed once for each instance that belongs to at least one of the categories. Those smaller operations obtain triples which have the instance as a subject, and every triple is accompanied by the types of its value. (e.g., if the given category is "SoccerPlayer", one of the triples would be "David_Beckham PlaysOn Manchester_City", and the types of the value would be "Soccer Team" and "Association"). Then, using the ontology reasoner, those types are processed to obtain the more concrete type or types of each value (in the previous example, it would be "Soccer Team"). That information is used to generate the returned list of ranges and a list of values used to provide suggestions. Finally, for every range, its label is obtained from the ontology if available.

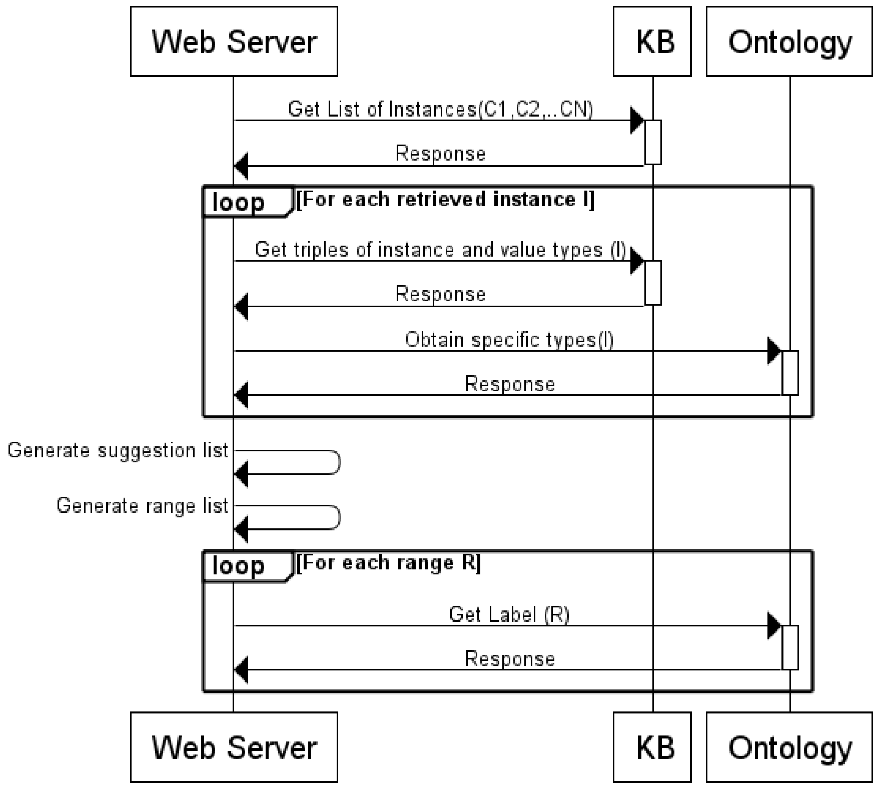


Figure A.6: Sequence diagram: "Range list" operation.

"Suggestion list" Operation

This operation (Figure A.7) provides suggestions of values for a range, property, and list of categories as the user types. For example, if the users wants to enter information about the Birth Place of a Soccer Player, and the ranges for that property are "City" and "Town", it will suggest cities and towns filtered by the entered text, ordered by how many times Soccer Players were born there.

1. First it checks if a table with all the instances with class the given range exists in the database (for example, if a table with all the instances of "Town" exists).

2. If it does not exist, that table is created and populated.

3. Then, it checks if a table with values whose class is the given range and used by instances of the given categories for the given property exists. (for example, if a table that contains towns used as a birth place by soccer players exists). This table contains also the frequency of each value.

4. If it does not exist, that table is created and populated from data already in file cache (as it was calculated by the "Range list" operation).

5. Finally, the two previous tables are queried and combined to obtain the desired suggestion list.

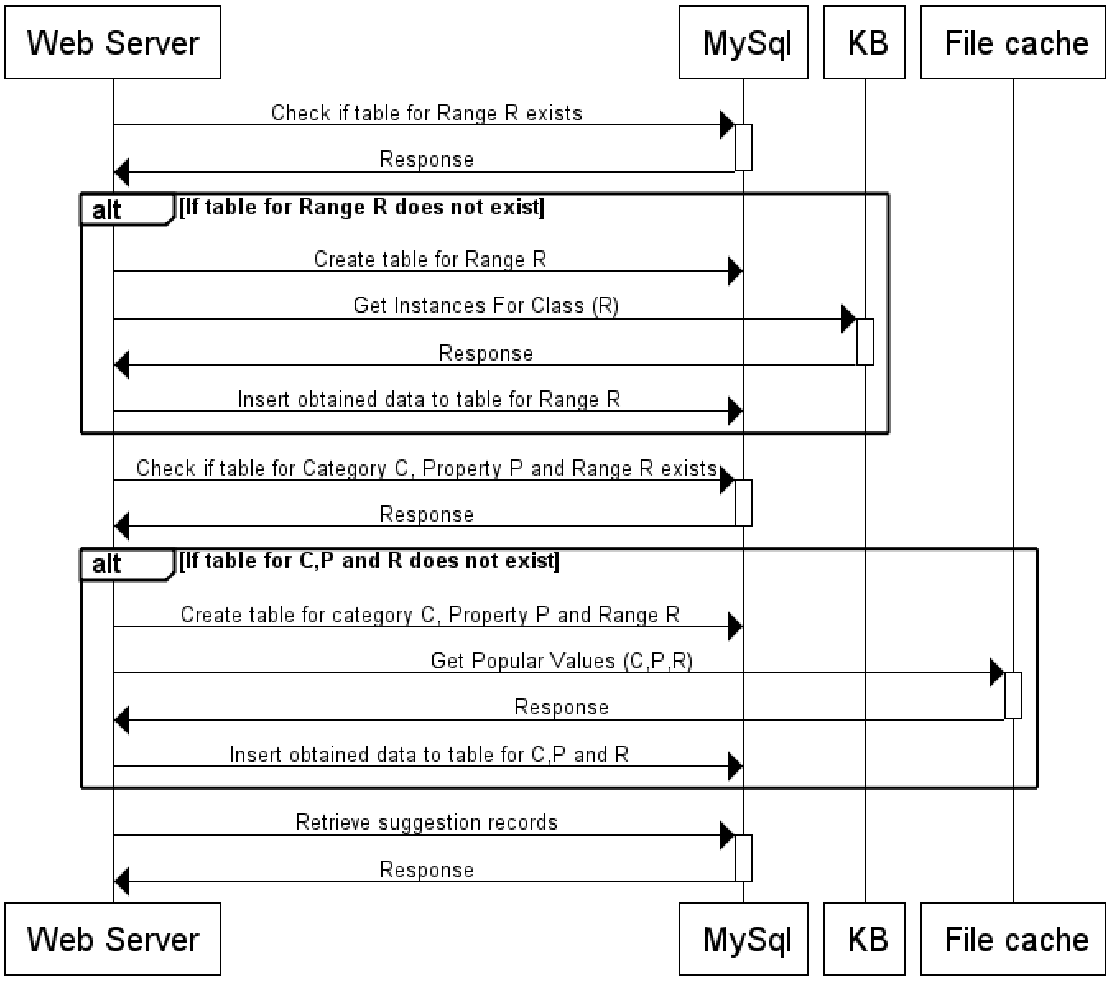

Web Server
MySql
KB
File cache
Check if table for Range R exists
Response
alt
[If table for Range R does not exist]
Create table for Range R
Get Instances For Class (R)
Response
Insert obtained data to table for Range R
Check if table for Category C, Property P and Range R exists
Response
alt
[If table for C,P and R does not exist]
Create table for category C, Property P and Range R
Get Popular Values (C,P,R)
Response
Insert obtained data to table for C,P and R
Retrieve suggestion records
Response
Web Server
MySql
KB
File cache

Figure A.7: Sequence diagram: "Suggestion list" operation.

"Class list" Operation

This operation (Figure A.8) returns a list of classes descendant of a given class, with the number of instances of every class, and filtered by an introduced text.

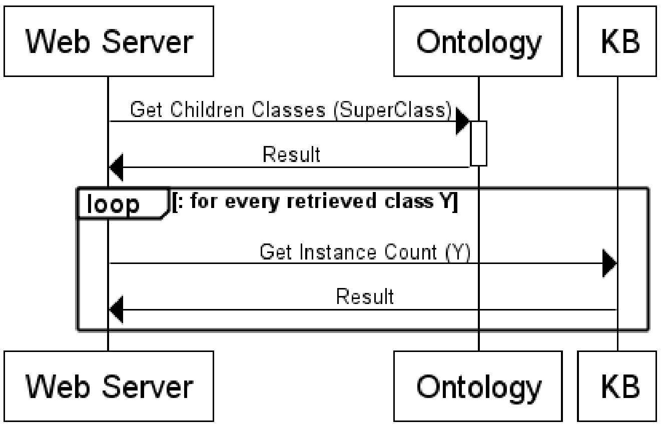


Figure A.8: Sequence diagram: "Class list" operation.

“Data for instance” Operation

Given a Wikipedia page URL, this operation (Figure A.9) finds an equivalent instance in the KB and returns a list of its properties, values for those properties, and for each value, its most concrete type or class.

To do so, it first converts the Wikipedia URL into an instance URI of the used KB. Then, it performs a process very similar to the process done in “Range list” operation, obtaining from the KB all the triples in which the subject is the instance, and the type for every triple’s value.

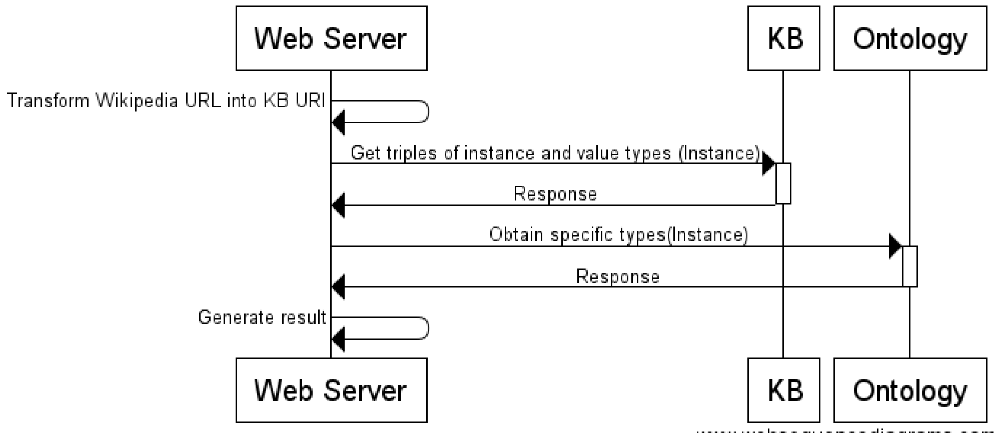

Web Server
KB
Ontology
Transform Wikipedia URL into KB URI
Get triples of instance and value types (Instance)
Response
Obtain specific types(Instance)
Response
Generate result
Web Server
KB
Ontology

Figure A.9: Sequence diagram: “Data for instance” operation.

A.2.4 Deployment Diagram

In this section a deployment diagram (Figure A.10) is shown. The nodes implied on the deployment are four:

1. **Web browser:** This node is on the client, and it executes the Frontend, previously served by the web Server. It interacts asynchronously with the web Server to ask for data and send the user interactions.

2. **Web server:** This is the main node. It executes the developed system on a Tomcat Server. Some of the components interact with the Frontend, while others interact with the SPARQL endpoint, MySQL database, ontology or file cache.

3. **Sparql Endpoint:** This is the server that hosts the KB. Queries are made via an HTTP interface.

4. **MySQL Database:** This server hosts the database used for storing suggestions and user interactions. It is queried by the web server using JDBC.

Note that, although the web server, SPARQL, and MySQL nodes are separated, two or the three of them can be hosted in the same machine.

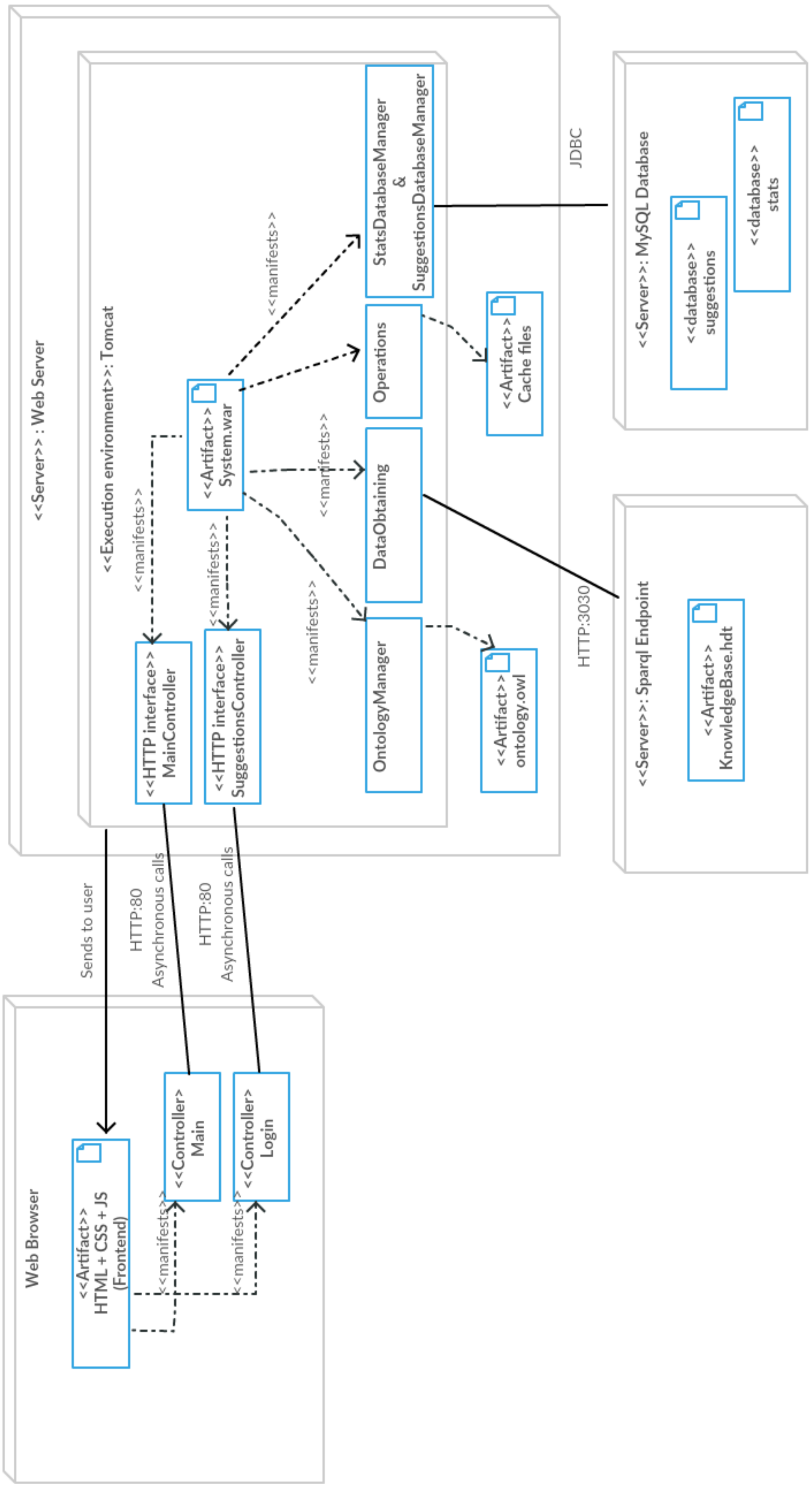

<<Server>> : Web Server
<<Execution environment>>: Tomcat
<<Artifact>> System.war
<<manifests>>
<<HTTP interface>> MainController
<<HTTP interface>> SuggestionsController
OntologyManager
DataObtaining
Operations
StatsDatabaseManager & SuggestionsDatabaseManager
<<Artifact>> ontology.owl
<<Artifact>> Cache files
JDBC
HTTP:3030
<<Server>>: MySQL Database
<<database>> suggestions
<<database>> stats
<<Server>>: Sparql Endpoint
<<Artifact>> KnowledgeBase.hdt
Sends to user
HTTP:80 Asynchronous calls
HTTP:80 Asynchronous calls
Web Browser
<<Artifact>> HTML + CSS + JS (Frontend)
<<Controller>> Main
<<Controller>> Login

Figure A.10: Deployment diagram: developed system.

A.3 MySQL Data Model

In this section, the data model of the two used relational databases is shown, with a brief description of their entities. As this databases are used for secondary tasks, its structure has not been thoroughly optimized nor normalized.

User Interactions Database

The data model of the database that stores all the users interactions is shown on Figure A.11. The entities that compose the model are:

- **Register**: it represents a record of one interaction of a certain user. It is identified by an auto-incremented field, and contains the session identifier (unique for each generated KB instance), the time stamp (the exact time when that interaction occurred) and three different strings: a subject, an action, and a value. The subject is the element target of the interaction, for example, a text box. The action is what happened with that element, for example, that it was clicked. The value is a feature of the element, such as the contained text.

- **Infobox**: this entity represents the final content generated by the user; every instance of it is associated with a certain register that indicates the action of saving the data. It contains the RDF and Infobox code.

- **Survey**: this entity represents the filled survey by a user of the prototype after having finished the creation of the content; every instance of it is associated with one register that indicates the action of saving the survey.

- **Summary**: this entity represents the summary of the interaction of one user; it contains data such as the session identifier, the username, and the seconds it took the user to create the content.

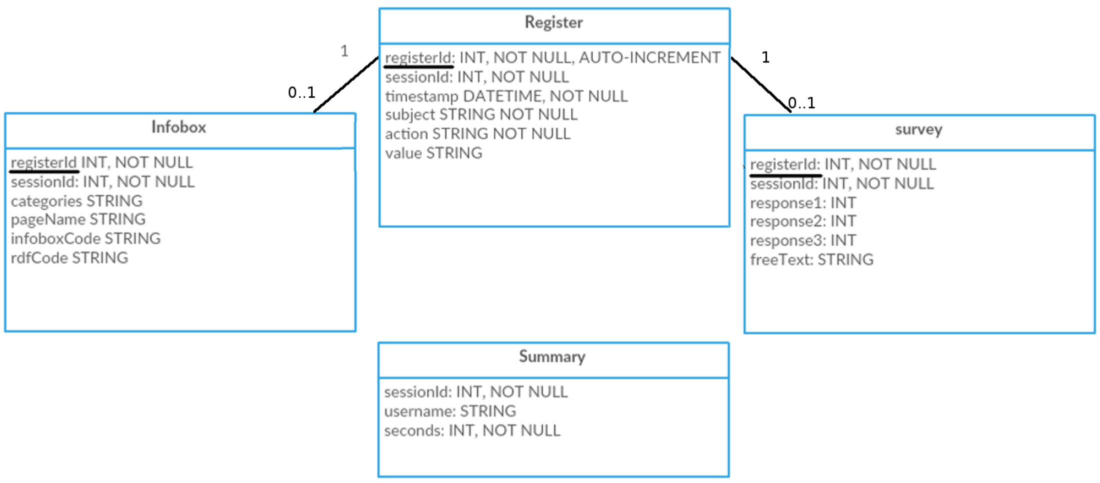


Figure A.11: Data Model of the users interactions database.

Suggestions Database

The data model of the database that stores the values that are suggested to the user when he types in the prototype is shown on Figure A.12. The entities that compose the model are:

- **rangeTableCLASS**: for every class or range whose values have been suggested, one table like this exists, where "CLASS" is replaced with the name of that range. For example, if suggestions of places have been provided, a table "rangeTablePlace" will exist. Those tables contain all the instances that belong to the given range in the KB, with its URI and the label that represents it.

- **CATEGORIES-PROPERTY-RANGE**: one table of this type is created for each combination of categories, property, and range of values that have been suggested. For example, if the user received suggestions of cities were soccer players were born, a table named "SoccerPlayerBirthPlaceCity" will exist. It contains the URIs and labels of instances that belong to the given range (similar to the previous entity), but only the ones that appear as a value on triples which subject is an instance of the selected categories, and property is the given one. Every value comes with the number of uses or frequency of that value.

These two kinds of tables are combined using join operations to generate the list of suggestion that is sent and shown to the user. For example, if a user needs suggestions of universities where actors studied, the tables "rangeTableUniversity" (that contains all the universities) and "ActorStudiedInUniversity" (that contains the universities where actors studied, with its frequency) will be combined. Also, as the "uri" attribute is used for joining the tables, it is indexed to improve the operation speed.

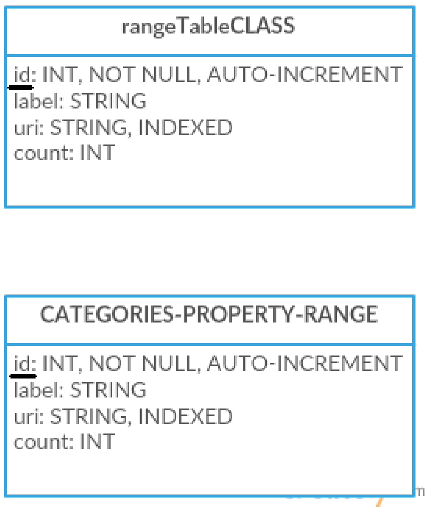


Figure A.12: Data Model of the suggestions database.

Appendix B

System Set-up, Configuration and Technical Aspects

This appendix contains the needed information to run the developed project in a personal computer, customize its functionalities (thanks to the configuration files), and load different KBs. Also, the exposed HTTP API and the performed SPARQL queries are described. The guides are designed for system administrators or technical experienced users, not for the final user that will actually create content with the prototype using a web browser.

B.1 Running the Prototype

In this section, the requirements and the steps to run the prototype are specified. The used KB on the guide is DBpedia 2015-04; prepared configuration files, an HDT file containing the KB and a *owl* file with the associated ontology are provided. However, this guide can be used with any other KB.

System Requeriments

In order to execute the prototype, the used computers have to meet some requirement. As the web server, the MySQL database and the SPARQL endpoint can be executed in different machines, the requirements for each one are detailed in Table B.2. The system has been tested using a unique host running Ubuntu 14.04 and Windows 7. A Linux system is recommended, as some limitations arise in the Windows set-up.

Download of the Required Files

The prototype source code and required files are hosted on a GitHub public repository[1]. To download it, the commands shown on Listing B.1 has to be executed on a terminal. This commands download the project files in the working directory of the terminal. Note that, if using Windows, due to a limitation on folder

[1]`https://github.com/ismaro3/infoboxer`

Requirement	WebServer	MySQL	SPARQL
OS	Linux, Windows XP or greater	Linux, Windows XP or greater	Linux, Windows XP or greater
RAM	At least 4GB	At least 2GB	At least 4GB
Java	Java 8 JRE	None	Java 8 JRE
GIT	v2.0 client or greater	v2.0 client or greater	v2.0 client or greater
Other software	NodeJS v.5.10.0 or greater if using Windows	MySQL v5.1 or greater	GIT

Table B.2: System requirements for running the prototype.

paths length, the *infoboxer* directory created when downloading the files can not be moved to a different location, so the command should be executed in the final path where the project is located.

```
git clone https://github.com/ismaro3/infoboxer.git
```

Listing B.1: Command for downloading the project code.

Also, the prepared DBpedia 2015-04 KB is uploaded to a Google Drive folder[2]. There, the HDT file containing the data and the associated ontology in *owl* format are available.

MySQL Database Configuration

In the host responsible of hosting the MySQL the following steps have to be followed:

1. Create a database user that will be used by the prototype's web Server. Don't grant too many privileges to it, as it could be dangerous if the system is compromised.

2. Create a database called "infoboxer" that will store the gathered user interactions. Its schema is defined in the *create_mysql_database.sql* file available in the previously downloaded prototype files.

3. Create a "suggestions" database that will store the values suggested to the user. Its schema is created automatically by the web server.

Both databases can be built importing the *create_mysql_database.sql* file into the database. The created user has to have permissions on the created databases to create, query and drop tables, and to insert data on them. Instructions on how to use MySQL can be found at the MySQL Reference Manual[3].

[2]`https://drive.google.com/open?id=0B1vXMuLLK1ybaU9zWXJPdGpVQnc`
[3]`http://dev.mysql.com/doc/refman/5.7/en/tutorial.html`

SPARQL Endpoint Execution

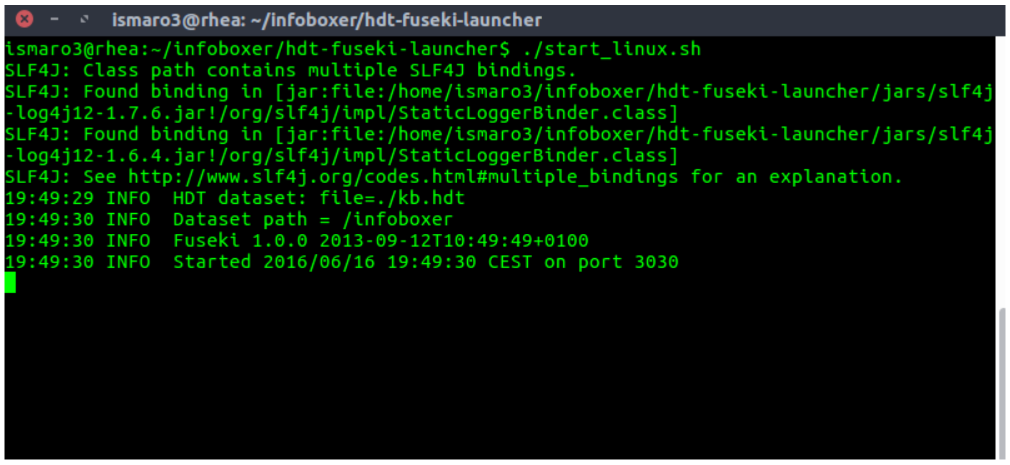
The SPARQL endpoint is located in the *hdt-fuseki-launcher* folder in the downloaded project files. That folder has to be moved to the machine where the SPARQL endpoint will execute. To execute the endpoint, the following steps have to be performed:

1. The KB in HDT format has to be placed inside the *hdt-fuseki-launcher* folder, with the name of *kb.hdt*. Please put there the *kb.hdt* DBpedia file downloaded from Google Drive. If a file with that name, or named *kb.hdt.index* already exists, please move or delete it.

2. The file *start_linux.sh* or *start_windows.bat* has to be executed, if using Linux or Windows respectively. If it is the first time that a given KB is used, an index will be built. It can take about 10 minutes with a KB of the size of DBpedia. A screenshot of the console output of the process is shown on Figure B.1.

3. The SPARQL endpoint will be running on port 3030.

For changing the port where the endpoint runs, or the name of the used HDT file, please edit the *start_linux.sh* or *start_windows.bat* file.



```
ismaro3@rhea: ~/infoboxer/hdt-fuseki-launcher
ismaro3@rhea:~/infoboxer/hdt-fuseki-launcher$ ./start_linux.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/ismaro3/infoboxer/hdt-fuseki-launcher/jars/slf4j
-log4j12-1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/ismaro3/infoboxer/hdt-fuseki-launcher/jars/slf4j
-log4j12-1.6.4.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
19:49:29 INFO  HDT dataset: file=./kb.hdt
19:49:30 INFO  Dataset path = /infoboxer
19:49:30 INFO  Fuseki 1.0.0 2013-09-12T10:49:49+0100
19:49:30 INFO  Started 2016/06/16 19:49:30 CEST on port 3030
```

Figure B.1: Screenshot: SPARQL Endpoint process output.

Converting a KB into HDT Format

Although in this guide the DBpedia KB in HDT format is provided, most of the available KBs are not in that format. Usually, they are available in *RDF/XML* or *RDF N-triples* format and have to be converted before being loaded into the SPARQL endpoint. This is the case of the KBs detailed in Appendix B.3. To convert a KB into HDT format the following steps have to be performed:

1. Navigate to the *hdt-tools/bin* directory of the downloaded project files.

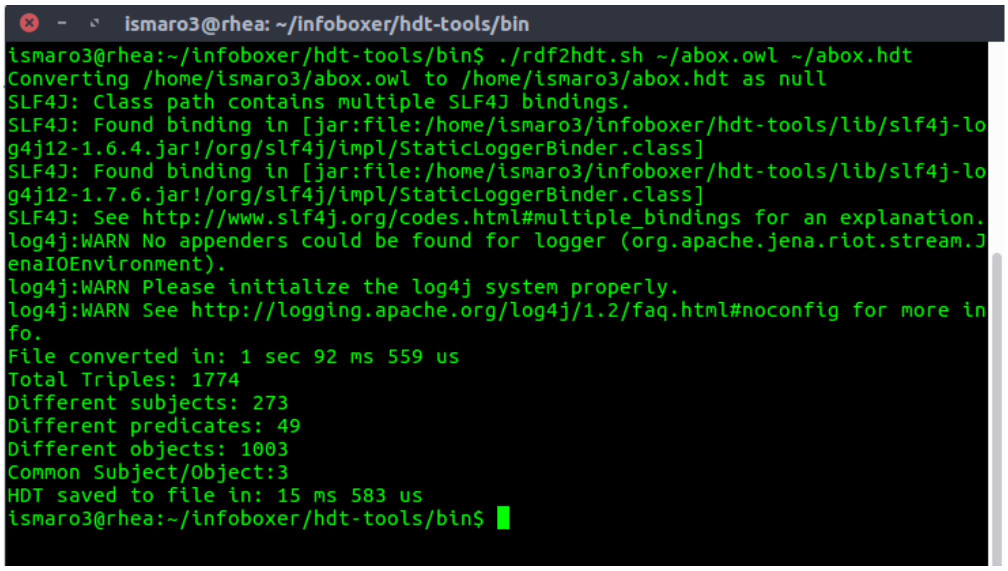
2. Execute the *hdt2rdf.sh* or *hdt2rdf.bat* file, passing as parameters the source and destination files, as shown in Listing B.2.

3. Rename the output HDT file to *kb.hdt* and move it into the *hdt-fuseki-launcher* directory.

```
./rdf2hdt.sh source.rdf destination.hdt
```

Listing B.2: Command for converting a Knowledge Base into HDT format

A screenshot of the console output of a conversion can be seen on Figure B.2.



```
ismaro3@rhea:~/infoboxer/hdt-tools/bin$ ./rdf2hdt.sh ~/abox.owl ~/abox.hdt
Converting /home/ismaro3/abox.owl to /home/ismaro3/abox.hdt as null
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/ismaro3/infoboxer/hdt-tools/lib/slf4j-lo
g4j12-1.6.4.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/ismaro3/infoboxer/hdt-tools/lib/slf4j-lo
g4j12-1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
log4j:WARN No appenders could be found for logger (org.apache.jena.riot.stream.J
enaIOEnvironment).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
File converted in: 1 sec 92 ms 559 us
Total Triples: 1774
Different subjects: 273
Different predicates: 49
Different objects: 1003
Common Subject/Object:3
HDT saved to file in: 15 ms 583 us
ismaro3@rhea:~/infoboxer/hdt-tools/bin$
```

Figure B.2: Screenshot: RDF to HDT conversion process output.

Web Server Configuration

Before starting the server, some configuration files have to be modified to adapt the prototype to the hosts where it will run. In this section, the configuration files for DBpedia are used as a reference. However, in Appendix B.3, files for other three KBs are provided.

The configuration file for the web server is stored in the *infoboxer* folder, specifically in the *src/main/resources/application.properties* file. The changes that have to be made to the file are now detailed:

1. The *"server.port"* option has to be changed to the desired port where the web application will be served. By default, it is 8080. Ports like 80 require administrative privileges.

2. The *"translator.apiKEY"* value has to be introduced if "translator.enabled" equals true. A Key can be obtained on the Yandex Translator website[4].

3. The *"sparql.url"* option has to be changed to match the address of the server that hosts the SPARQL endpoint and the port where it is running.

[4]https://tech.yandex.com/translate/

4. The *“ontology.location”* value has to be modified to the path where the ontology file is located (in the case of the downloaded DBpedia KB, where the *owl* file is placed).
5. At the bottom of the file, the options referring to host, port, username, and password of the stats and suggestions database have to be changed to the match host where the MySQL database is hosted, and the created user.

The configuration file for the Frontend is stored in the *infoboxer* folder, specifically in the *src/main/resources/static/data/config.json* file. The only change to be performed is to modify the *“base”* property, so it is equal to the public address of the machine that hosts the web server with the port where it is running.

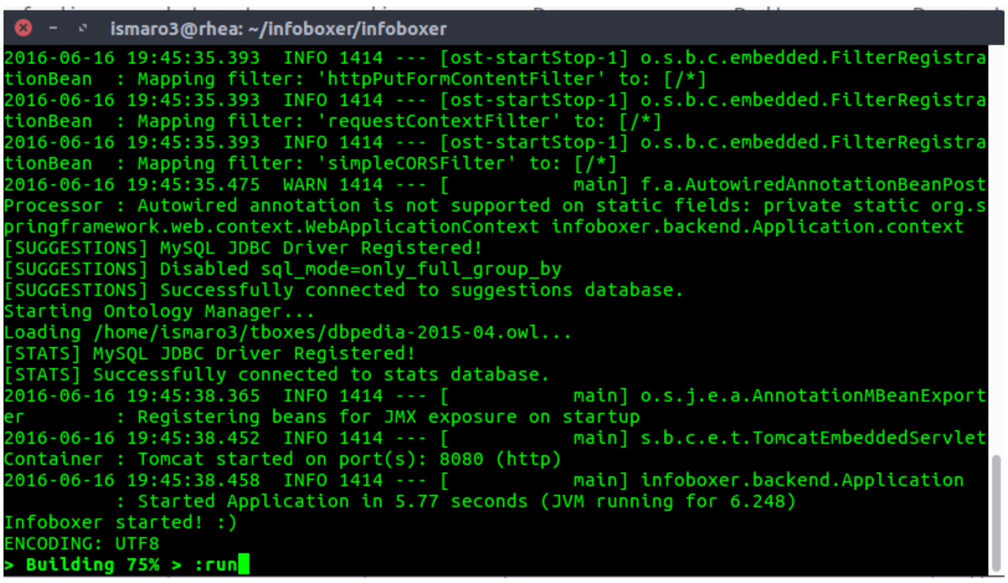
In the Appendix B.2, all the options of the configuration files are detailed.

Web Server Execution

Finally, to start the web server, the steps are the following:

1. Before the first execution, the dependencies of the Frontend have to be downloaded. To do so, if using Linux, the file *install_frontend_auto_linux.sh* located in the *infoboxer* folder has to be executed. If using Windows, the command *npm install* has to be executed in a terminal whose working directory is *infoboxer/src/main/resources/static.*
2. The MySQL database and SPARQL endpoint have to be running. If don't, they have to be started as detailed previously.
3. The file *start_linux.sh* or *start_windows.bat* has to be executed to launch the server. The first time it will take some minutes as it has to download the needed dependencies. A screenshot of the console output of the web server process is shown on Figure B.3

So, from now on, only the two last steps of the previous list have to be followed to launch the server. The web application can be accessed from a web browser on the address and port configured for the web server.



```
2016-06-16 19:45:35.393  INFO 1414 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistra
tionBean  : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2016-06-16 19:45:35.393  INFO 1414 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistra
tionBean  : Mapping filter: 'requestContextFilter' to: [/*]
2016-06-16 19:45:35.393  INFO 1414 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistra
tionBean  : Mapping filter: 'simpleCORSFilter' to: [/*]
2016-06-16 19:45:35.475  WARN 1414 --- [           main] f.a.AutowiredAnnotationBeanPost
Processor : Autowired annotation is not supported on static fields: private static org.s
pringframework.web.context.WebApplicationContext infoboxer.backend.Application.context
[SUGGESTIONS] MySQL JDBC Driver Registered!
[SUGGESTIONS] Disabled sql_mode=only_full_group_by
[SUGGESTIONS] Successfully connected to suggestions database.
Starting Ontology Manager...
Loading /home/ismaro3/tboxes/dbpedia-2015-04.owl...
[STATS] MySQL JDBC Driver Registered!
[STATS] Successfully connected to stats database.
2016-06-16 19:45:38.365  INFO 1414 --- [           main] o.s.j.e.a.AnnotationMBeanExport
er         : Registering beans for JMX exposure on startup
2016-06-16 19:45:38.452  INFO 1414 --- [           main] s.b.c.e.t.TomcatEmbeddedServlet
Container : Tomcat started on port(s): 8080 (http)
2016-06-16 19:45:38.458  INFO 1414 --- [           main] infoboxer.backend.Application
          : Started Application in 5.77 seconds (JVM running for 6.248)
Infoboxer started! :)
ENCODING: UTF8
> Building 75% > :run
```

Figure B.3: Screenshot: web server process output.

B.2 Configuration Files

In this section, the three configuration files used in the prototype are detailed. They are used to configure the used KB, functionality of the web server and Frontend, and what categories can be selected by the user.

Web Server Configuration File

The web server configuration file, available in the */infoboxer/src/main/resources/static/application.properties* directory of the project, contains a set of key-value pairs that are now described. All the keys inside a group have the name of the group as a prefix separated by a dot. Note that neither the keys nor the values are surrounded by quotes.

- **messages.enabled**: if set to *true*, debug messages are shown in the process console.
- **translator**: it defines three options related to the automatic translation of property and ranges:
  - **enabled**: whether the property and range translation from English to other languages is enabled.
  - **apiURL**: the URL of the Yandex translator API being used, for example *https://translate.yandex.net/api/v1.5/tr.json/translate*.
  - **apiKEY**: the Yandex Translator Developer Key being used.
- **sparql.url**: the URL where the SPARQL endpoint is located.
- **ontology**: it groups three settings about the ontology associated to the KB and its classes:
  - **location**: the local path where the ontology file is located.
  - **allowedClasses**: a comma-separated list of regular expressions, that defines which classes are taken into account. Classes not defined here are ignored by the system.
  - **classTransformations**: a list of comma-separated groups of parenthesis, each containing a pair of properties separated by comma. The first property in every group is transformed to the second property in the same group. Used when a property in the KB has to be transformed.
- **label**: it groups settings about how property, class, and instance labels are derived from URIs:
  - **propertyDelimiter**: the last character of a property or instance URI after which the property or instance name is defined. For example, for the property <http://dbpedia.org/ontology/birthPlace>, this character is '/' (without quotes).

  - **typeDelimeter**: the last character of a class URI after which the class name is defined. For example, for the property `<http://a.org/SoccerPlayer>`, this character is '/'.
  - **uriToLabelMode**: it defines how the property or class name is represented in an URI. Can be either 'camelCase' (if the name is defined like 'SoccerPlayer') or 'underscore' (if it is defined like 'soccer_player', words separated by underscores).

- **instanceLabel**: it holds settings about the extraction of the label of instances from the KB:

  - **fromKB:** if *true*, the label for suggested instances is retrieved from the KB, using for each instance the property defined in the next option. If 'false', the label is derived from the URI using the 'propertyDelimiter'.
  - **property:** the property, surrounded with '<' and '>', whose value is used as a label for every suggested instance, if 'fromKB' is set to 'true'. An example would be `<http://www.w3.org/2004/02/skos/core#prefLabel>`. Leave empty if no property is used.

- **unknownType**: it contains expressions to control what type is assigned to a value or instance that has no type defined in the KB:

  - **resource.value**: regular expression to define how are resources or instances identified in the KB (for example, *".*dbpedia.org/resource.*"* for DBpedia).
  - **resource.type**: the URI of the type, surrounded with '<' and '>', that is assigned to a value identified as a resource, when no type is found in the KB.
  - **langString.type**: the URI of the type, surrounded with '<' and '>', that is assigned to a value whose format is the format of a langString (string with language information associated).
  - **string.type**: the URI of the type, surrounded with '<' and '>', that is assigned to a value whose format is the format of a string.
  - **numeric.type**:the URI of the type, surrounded with '<' and '>', that is assigned to a value whose format is the format of a Number.
  - **else.type**: the URI of the type, surrounded with '<' and '>', that is assigned to a value not identified as any of the previous types (resource, lang-string, string nor numeric).

- **stats.db**: it contains settings about the database of users interactions.

  - **host**: the IP address of the MySQL host.
  - **port**: the port of the MySQL server.
  - **username**: The MySQL user created for the prototype.
  - **password**: the password of the user.

  - **database**: name of the database of stats. Recommended to be "infoboxer".

- **suggestions.db**: it contains settings about the database of suggested values.
  - **host**: the IP address of the MySQL host.
  - **port**: the port of the MySQL server.
  - **username**: the MySQL user created for the prototype.
  - **password**: the password of the user.
  - **database**: name of the database of stats. Recommended to be "suggestions".

Frontend Configuration File

The Frontend configuration file, available in the */infoboxer/src/main/resources/static/data/config.json* file, contains a JSON object whose content is now described. All the keys and string values are surrounded by quotes. Booleans, numbers, and arrays are not.

- **endpoint**: it is a JSON object that groups settings about where the web server is located and paths of different methods:
  - **base**: the public HTTP address with port where the web server is running. For example, http://sid06.cps.unizar.es:8080.
  - **data**: the relative path of the web server from which all the operations are offered. It must be "/infoboxer".
  - **suggestions**: the relative path of the web server from which the suggestions operations are offered. It must be "/suggestions".
  - **fromWikipedia**: the relative path of the web server where the "Data for instance" operation is served.
  - **generateInfobox**: the relative path of the web server where the operation that generates a Wikipedia Infobox is served.
  - **stats**: a JSON object that contains properties defining the location of each one of the methods for registering user interaction and stats. The properties are "newSession", "newAction", "closeSession", "saveInfobox", "saveRdf", "saveSurvey" and "wikimediaTime", and their values must be the name of the property prefixed by "/stats/".

- **stats**: a JSON object that contains a unique property, "activated", whose value is a boolean. If *true*, the Frontend makes a call to the web server on every user interaction, so it is registered.

- **thumbnail**: a JSON object that contains settings of the shown image thumbnail in the infobox simulator:

  - **activated**: boolean indicated if the thumbnail is shown or not.
  - **property**: property not surrounded with "<" nor ">" that identifies an image thumbnail in the current KB.
  - **description**: property not surrounded with "<" nor ">" that identifies the description of an entity in the current KB.

- **infoboxCode**: JSON object with settings about the generated infobox code.

  - **format**: string that can be "wikipedia" or "mediawiki". If "wikipedia", the generated code will have the format required in Wikipedia. If "mediawiki", the format will be the required by the MediaWiki wikis.
  - **serverside**: boolean indicating if the Infobox generated code will be generated in the client (*false*), or will be generated on the server (*true*). If generated on the server, Wikipedia Infobox templates are applied, so it can be inserted in Wikipedia pages. Only valid if *"wikipedia"* is selected in the *"format"* option.

- **categoriesRestrictions**: JSON object that contains settings about the categories that can be selected:

  - **restrictByNumber**: boolean indicating if a limit on the number of selected categories is set.
  - **maxNumber**: if the previous property is set to *true*, it indicates the maximum number of categories that can be selected.
  - **restrictByWhitelist**: boolean indicating if, when two or more categories are selected, only the defined combinations in the next setting can be selected.
  - **whitelist**: array containing arrays of posible classes combinations. Only valid if the previous setting is set to *true*.

- **resourcePrefix**: a JSON object that contains definitions about how resources are represented in the KB. It contains a property for every language to be supported ("es", "en", "fr",...) , and one "default" property used when the interface language is not listed there. The value of every property is the URI prefix used for representing resources. For example, for DBpedia, it would be "http://dbpedia.org/resource/".

Frontend Categories File

This file, located in */infoboxer/src/main/resources/static/data/categories.json*, contains the categories shown in the drop-down list of the Frontend. Its format is a recursive JSON array of "Category" objects. Each of these objects contain the following properties:

- **__id**: the URI of the class or category, surrounded by "<" and ">". For example, <http://dbpedia.org/ontology/Artist>.

- **name**: the displayed name for that category. For example, “Artist”.
- **children**: an array of “Category” objects like the one being described. They will be shown as children of this category.

  More than one category can be present with the same “_id” attribute, but the “name” attribute can not be repeated. A little example is shown on Listing B.3.

```
[
  {
    ``_id'': ``<http://dbpedia.org/ontology/Person>'',
    ``name'': ``Person'',
    ``children'': [
                    {
                      ``_id'': ``<http://dbpedia.org/ontology/Artist>'',
                      ``name'': ``Artist'',
                      ``children'':   []
                    }
    ]
  }
]
```

File B.3: Example of Frontend categories file.

B.3 Loading different KBs

In this section, first the requirements that a KB must meet to be loaded into the system are enumerated. Then, the needed files, configurations and steps to load four different KB on the system are detailed. Also, for each KB, some statistics about instances, classes, and properties are given.

B.3.1 Knowledge Base Considerations

In order to use a certain KB in the developed system, it must satisfy some requirements. Some or them are mandatory so the system can properly work. Others are optional, and its compliance improve how data is presented to user.

Mandatory Features

1. **Required files**: the ontology schema and the KB must be separated in two different files.
2. **Files format**: the format of the schema and KB should be either RDF/XML or N-triples.

Optional Features

1. **Property and range names**: in order to show the name (label) of properties and ranges correctly, its definition in the ontology schema must have “*rdfs:label*” defined, so its value is used. In the case it is not present, the shown value will be automatically derived from its URI. For example, for

the URI "http://dbpedia.org/ontology/birthPlace" the text "Birth place" will be shown. The way the name is derived from the URI can be set on the Backend configuration files.

2. **Property descriptions**: to show a description of the obtained properties when the user hovers the mouse over their name, their definitions in the ontology schema should have "*rdfs:comment*" defined. If not present, no description will be shown.

3. **Instance names**: Suggested instances names or labels are usually automatically derived from its resource URI. Besides, a property's value in the KB can be used as the label for an instance. For example, each instance of a KB could have a *'name'* or *'label'* property whose value represents the instance.

B.3.2 DBpedia

DBpedia is the most famous KB of the Linked Data project. It is a general purpose KB whose data comes from Wikipedia, but is also connected with well-known KBs like GeoNames, Gutenberg Project or FOAF. It has several versions, one for each Wikipedia language. Currently, the English version contains almost 6,000,000 instances, with a mean of 7 properties per instance. The steps to use this KB (version 2015-04) are now described.

Download of the Required Files

1. First, in the downloads page of DBpedia 2015-04[5] the ontology file[6] must be downloaded.

2. Then, in the same page, the "Mapping-based types" file[7], the "Mapping-based types (transitive)" file[8] and the "Mapping based properties" file[9] have to be downloaded.

3. Next, the files downloaded in the previous point have to be decompressed and joined into one only file to form the KB. The *cat* UNIX tool can be used for that.

4. Finally, the composed KB has to be converted into HDT and loaded into the endpoint, and the ontology location has to be written on the Backend configuration file.

If a newer version of DBpedia or a version in a different language is wanted, the same steps have to be performed but downloading the files appropriate for the

[5]`http://wiki.dbpedia.org/Downloads2015-04`

[6]`http://downloads.dbpedia.org/2015-04/dbpedia_2015-04.owl.bz2`

[7]`http://downloads.dbpedia.org/2015-04/core-i18n/en/instance-types_en.nt.bz2`

[8]`http://downloads.dbpedia.org/2015-04/core-i18n/en/instance-types-transitive_en.nt.bz2`

[9]`http://downloads.dbpedia.org/2015-04/core-i18n/en/mappingbased-properties_en.nt.bz2`

desired version. A listing of the available versions can be found at the DBpedia Dataset page[10].

Backend Configuration

An example Backend configuration file for using both DBpedia 2015-04 English and Spanish version is shown on Listing B.4. Only fields related to the KB configuration are shown.

```
#Infoboxer Core Backend configuration file for DBpedia
##Ontology
ontology.location =/PATH/TO/TBOX.owl
#Allowed classes for all: obtaining superclasses, filtering...
ontology.allowedClasses = http://.*dbpedia\.org/.*,http://www.w3.org/.*
    XMLSchema.*,http://www.w3.org/.*langString.*,http://www.w3.org/.*owl#
    Thing.*,http://xmlns.com/foaf/.*
ontology.classTransformations = (http://xmlns.com/foaf/0.1/Person,http://
    dbpedia.org/ontology/Person)

label.propertyDelimiter = /
label.typeDelimiter = /
label.uriToLabelMode = camelCase

instanceLabel.fromKB = false
instanceLabel.property =


unknownType.resource.value=.*dbpedia.org/resource.*
unknownType.resource.type=<http://www.w3.org/2002/07/owl#Thing>
unknownType.langString.type=<http://www.w3.org/1999/02/22-rdf-syntax-ns#
    langString>
unknownType.string.type=``XMLSchema#String''
unknownType.numeric.type=<http://www.w3.org/2001/XMLSchema#integer>
unknownType.else.type=<http://www.w3.org/2002/07/owl#Thing>
```

File B.4: DBpedia Backend configuration file.

Frontend Configuration

An example Frontend configuration file for using DBpedia is shown on Listing B.5. Only fields related to the KB configuration are shown. Also, Listing B.6 shows an example category file for DBpedia.

```
``resourcePrefix'':{
  ``en'': ``http://dbpedia.org/resource/'',
  ``es'': ``http://es.dbpedia.org/resource/'',
  ``default'':  ``http://dbpedia.org/resource/''
}
```

File B.5: DBpedia Frontend configuration file.

```
[
  {
    ``_id'': ``<http://dbpedia.org/ontology/Person>'',
    ``name'': ``Person'',
    ``children'' : [
      {
        ``_id'': ``<http://dbpedia.org/ontology/Athlete>'',
        ``name'': ``Athlete'',
        ``children'': [
          {
```

[10]http://wiki.dbpedia.org/datasets

```
            ``_id'': ``<http://dbpedia.org/ontology/BasketballPlayer>'',
            ``name'': ``Basketball Player''
          },
          {
            ``_id'': ``<http://dbpedia.org/ontology/SoccerPlayer>'',
            ``name'': ``SoccerPlayer''
          }
        ]
      },

      {
        ``_id'': ``<http://dbpedia.org/ontology/Artist>'',
        ``name'': ``Artist'',
        ``children'': [
          {
            ``_id'': ``<http://dbpedia.org/ontology/Actor>'',
            ``name'': ``Actor''
          }
        ]
      },
      {
        ``_id'': ``<http://dbpedia.org/ontology/Governor>'',
        ``name'': ``Governor''
      },
      {
        ``_id'': ``<http://dbpedia.org/ontology/Bodybuilder>'',
        ``name'': ``Bodybuilder''
      }
    ]
  }
]
```

File B.6: DBpedia Frontend categories file.

Using the KB

The DBpedia KB is working successfully, as it has been the main KB used during the prototype development (see screenshot in Figure B.4). It is the most complete tested KB; property and range labels, and property comments can be extracted from the ontology, as the “rdfs:label” and “rdfs:comment” attributes are present. Besides, as the used URIs for the instances are very descriptive, the labels used for the suggested values can be derived from them.

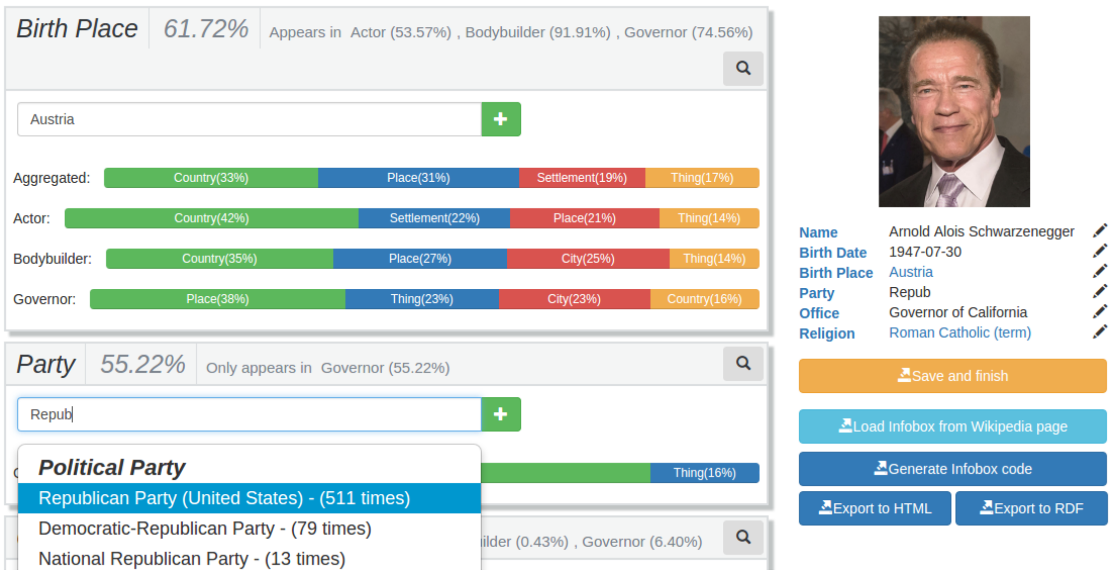
There are more than 700 classes defined on the DBpedia ontology. Some of them are shown in Table B.3 along the number of instances of each one on the English version of DBpedia 2015-04. They are indented according to its level in the hierarchy (for example, “Athlete” is a child class of “Person”). Note that, as “Person” is superclass of the rest of shown classes, it has a greater amount of instances. It also happens for “Athlete” being superclass of “SoccerPlayer” and “BasketballPlayer”, and for “Artist” being superclass of “Actor”. For example, for the class “SoccerPlayer”, a total of 414 properties were retrieved, being 40 statistical and 374 semantic. Some statistical properties are shown in Table B.5, with the number and percentage of uses and the suggested ranges with their use frequency. Regarding the semantic properties, some of them are “Home Town”, “Former team” or “Spouse”.

Class	**Number of instances**
Person	710143
··Athlete	30071
····SoccerPlayer	102618
····BasketballPlayer	9411
··Artist	30451
····Actor	5102
··Governor	2689
··Bodybuilder	235

Table B.3: Classes and number of instances for DBpedia.

Property	Name	Team	Birthplace
Uses of property	82811 (80.70%)	82661 (80.55%)	74879 (72.97%)
Shown ranges	String (100%)	Soccer club (76%) Thing (22%) Sports Team (1%) Event (1%)	Country (42%) Settlement (24%) Place (18%) Thing (16%)

Table B.5: Properties for "SoccerPlayer" class, with their ranges and uses.


Birth Place 61.72% Appears in Actor (53.57%) , Bodybuilder (91.91%) , Governor (74.56%)

Austria

Aggregated: Country(33%) Place(31%) Settlement(19%) Thing(17%)

Actor: Country(42%) Settlement(22%) Place(21%) Thing(14%)

Bodybuilder: Country(35%) Place(27%) City(25%) Thing(14%)

Governor: Place(38%) Thing(23%) City(23%) Country(16%)

Party 55.22% Only appears in Governor (55.22%)

Repub

Political Party
Republican Party (United States) - (511 times)
Democratic-Republican Party - (79 times)
National Republican Party - (13 times)

Thing(16%)

ilder (0.43%) , Governor (6.40%)

Name	Arnold Alois Schwarzenegger
Birth Date	1947-07-30
Birth Place	Austria
Party	Repub
Office	Governor of California
Religion	Roman Catholic (term)

Save and finish
Load Infobox from Wikipedia page
Generate Infobox code
Export to HTML
Export to RDF

Figure B.4: Screenshot: prototype using DBpedia KB.

B.3.3 GeoSpecies

GeoSpecies is a KB that contains information about biological Kingdoms, Orders, Families, Species, etc. Currently, it contains more than 20,000 instances, with a mean of 21 properties per instance. The steps to use this KB are now described.

Download of the Required Files

The GeoSpecies dataset can be found at its Datahub Site[11]. There, the ontology [12] and the KB file[13] are found. The KB has to be converted to HDT format using the proper tool specified in Appendix B.1.

Backend Configuration

An example Backend configuration file for using GeoSpecies is shown on Listing B.7. Only fields related to the KB configuration are shown.

```
#Infoboxer Core Backend configuration file for GeoSpecies
ontology.location =/PATH/TO/TBOX.owl
ontology.allowedClasses = http://rdf.geospecies.org.*,http://purl.org/.*,
    http://www.w3.org/.*XMLSchema.*,http://www.w3.org/.*langString.*,
    http://www.w3.org/.*owl#Thing.*,http://xmlns.com/foaf/.*
ontology.classTransformations =
label.propertyDelimiter = #
label.typeDelimiter = #
label.uriToLabelMode = camelCase

instanceLabel.fromKB = true
instanceLabel.property = <http://www.w3.org/2004/02/skos/core#prefLabel>

unknownType.resource.value=.*lod.geospecies.org.*
unknownType.resource.type=<http://www.w3.org/2002/07/owl#Thing>
unknownType.langString.type=<http://www.w3.org/1999/02/22-rdf-syntax-ns#
    langString>
unknownType.string.type=<http://www.w3.org/2001/XMLSchema#String>
unknownType.numeric.type=<http://www.w3.org/2001/XMLSchema#integer>
unknownType.else.type=<http://www.w3.org/2002/07/owl#Thing>
```

File B.7: Geospecies Backend configuration file.

Frontend Configuration

An example Frontend configuration file for using GeoSpecies is shown on Listing B.8. Only fields related to the KB configuration are shown. Also, Listing B.9 shows an example category file for Geospecies.

```
``resourcePrefix'':{
  ``en'': ``http://lod.geospecies.org/resource/'',
  ``default'':  ``http://lod.geospecies.org/resource/''
}
```

File B.8: Geospecies Frontend configuration file.

```
[
  {
    ``_id'': ``<http://rdf.geospecies.org/ont/geospecies#KingdomConcept>'',
    ``name'': ``Kingdom'',
    ``children'': [
    ]
  },
  {
    ``_id'': ``<http://rdf.geospecies.org/ont/geospecies#PhylumConcept>'',
    ``name'': ``Phylum'',
```

[11]https://datahub.io/es/dataset/geospecies

[12]https://datahub.io/es/dataset/geospecies/resource/d1c7cdf1-cd87-4764-8ed9-01e6baa90c5b

[13]http://lod.geospecies.org/geospecies.rdf.gz

```
    ``children'': [
    ]
  },
  {
    ``_id'': ``<http://rdf.geospecies.org/ont/geospecies#ClassConcept>'',
    ``name'': ``Class'',
    ``children'': [
    ]
  },
  {
    ``_id'': ``<http://rdf.geospecies.org/ont/geospecies#OrderConcept>'',
    ``name'': ``Order'',
    ``children'': [
    ]
  },
  {
    ``_id'': ``<http://rdf.geospecies.org/ont/geospecies#FamilyConcept>'',
    ``name'': ``Family'',
    ``children'': [
    ]
  },
  {
    ``_id'': ``<http://rdf.geospecies.org/ont/geospecies#SpeciesConcept>'',
    ``name'': ``SpeciesConcept'',
    ``children'': [
    ]
  }
]
```

File B.9: Geospecies Frontend categories file.

Using the KB

The GeoSpecies KB is working successfully in the system, as seen in Figure B.5. However, property and range labels, and property comments can not be extracted from the ontology by using either "rdfs:label" or "rdfs:comment", so they are derived from the URI. In contrast, the labels used for the suggested value are obtained from the KB using the "<http://www.w3.org/2004/02/skos/core#prefLabel>" property.

There are six main classes in the KB, which are shown in Table B.6 along the number of instances of each one. For the class with more instances ("SpeciesConcept"), a total of 90 properties were retrieved, being 73 statistical and 17 semantic. The most popular statistical properties are shown in Table B.8, with the number and percentage of uses and the suggested ranges with their use frequency. Regarding the semantic properties, some of them are "Has Vernacular Name", "Has Basionym Name" or "Has Nomenclatural Code String".

Class	Number of instances
KingdomConcept	8
PhylumConcept	78
ClassConcept	50
OrderConcept	217
FamilyConcept	1650
SpeciesConcept	18878

Table B.6: Classes and number of instances for Geospecies.

Property	Close match	Has Canonical Name	In Order
Uses of property	18878 (100%)	18878 (100%)	18878 (100%)
Shown ranges	Thing (58%) DBpedia Resource (22%) Bio2RDFtaxo (21%)	String (100%)	Order Concept (100%)

Table B.8: Properties for ”Species Concept” class, with their ranges and uses.

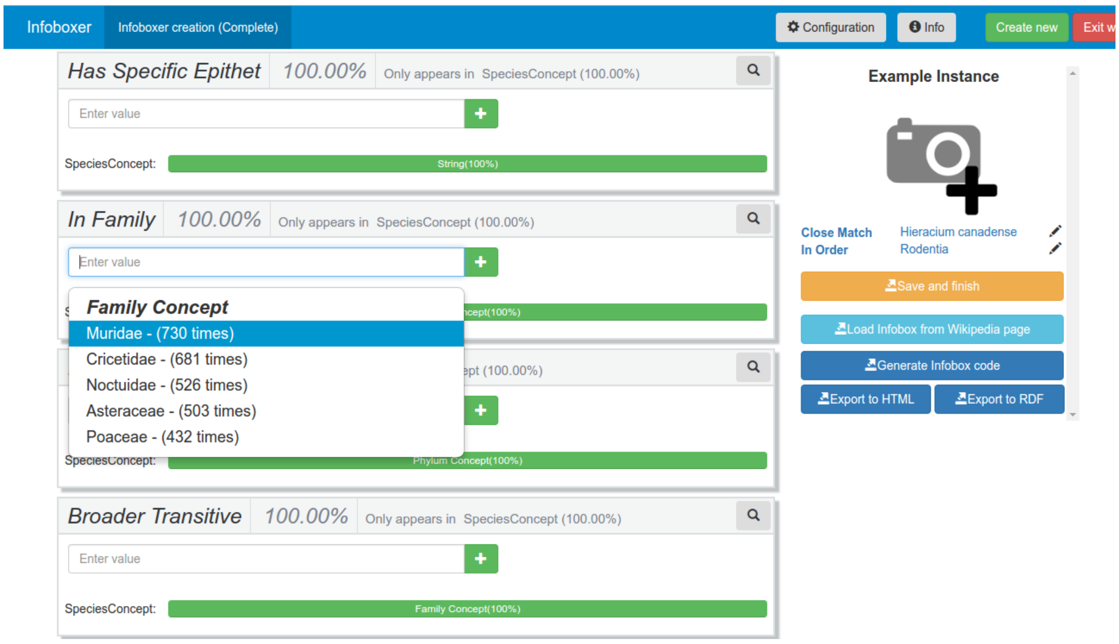


Figure B.5: Screenshot: prototype using GeoSpecies KB.

B.3.4 IIMB Test KB

IIMB[14] (ISLab Instance Matching Benchmark) test KB is a KB that provides data about actors, sport persons, and business firms taken from the OKKAM project, used for testing purposes. It contains about 200 properties with a mean of 8 properties per instance. The steps to use this KB are now described:

Download of the Required files

The dataset can be downloaded from the IIMB page [15]. This file has to be uncompressed, and the original ontology (“tbox.owl”) and KB (“abox.owl”) files have to be used (located in the root of the directory). The KB file has to be converted into HDT format, and the ontology file location has to be indicated in the web

[14]`http://islab.di.unimi.it/iimb/`
[15]`http://islab.di.unimi.it/iimb/iimb.tgz`

server configuration files.

Backend Configuration

An example Backend configuration file for using GeoSpecies is shown on Listing B.10. Only fields related to the KB configuration are shown.

```
#Infoboxer Core Backend configuration file for IIMB test KB
ontology.location =/PATH/TO/TBOX.owl
ontology.allowedClasses = http://islab.dico.unimi.it/iimb/.*,
http://www.w3.org/.*XMLSchema.*,
http://www.w3.org/.*langString.*,
http://www.w3.org/.*owl#Thing.*,http://xmlns.com/foaf/.*
ontology.classTransformations =


label.propertyDelimiter = #
label.typeDelimiter = #
label.uriToLabelMode = underscore

instanceLabel.fromKB = false
instanceLabel.property =

unknownType.resource.value=http://islab.dico.unimi.it/imb/tbox.owl*
unknownType.resource.type=<http://www.w3.org/2002/07/owl#Thing>
unknownType.langString.type=<http://www.w3.org/1999/02/22-rdf-syntax-ns#
    langString>
unknownType.string.type=‘‘XMLSchema#String’’
unknownType.numeric.type=<http://www.w3.org/2001/XMLSchema#integer>
unknownType.else.type=<http://www.w3.org/2002/07/owl#Thing>
```

File B.10: IIMB Backend configuration file.

Frontend Configuration

An example Frontend configuration file for using GeoSpecies is shown on Listing B.11. Only fields related to the KB configuration are shown. Also, Listing B.12 shows an example category file for IIMB.

```
‘‘resourcePrefix’’:{
  ‘‘en’’: ‘‘http://islab.dico.unimi.it/iimb/abox.owl\#’’,
  ‘‘default’’:  ‘‘http://islab.dico.unimi.it/iimb/abox.owl\#’’
}
```

File B.11: IIMB Frontend configuration file.

```
[
  {
    ‘‘_id’’: ‘‘<http://islab.dico.unimi.it/iimb/tbox.owl\#actor>’’,
    ‘‘name’’: ‘‘IIMB Actor’’,
    ‘‘children’’: [
    ]
  },
  {
    ‘‘_id’’: ‘‘<http://islab.dico.unimi.it/iimb/tbox.owl\#sportsperson>’’,
    ‘‘name’’: ‘‘IIMB Sports Person’’,
    ‘‘children’’: [
    ]
  },
  {
    ‘‘_id’’: ‘‘<http://islab.dico.unimi.it/iimb/tbox.owl\#business_firm>’’,
    ‘‘name’’: ‘‘IIMB Business Firm’’,
    ‘‘children’’: [
    ]
  }
]
```

File B.12: IIMB Frontend categories file.

Using the KB

The IIMB test KB is working successfully in the system, as seen in Figure B.6. However, property and range labels, and property comments can not be extracted from the ontology using *rdfs:label* nor *rdfs:comment*, so they are derived from the URI. The labels used for the suggested values are also derived from the URI.

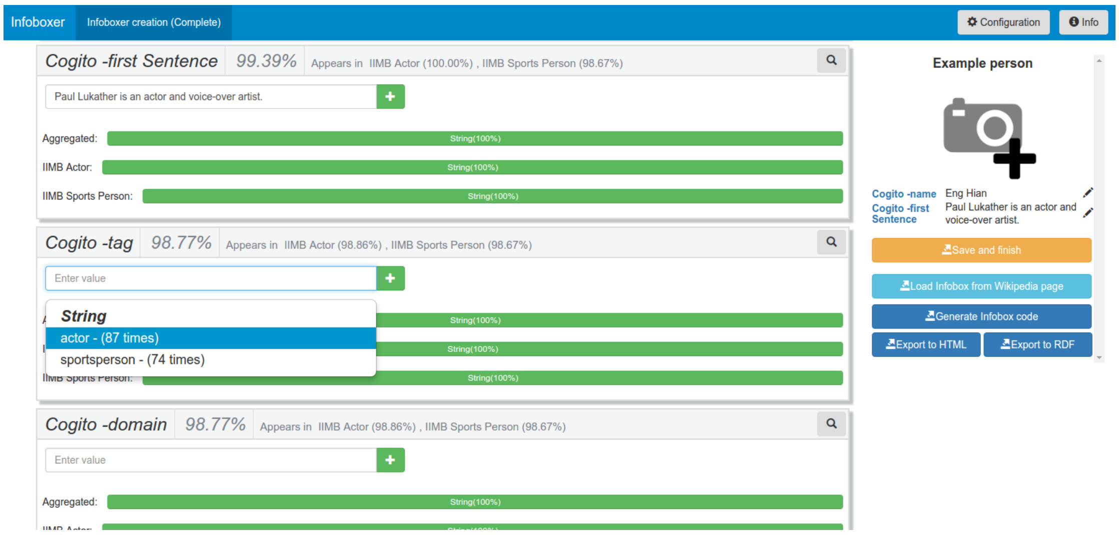
There are three main classes in the KB, which are shown in Table B.9 along the number of instances of each one. For the class with more instances ("Actor"), the most popular properties are shown in Table B.11, with the number and percentage of uses and the suggested ranges with their use frequency. As it can be seen, only the String range is present; in the whole KB, all the values are String. Also, no semantic properties were retrieved.

Class	Number of instances
Business-Firm	59
Actor	88
SportsPerson	75

Table B.9: Classes and number of instances for IIMB test KB.

Property	first Sentence	tag	domain
Uses of property	88 (100%)	87 (98.86%)	87 (98.86%)
Shown ranges	String (100%)	String (100%)	String (100%)

Table B.11: Properties for "Actor" class, with their ranges and uses.



Cogito -first Sentence 99.39% Appears in IIMB Actor (100.00%) , IIMB Sports Person (98.67%)

Paul Lukather is an actor and voice-over artist.

Aggregated: String(100%)
IIMB Actor: String(100%)
IIMB Sports Person: String(100%)

Cogito -tag 98.77% Appears in IIMB Actor (98.86%) , IIMB Sports Person (98.67%)

Enter value

String
actor - (87 times)
sportsperson - (74 times)

String(100%)
String(100%)
IIMB Sports Person: String(100%)

Cogito -domain 98.77% Appears in IIMB Actor (98.86%) , IIMB Sports Person (98.67%)

Enter value

Aggregated: String(100%)

Example person

Cogito -name	Eng Hian
Cogito -first Sentence	Paul Lukather is an actor and voice-over artist.

Figure B.6: Screenshot: prototype using IIMB KB.

B.3.5 SwetoDBLP

SwetoDBLP[16] is "a large-size ontology (spin-off of SWETO ontology) focused on bibliography data of Computer Science publications where the main data source is DBLP". Currently, it contains more than 920,000 instances, with a mean of 14 properties per instance. The steps to use this KB are now described.

Download of the Required Files

At the SwetoDBLP main page the ontology[17] (schema) and KB[18] (instances) files can be found. The only needed processing is to convert the KB file to HDT. The ontology file location has to be indicated in the web server configuration file.

Backend Configuration

An example Backend configuration file for using SwetoDBLP is shown on Listing B.13. Only fields related to the KB configuration are shown.

```
#Infoboxer Core Backend configuration file for SwetoDBLP
ontology.location =/PATH/TO/TBOX.RDF
ontology.allowedClasses = http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq,
http://lsdis.cs.uga.edu.*,http://www.w3.org/.*XMLSchema.*,
http://www.w3.org/.*langString.*,http://www.w3.org/.*owl#Thing.*,
http://xmlns.com/foaf/.*
ontology.classTransformations =


label.propertyDelimiter = #
label.typeDelimiter = #
label.uriToLabelMode = underscore

instanceLabel.fromKB = true
instanceLabel.property =<http://www.w3.org/2000/01/rdf-schema#label>


unknownType.resource.value=http://lsdis.cs.uga.edu/projects/semdis/opus#
unknownType.resource.type=<http://www.w3.org/2002/07/owl#Thing>
unknownType.langString.type=<http://www.w3.org/1999/02/22-rdf-syntax-ns#
    langString>
unknownType.string.type=<http://www.w3.org/2001/XMLSchema#string>
unknownType.numeric.type=<http://www.w3.org/2001/XMLSchema#integer>
unknownType.else.type=<http://www.w3.org/2002/07/owl#Thing>
```

File B.13: SwetoDBLP Backend configuration file.

Frontend Configuration

An example Frontend configuration file for using SwetoDBLP is shown on Listing B.14. Only fields related to the KB configuration are shown. Also, Listing B.15 shows an example category file for SwetoDBLP.

[16]http://lsdis.cs.uga.edu/projects/semdis/swetodblp/

[17]http://lsdis.cs.uga.edu/projects/semdis/swetodblp/august2007/opus_august2007.rdf

[18]http://lsdis.cs.uga.edu/projects/semdis/swetodblp/august2007/swetodblp_august2007.rdf.gz

```
``resourcePrefix'':{
  ``en'': ``http://dblp.uni-trier.de/rec/'',
  ``default'':  ``http://dblp.uni-trier.de/rec/''
}
```

File B.14: SwetoDBLP Frontend configuration file.

```
[
  {
    ``_id'': ``<http://lsdis.cs.uga.edu/projects/semdis/opus#Book>'',
    ``name'': ``SwetoDBLP Book''
  },
  {
    ``_id'': ``<http://lsdis.cs.uga.edu/projects/semdis/opus#Proceedings>'',
    ``name'': ``SwetoDBLP Proceedings''
  },
  {
    ``_id'': ``<http://lsdis.cs.uga.edu/projects/semdis/opus#
        Article_in_Proceedings>'',
    ``name'': ``SwetoDBLP Article in Proceedings''
  },
  {
    ``_id'': ``<http://lsdis.cs.uga.edu/projects/semdis/opus#Webpage>'',
    ``name'': ``SwetoDBLP Webpage''
  },
  {
    ``_id'': ``<http://lsdis.cs.uga.edu/projects/semdis/opus#Article>'',
    ``name'': ``SwetoDBLP  Article''
  },
  {
    ``_id'': ``<http://lsdis.cs.uga.edu/projects/semdis/opus#Masters_Thesis>'',
    ``name'': ``SwetoDBLP Masters Thesis''
  },
  {
    ``_id'': ``<http://lsdis.cs.uga.edu/projects/semdis/opus#Doctoral_Dissertation
        >'',
    ``name'': ``SwetoDBLP Doctoral Dissertation''
  },
]
```

File B.15: SwetoDBLP Frontend categories file.

Using the KB

The SwetoDBLP KB is working successfully in the system, as seen in Figure B.7. However, property and range labels, and property comments can not be extracted from the ontology using "rdfs:label" nor "rdfs:comment", so they are derived from the URI. In contrast, the labels used for the suggested value are obtained from the KB using the "<http://www.w3.org/2000/01/rdf-schema#label>" property (as seen in Figure B.7).

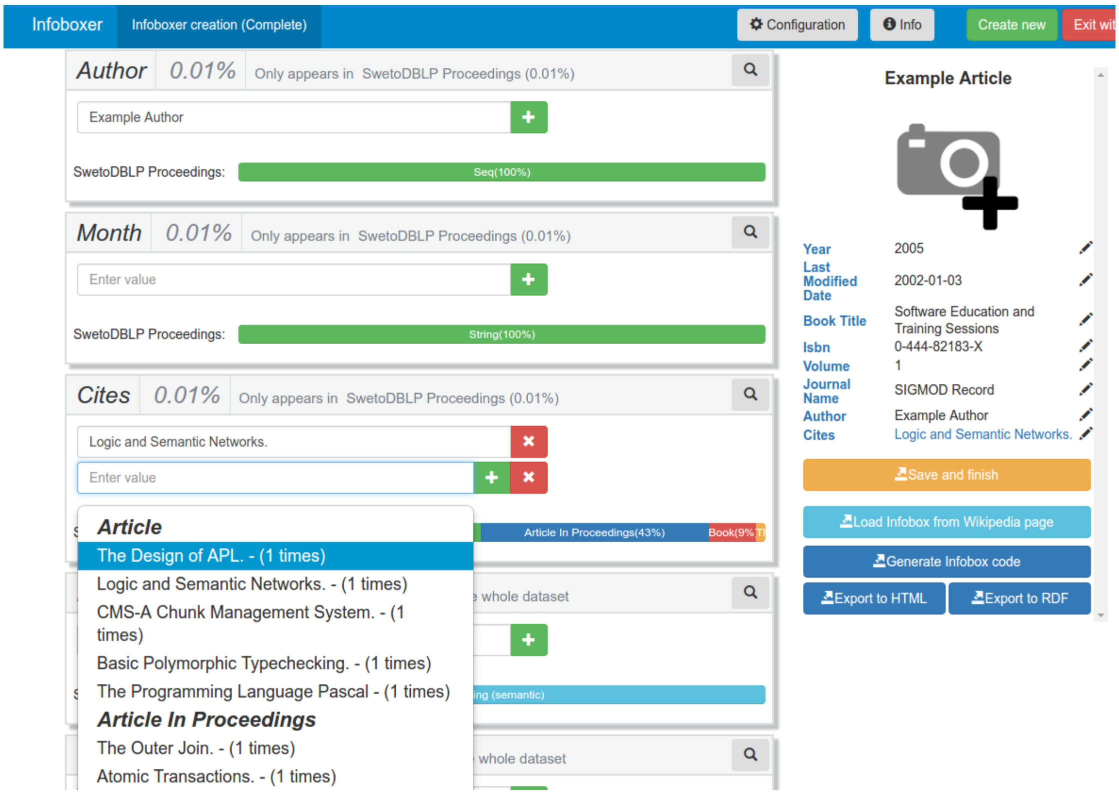
There are seven main classes in the KB, which are shown in Table B.12 along the number of instances of each one. For a class with a representative number of instances ("Proceedings"), a total of 36 properties were retrieved, being 15 of them statistical and 21 semantic. Some statistical properties are shown in Table B.8, with the number and percentage of uses and the suggested ranges with their use frequency. Regarding the semantic properties, the most popular are "Pages", "Cdrom" and "Gmonth".

Class	Number of instances
Book	1235
Proceedings	9027
Article in Proceedings	561895
Webpage	10610
Article	340488
Masters Thesis	8
Doctoral Dissertation	89

Table B.12: Classes and number of instances for SwetoDBLP.

Property	Last Modified Date	Isbn	Cites
Uses of property	9027 (100%)	7769 (86.06%)	1 (0.01%)
Shown ranges	Date (100%)	String (100%)	Article (46%) Article in Proceeding (43%) Book (9%) Thing (2%)

Table B.14: Properties for ”Proceedings” class, with their ranges and uses.



Author 0.01% Only appears in SwetoDBLP Proceedings (0.01%)

Example Author

SwetoDBLP Proceedings: Seq(100%)

Month 0.01% Only appears in SwetoDBLP Proceedings (0.01%)

Enter value

SwetoDBLP Proceedings: String(100%)

Cites 0.01% Only appears in SwetoDBLP Proceedings (0.01%)

Logic and Semantic Networks.

Enter value

Article
The Design of APL. - (1 times)
Logic and Semantic Networks. - (1 times)
CMS-A Chunk Management System. - (1 times)
Basic Polymorphic Typechecking. - (1 times)
The Programming Language Pascal - (1 times)
Article In Proceedings
The Outer Join. - (1 times)
Atomic Transactions. - (1 times)

Article In Proceedings(43%) Book(9% T

whole dataset

ing (semantic)

whole dataset

Example Article

Year	2005
Last Modified Date	2002-01-03
Book Title	Software Education and Training Sessions
Isbn	0-444-82183-X
Volume	1
Journal Name	SIGMOD Record
Author	Example Author
Cites	Logic and Semantic Networks.

Figure B.7: Screenshot: prototype using SwetoDBLP KB.

B.4 Exposed HTTP API

Here the exposed operations through HTTP by the Web Server are described. For each operation, a description of the returned data, parameters and an example response is provided.

Some aspects have to be taken into account when making HTTP requests:

- All parameters are passed in the URL (e.g: */infoboxer/propertyList?classList=...&language=es*).
- All parameters have to be in URL-encoded format.
- All categories, ranges, and properties passed by parameter have to be in URI format (e.g: *<http://dbpedia.org/ontology/SoccerPlayer>*) and surrounded by "<" and ">".
- All parameters are mandatory unless indicated.

"Instance count" Operation

- **Path:** /infoboxer/instanceCount
- **Returns:** number of instances that belong to at least one of the indicated categories.
- **Parameters:**
  - *classList*: category list in URI format, sepparated by commas.
- **Response example:**

```
{"count":161}
```

Result B.16: Example result of "Instance count" HTTP operation.

"Property list" operation

- **Path:** /infoboxer/propertyList
- **Returns:** List of properties used at least once by instances that belong to one or more classes of the given class list. For each property, it is included the URI, the label, the number of instances of the given classes that manifest that property, a comment if available and a flag that indicates if is semantic or not. If it is semantic, a "rangeForSemantic" object is added, which contains the semantic range for that property. Also, in that case, the count represents the number of instances in the whole KB that use the property.
- **Parameters:**
  - *classList*: category list in URI format, sepparated by commas.

- *language (Optional)*: “es” for Spanish, “en” for English, “fr” for French... Default is “en”.
- *semantic (Optional)*: if set to true, semantic properties will be returned. Else, only statistic properties will be returned.

- **Response example:**

```
[
    {
     "_id":"<http://dbpedia.org/ontology/birthPlace>",
     "count":1,
     "label":"Birth place",
     "semantic":false,
     "comment":"where the person was born",
     "rangeForSemantic":null,
    },
    {
     "_id":"<http://dbpedia.org/ontology/child>",
     "count":1,
     "label": "Child",
     "semantic":true,
     "comment":null,
     "rangeForSemantic": {"_id": <http://dbpedia.org/
        ontology/Person>, "label": "Person"}
    }
]
```

Result B.17: Example result of “Property list” HTTP operation.

“Range list” Operation

- **Path:** /infoboxer/rangesAndUses
- **Returns:** given one or more categories it returns, for each property used by instances of those categories, a list of ranges and, for each range, the number of uses. For each property, the three most used ranges are returned, and a fourth one grouping the rest is given when there are more than three ranges.
- **Parameters:**
  - *classList*: category list in URI format, sepparated by commas.
  - *language (Optional)*: “es” for Spanish, “en” for English, “fr” for French... Default is “en”.
- **Response example:**

```
[  {
      "key":"<http://dbpedia.org/ontology/occupation>",
      "value":[
         {
            "_id":"owl#Thing",
            "count":79,
            "label":"Thing"
         }
      ]
   },
   {
      "key":"<http://dbpedia.org/ontology/nationality>",
      "value":[
         {
            "_id":"owl#Thing",
            "count":5,
            "label":"Thing"
         },
         {
            "_id":"<http://dbpedia.org/ontology/Country>
               ",
            "count":57,
            "label":"Country"
         }
      ]
   } ]
```

Result B.18: Example result of "Range list" HTTP operation.

"Class list" Operation

- **Path:** /infoboxer/classList
- **Returns:** list of classes descendant of the given category, with the number of instances of every class. Only a maximum of 50 results are retrieved, according to the introduced text.
- **Parameters:**
  - *superClass*: class in URI format whose children classes are retrieved.
  - *label (Optional)*: text used for filtering results.
  - *language (Optional)*: "es" for Spanish, "en" for English, "fr" for French... Default is "en".
- **Response example:**

```
[   {"_id":"<http://dbpedia.org/ontology/Athlete>", "
   count":141253,"semantic":false},
    {"_id":"<http://dbpedia.org/ontology/Saint>","count"
       :1763,"semantic":false},
        {...} ]
```

Result B.19: Example result of "Class list" HTTP operation.

“Suggestion list” Operation

- **Path:** /suggestions
- **Returns:** list of suggested values for the given list of classes, property and range, sorted by frequency and filtered by the given text “label”. If “property” and “classList” parameters are not provided, the suggested values will be values with type “rangeType”.
- **Parameters:**
  - *classList (Optional)*: category list in URI format, sepparated by commas.
  - *property (Optional)*: property in URI format.
  - *rangeType*: range in URI format.
  - *label (Optional)*: text used for filtering results.
- **Response example:**

```
[
      {
            "_id":"<http://dbpedia.org/resource/Rusia>",
            "count":8,
            "label":"Rusia"
      },
      {
            "_id":"<http://dbpedia.org/resource/United\
               _States\_of\_America>",
            "count":8,
            "label":"United States of America"
      }
]
```

Result B.20: Example result of “Suggestion List” HTTP operation.

“Data for instance” Operation

- **Path:** /fromWikipedia
- **Returns:** Given a Wikipedia page URL, it finds an equivalent instance in the KB and returns a list of its properties, and, for each one, a list of values and their concrete types.
- **Parameters:**
  - *wikipediaUrl*: Wikipedia page URL.
- **Response example:**

```
[  {
      "key":"<http://dbpedia.org/ontology/profession>",
      "value":[
         {
            "uri":"<http://es.dbpedia.org/resource/
               Registro\_de\_la\_propiedad>",
            "label":"Registro de la propiedad",
            "type":"unknownRDFType"
         }
      ]
   },
   {
      "key":"<http://dbpedia.org/ontology/successor>",
      "value":[
         {
            "uri":"<http://es.dbpedia.org/resource/Angel
               \_Acebes>",
            "label":"Angel Acebes",
            "type":"<http://dbpedia.org/ontology/
               President>"
         },
         {
            "uri":"<http://es.dbpedia.org/resource/
               Rodrigo\_Rato>",
            "label":"Rodrigo Rato",
            "type":"<http://dbpedia.org/ontology/
               President>"
         }
      ]
   } ]
```

Result B.21: Example result of “Data for instance” HTTP operation.

B.5 SPARQL Queries

In this section the SPARQL queries used in every important operation are detailed. As the used SPARQL endpoint does not optimize the queries like a Database Management System would do, some operations have been decomposed in smaller ones and additional processing was needed in some of them. The shown queries are the result of an optimization process, the fastest queries among a lot of tried alternatives. All of them can be used with one or more categories.

“Instance count” Operation

This is a simple operation that requires no further processing, only a SPARQL query (see Listing B.22).

```
SELECT (COUNT (DISTINCT ?name) AS ?pcount)
{
      {?name a "class1" } UNION ... UNION {?name a "class2"}
}
```
Query B.22: SPARQL query for “Instance count” operation.

“Property list” Operation

This operation, as mentioned in Section 4.1.2, retrieves the list of statistical and semantic properties for a given list of categories. First obtains a list of statistical properties and its frequency from the KB, using the query on Listing B.23. Then, the list of semantic properties is retrieved from the ontology, and for each one, the count of instances in the whole KB that use that property (query on Listing B.24). Finally, the results are combined.

```
SELECT DISTINCT ?property (COUNT(DISTINCT ?name) as ?count)
{

    { ?name a "class1" } UNION {?name a "classN"}
    ?name ?property ? value

}
GROUP BY ?property
```
Query B.23: SPARQL query for “Property list” operation.

```
SELECT (COUNT(DISTINCT ?name) AS ?count)
{
    ?name "semanticProperty" ?value
}
```
Query B.24: SPARQL query to obtain the number of instances that use a property in the whole KB.

“Range list” Operation

This operation, as mentioned in Section 4.1.2, is decomposed in multiple queries.

1. First, two operations for obtaining a list of instances are performed. One of them obtains the instances that belong to at least one of the given categories (Listing B.25), while the other retrieves the instances that belong to all the given categories (Listing B.26). The first one is used for obtaining ranges, and the second one for obtaining suggestion values.

2. For each retrieved instance of the first list, all the triples which have the instance as a subject along the types for every value are retrieved (see query on Listing B.27 and example result on Listing B.29). Note that, if in the Backend configuration it has been indicated that a value of a property can

be used as the label for the retrieved instances, the query on Listing B.28 is performed. Its difference with the previous one is that it also retrieves, for each value, its label according to the indicated property ("PROPERTY" in the example).

3. Then, for each different retrieved value, its most concrete type is obtained using the ontology and reasoner. In the case that the type is not known (like in "Leytonstone", the value of "BirthPlace" in the example), a default one is assigned (usually "owl#Thing").

```
SELECT DISTINCT ?name {

{?name a "class1"} UNION {?name a "class2"}

}
```

Query B.25: SPARQL query to obtain instances of at least one of the given categories.

```
SELECT DISTINCT ?name {

{?name a "class1"} .
{?name a "class2"}

}
```

Query B.26: SPARQL query to obtain instances of all the given categories.

```
SELECT DISTINCT ?prop ?value ?type
{
    "instance" ?prop ?value .
    OPTIONAL {?value a ?type}

}
```

Query B.27: SPARQL query to obtain properties values and types for an instance.

```
SELECT DISTINCT ?prop ?value ?type ?label
{
    "instance" ?prop ?value .
    OPTIONAL {?value a ?type} .
        OPTIONAL {?value PROPERTY ?label}

}
```

Query B.28: SPARQL query to obtain properties values types and labels for an instance.

```
PROPERTY        VALUE                      TYPE
PlaysOn         Madrid_F.C.                Thing
PlaysOn         Madrid_F.C.                Soccer_Club
Name            "David"^^xsd:String
BirthPlace      Leytonstone
```

Query B.29: Example result of the SPARQL “Range list” operation

“Suggestion list” Operation

This operation, that returns a list of value suggestions for a certain category list, property, and range, takes its data from the cache file generated by the “Range list” operation and complements it with values from all the KB that belong to that range. Those values are obtained with the SPARQL query previously shown in Listing B.25.

“Data for instance” Operation

This operation, that obtains all the properties and values of an instance equivalent to the given Wikipedia page URL, uses the query previously shown in Listing B.27, with little additional processing.

Appendix C

Evaluations

This appendix gathers the collected data in the different performed tests. Firstly, the results of a comparison between different system approaches are provided. Then, data of the three different evaluations where users participated are shown.

C.1 Comparisons Between Approaches

In this section, the results of multiple comparisons of time and disk space between different approaches are shown. These results support the decision of using a Fuseki SPARQL Endpoint over a MongoDB database because its greater performance in the application context, and specifically a modified version that uses the HDT compression format, because the little used disk space.

Time Comparison between MongoDB, Fuseki HDT and Fuseki RDF

Table C.1 shows the results of an evaluation that compares the time that different versions of the system took to generate a template for the "SoccerPlayer" and "Basketball Player" categories, using DBpedia 2015-04. The shown times are the arithmetic mean of three different executions. The version that uses MongoDB was an early prototype discarded because of its inefficiency, and the system version that uses a Fuseki SPARQL Endpoint (HDT or RDF) is the used in the developed system.

As it can be seen on the results, using the Fuseki SPARQL endpoints more time is spent on the first queries, but the expensive process of generating intermediate data and importing it to MongoDB is not required. In the three approaches, the second and next queries are quicker because of the built response file cache. The Fuseki HDT and RDF versions take approximately the same time, and are far quicker than the MongoDB version.

	MongoDB	**Fuseki HDT**	**Fuseki RDF**
Generate SoccerPlayer intermediate data	1331.18 s	N/A	N/A
Generate BasketballPlayer intermediate data	203.66 s	N/A	N/A
Import to MongoDB	66.40 s	N/A	N/A
1st SoccerPlayer query	92 s	184.22 s	191.45 s
1st BasketballPlayer query	91 s	17.44 s	17.57 s
1st multicategory query	95 s	198.99 s	189.28 s
Next SoccerPlayer queries	0.5 s	0.5 s	0.5 s
Next BasketballPlayer queries	0.5 s	0.5 s	0.5 s
Next multicategory queries	0.5 s	0.5 s	0.5 s
Total one query	1879.24 s	400.65 s	398.3 s
Total two queries	**1880.74 s**	**402.15 s**	**399.8 s**

Table C.1: Required time to create “SoccerPlayer” and “BasketballPlayer” templates.

Detailed Time Comparison between Fuseki HDT and RDF

In this section, more detailed results of the comparison between the two SPARQL endpoints (Jena Fuseki, that uses plain RDF, and a modified version that uses HDT to store the data) are given.

Table C.2 shows the difference in the average time required to generate a semantic template for the categories “Body Builder”, “Actor”, and “Governor”, its combination (BB&G&A), “Soccer Player”, “Basketball player”, and its combination (SP&BP) using the RDF and the HDT versions of the used Fuseki SPARQL Endpoint. Notice that, for the combinations, the number of instances to process is exactly the sum of the number of instances of each category, as no instance in the KB belongs to all the categories of the combinations. Also, as the prototype implements a caching mechanism, this time is needed the first time the semantic template has to be created.

Tables C.3 and C.4 show the time took on the generation of templates of one category, broke down by each of the performed operations, and using the HDT endpoint and RDF, respectively. Tables C.6 and C.6 show the broke down times on the generation of templates for multiple categories (BB&G&A and SP&BP). The results show that both approaches are pretty similar in times; the differences are very little.

Category	# instances	Time needed (s) Fuseki RDF	Time needed (s) Fuseki RDF HDT	Speedup (%)
Body Builder	235	0.54	0.48	5.9
Governor	2,689	17.24	11.04	21.92
Actor	5,102	12.6	11.27	5.57
BB&G&A	8,026	19.65	21.16	-3.7
Soccer Player	102,618	191.45	184.22	1.92
Basketball Player	9,411	17.57	17.44	0.74
SP&BP	112029	189.28	198.99	-4.87

Table C.2: Required time to create templates using RDF and HDT.

Operation	Soccer Player	Basketball Player	Gover-nor	Ac-tor	Body Builder
# instances	102618	9411	2689	5102	235
Instance Count (s)	0.35	0.03	10.25	0.03	0.01
Property List (s)	17.54	2.21	1.90	2,23	0.09
Range List (s)	166.33	15.20	8.88	9.01	0.38
Total time (s)	**184.22**	**17.44**	**11.04**	**11.27**	**0.48**

Table C.3: Required time of mono-category operations using HDT Endpoint.

Operation	Soccer Player	Basketball Player	Gover-nor	Ac-tor	Body Builder
# instances	102618	9411	2689	5102	235
Instance Count (s)	0.72	0.07	0.58	0.34	0.02
Property List (s)	12.72	1.98	3.64	2.31	0.07
Range List (s)	178.01	15.52	13.02	9.95	0.45
Total time (s)	**191.45**	**17.57**	**17.24**	**12.6**	**0.54**

Table C.4: Required time of mono-category operations using RDF Endpoint.

Operation	BB&G&A	SP&BP
# instances	8026	112029
Instance Count (s)	0.03	0.40
Property List (s)	4.09	16.45
Range List (s)	17.04	165.68
Total time (s)	**21.16**	**182.53**

Table C.6: Required time of multi-category operations using HDT Endpoint.

Operation	BB&G&A	SP&BP
# instances	8026	112029
Instance Count (s)	0.06	0.16
Property List (s)	3.18	10.9
Range List (s)	16.41	167.31
Total time (s)	**19.65**	**178.37**

Table C.8: Required time of multi-category operations using RDF Endpoint.

Disk Space Comparison

In Table C.9 the disk space used by the different storage approaches with DBPedia 2015-04 is detailed. With MongoDB, a file for every category has to be generated, so the size of the five tested categories is specified. With the Fuseki SPARQL endpoint, either the RDF or the HDT version, all the KB is loaded into it, so the size of the full KB is shown. The results show that the HDT version of the Fuseki SPARQL endpoint is the approach that occupies less space disk. As RDF and HDT were similar in means of time to generate the templates, the disk space was decisive to choose HDT over RDF. MongoDB also would take too much space to store all the categories from a KB, and was discarded because its bad performance in the system context.

	MongoDB	**Fuseki RDF**	**Fuseki HDT**
Soccer Player	627.4 MB	3 GB (All the KB)	714 MB (All the KB)
Basketball Player	67.2 MB		
Governor	18.5 MB		
Actor	29.2 MB		
Body Builder	0.859 MB		

Table C.9: Disk space used with MongoDB, Fuseki RDF and HDT approaches.

C.2 Editatón por la visibilidad de las mujeres de Aragón

This section provides data gathered in the "Editatón por la visibilidad de las mujeres de Aragón" (*Edit-a-thon* for the visibility of women in Aragón), whose results were analysed in Section 4.2. In Table C.11, for every infobox created in the event, it is specified the took time, page name, categories selected, properties filled, how many of them could be linked, and the number of them that were linked.

ID	time (s)	Page	Categories	Props.	Linkable	Linked
12	236	Carmen Magallón Portolés	Scientist	6	4	3
14	1020	Eva berlanga camacho	Scientist	6	4	3
15	474	María Lostal	Resistance Member	3	2	2
17	411	María Jesús Fernández	Scientist	17	11	2
19	194	María Jesús Fernández	Scientist	12	8	0
20	256	Luisa Gavasa Moragón	Actress	7	4	1
22	136	María del Carmen García	Historian	3	3	3
23	600	Ana Labordeta de Grandes	Actress	5	2	0
26	180	Paula Ortiz Álvarez	Screenwriter	8	5	4
27	97	Paula Ortiz	Screenwriter	8	5	4
29	633	María Josefa Yzuel Giménez	Scientist and Coach	15	10	4
30	669	María José Pueyo Bergua	Athlete	9	7	0
31	386	Belen Masia	Scientist	9	6	4

Table C.11: Time, categories, properties and linked properties on every infobox created in the first Wikipedia *edit-a-thon*.

C.3 Wikinformática 2016

In this section, some results of the "Wikinformática 2016" *edit-a-thon* are detailed. First, raw data of the creation of infoboxes of scientists is shown. Then, the evaluation of the offered ranking of properties and its results are described.

Data of Created Infoboxes

In Table C.12 the raw data that supports the Figure 4.7 (Section 4.2) is shown. For every scientist infobox created, the seconds it took and the page name are detailed. Also, the number of properties filled, how many were linkable and the quantity of linked properties is provided. Note that linkable properties are properties whose introduced values (for example, *"Zaragoza"* for birth place) were already in the KB, hence they could be linked.

Evaluation of Property Ranking

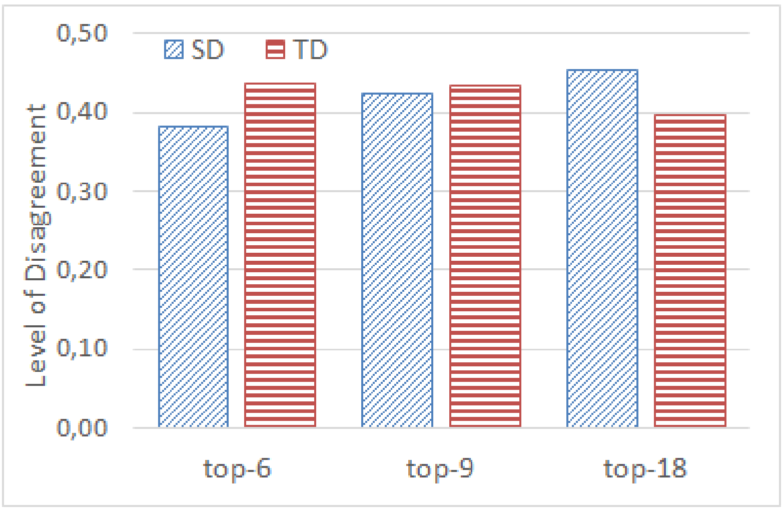


Figure C.1: Tester Disagreement (TD) and System Disagreement (SD) for the Wikinformática edit-a-thon test.

The performed evaluation of the property ranking offered to the user focused on the infoboxes of the "Scientist" category, and compared the order of properties generated by the system with the order of properties that the users filled in. To compare these rankings, the approach described in [36] was used for the top 6 (the average number of properties for Scientists in DBpedia), 9 (the average of properties filled in by the users), and 18 (the maximum number of properties): first, the *generalized Kendall's tau* [12], a measure widely used to compare rankings, was computed. Then, a global level of disagreement between all the users and the system, and a global level of disagreement among the users, *SD* and *TD* in [36], respectively, was obtained. If $SD \leq TD$, the disagreement between the users and the system is not higher than the disagreement between the users themselves. This happens for the top-6 and top-9 rankings (see Figure C.1) but for the top-18 the disagreement of the system is slightly higher. This occurs because the infoboxes

ID	**time** (s)	**Page name**	**Properties**	**Linkable**	**Linked**
9685	550	Frances Elizabeth Allen	9	4	4
9687	728	Anita Borg Naffz	14	4	2
9697	551	Augusta Ada Lovelace (Ada Byron)	9	4	1
9698	535	María López	9	4	2
9703	434	Alexandra Ferrerón	9	3	1
9706	210	Alexandra Ferrerón	9	4	0
9707	302		6	3	2
9710	903	Nieves Rodríguez Brisaboa	7	5	4
9711	550	Nieves Rodríguez Brisaboa	7	5	4
9712	513	Nieves Rodríguez Brisaboa	5	3	3
9713	402	Nieves Rodríguez Brisaboa	5	4	3
9714	393	Nieves Rodríguez Brisaboa	6	4	1
9716	148	Nieves Rodríguez Brisaboa	6	4	3
9720	667	Nieves Rodríguez Brisaboa	7	4	4
9722	1670	Anita Borg	15	6	5
9723	1255	Frances Elizabeth Allen	18	11	7
9726	102	Anita Borg	10	6	4
9734	500	María López	7	5	5
9736	469	Asunción Gómez Pérez	13	6	5
9737	774	Ana Cristina Murillo	7	2	2
9740	155	Arantza Illaramendi	4	3	2
9747	708	Anita Borg	15	6	2
9752	991		8	6	2
9753	915	Alicia Asín	14	6	0
9754	848		11	6	5
9764	1413	Susan Kare	10	4	3
9765	709	Carol Shaw	11	4	0
9768	271	Carol Shaw	11	3	1
9772			8	4	3
9776	399	Ada Lovelace	6	4	3
9777	333	Ada Lovelace	8	4	3
9779	394	Ana Cristina Murillo Arnal	8	3	0
9781	383	Alicia Asín	13	7	0
9788	320	Ana Cristina Murillo	5	4	3
9798	437	Celia Sánchez - Ramos Roda	9	3	0
9827	187		11	4	1
9830	698	ada lovelace	9	5	0
9833	285	María López Valdes	7	4	0
9834	215	Ada Lovelance	9	5	0
9859	721	Frances Elizabeth Allen	10	6	7
9861	780	Anita Borg	10	6	4
9863	566	Frances Elizabeth Allen	7	5	0
9865	438	Margarita Salas	8	5	0
9867	827	Barbara H. Liskov	9	4	0
9868	459	Barbara H. Liskov	11	6	0
9870			15	11	11
9873		Ana Cristina Murillo	14	10	10

Table C.12: Time, properties, and linked properties on every Scientist infobox created on Wikinformática

created with high number of properties made use of “semantic properties” which are not used by any instance of the scientist category in the KB (e.g., “eye colour” or “sibling”).

C.4 Amazon Mechanical Turk Tests

In this section, data gathered on the test set in Amazon Mechanical Turk (*MTurk*), explained in Section 4.2, are shown. Note that conclusions about the results are commented in that chapter.

Data of Created Infoboxes

Table C.13 shows information about the Infobox created by every user using the developed system. The information about the infoboxes created using the Wikipedia mechanism for those same users is shown on Table C.14. Note that, in the Wikipedia case, the linkable properties are those whose introduced values have a Wikipedia page.

ID	**Time (s)**	**Properties**	**Linkable**	**Linked**
8877	434	15	9	7
8879	722	14	7	4
8884	372	13	5	0
8890	425	14	6	4
8891	515	12	4	2
8894	159	11	5	3
8897	759	33	24	8
8898	464	20	10	5
8899	506	25	14	8
8900	193	9	2	1
8962	1048	11	4	1

Table C.13: Information about the infoboxes created on MTurk tests, using the developed system.

Survey Results

Table C.16 shows, for every user that participated in the test performed in Amazon Mechanical Turk, data such as age, education, experience using Wikipedia (WP), and which system he/she finds easier and faster to use. Then, Table C.18 and C.20 show the advantages and disadvantages of the developed system, according to each user. Note that the developed prototype is mentioned as *"Infoboxer"*.

ID	Time (s)	Properties	Linkable	Linked
8877	485	14	11	0
8879	492	20	10	0
8884	370	10	7	0
8890	426	9	6	0
8891	356	10	8	0
8894	489	9	7	0
8897	241	5	5	0
8898	227	6	4	0
8899	431	16	13	0
8900	103	5	3	0
8962	434	8	6	0

Table C.14: Information about the infoboxes created on MTurk tests, using the Wikipedia mechanism.

ID	Age	Education	Experience using WP	Experience editing WP	Infoboxes created before	Easier system	Faster system
8877	25-40	High School	Very familiar	No	0	Infoboxer	Infoboxer
8879	25-40	High School	Very familiar	No	0	Infoboxer	Infoboxer
8884	40-55	High School	Very familiar	No	0	Infoboxer	Infoboxer
8890	25-40	High School	Occasional	No	0	Wikipedia	Wikipedia
8891	< 25	Bachelor's Degree	Very familiar	No	0	Infoboxer	Infoboxer
8894	40-55	Undergrad. program	Occasional	No	0	Wikipedia	Infoboxer
8897	40-55	High School	Very familiar	No	0	Infoboxer	Wikipedia
8898	< 25	High School	Occasional	No	0	Infoboxer	Infoboxer
8899	40-55	Bachelor's Degree	Very familiar	No	0	Wikipedia	Wikipedia
8900	40-55	High School	Very familiar	No	0	Infoboxer	Infoboxer
8962	40-55	Master's Degree	Occasional	No	0	Infoboxer	Wikipedia

Table C.16: Information of the users that participated in the MTurk test.

ID	Comment
8877	It has a great layout and it makes it simple to search for things with the click of a button.
8879	I think it looks easier therefore making me feel like I am better qualified to use it.
8884	You can search easier.
8890	Unsure.
8891	It is easier to input the information.
8894	It is quicker to input information.
8897	It offers suggestions when you start typing.
8898	It's simple and clear.
8899	it fills in some information for you.
8900	Looks better and easier to read and figure out what data i needed to put in the boxes.
8962	The Infoboxer tool is neat and systematic, with slots for specific information making it smooth and streamlined.

Table C.18: Advantages of the developed system according to MTurk users.

ID	Comment
8877	It looks for specific information, so small typos could bring up different results.
8879	User friendly.
8884	Not sure what to do when info isn't relevant.
8890	Confusing.
8891	There are many irrelevant categories.
8894	You still have to search out the information to be input.
8897	It is slower to use.
8898	None.
8899	The way the boxes are aligned make it difficult to work smoothly.
8900	Could not see one.
8962	You need to break up the information into specific parts to suit the Infoboxer's requirement which makes it time consuming.

Table C.20: Disadvantages of the developed system according to MTurk users.

Appendix D

User Manual

In this appendix, the application functioning is explained, detailing each screen and its components, so a non-technical user can understand how to use it.

D.1 How to Access the System

To use the system, the user has to access with a web browser (tested on Google Chrome and Mozilla Firefox) to the address configured by the responsible technical person. Then, the login page will appear.

D.2 Login Page

The first view the user sees when he/she enters the prototype is the login page (Figure D.1). It allows the user to select the language, the type of properties shown, and to introduce an username that will be used to identify him/her in the gathered information (interactions, time used, stats...). The elements which appear in the view are:

1. A username input box, where the user must type a username that identifies him/her.
2. Two language selection buttons, where the user can select the interface and shown data language between English and Spanish.
3. Two properties mode selection buttons. The user can decide if statistical and semantic properties are shown, or only the statistical ones.
4. The start button, that requires a username to be introduced. If pressed, it takes the user to the main prototype page.

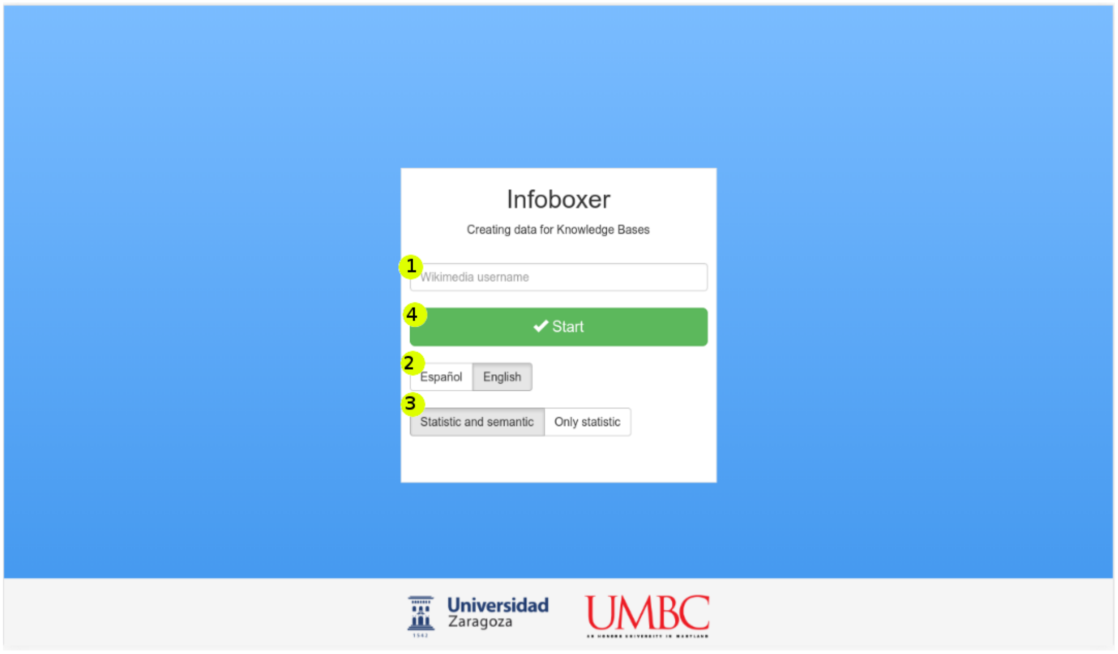


Figure D.1: Screenshot: login page of the prototype.

D.3 Main Page

The next page is the main prototype page (Figure D.2). In it, the user can input the name of created instance and select the appropiate categories that define it. Also, some presentation settings can be changed. The elements appearing in the view are:

1. A text input where the user has to input the name of the instance whose information wants to create. For example, “David Beckham”.

2. A category input where the user can select a category that best fits the instance being introduced. When the user makes click on it, a drop-down list appears, and its results are filtered as the user types. As classes usually have hierarchies, the user can fold and unfold them however he/she likes.

3. An “Add” button to add a new category input, so multiple categories can be selected.

4. A “Load data” button that generates the template for the introduced categories and shows it to the user. It has to be clicked when one or more categories are already selected. When clicked, a “waiting” icon appears while loading, and finally the template appears at the bottom of the page (See “Template page (expert)” and “Template page (simple)”).

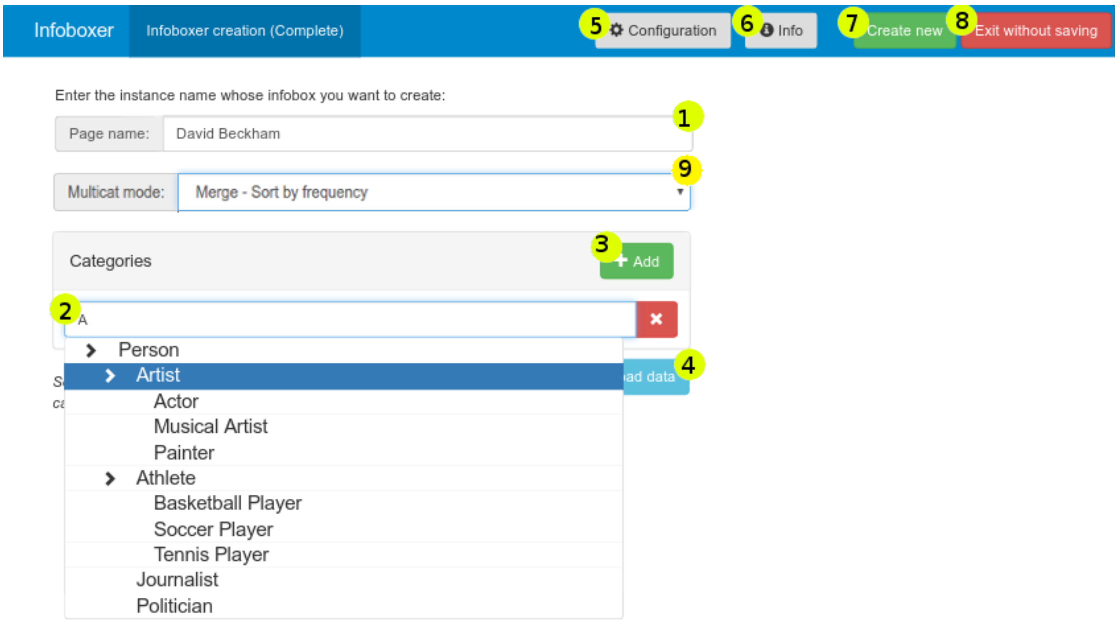
5. A “Configuration” button that shows a floating window where the user can configure presentation settings. It can be seen on Figure D.3 and its elements are:

   (a) Two buttons to select the interface and shown data language.

    (b) Two buttons to change the interface mode between complete/expert and simple/basic mode.

    (c) Two buttons to select what kind of properties are shown.

6. An “Info” button that shows information about the prototype creator.
7. A “Create new” button that discards all the introduced data and reloads the page.
8. A “Close without saving” button that takes to the login page without generating RDF nor Infobox code.
9. A dropdown, “Multicat mode”, that only appears in Expert mode when more than one category has been selected. It allows the user to select if properties are merged or not.

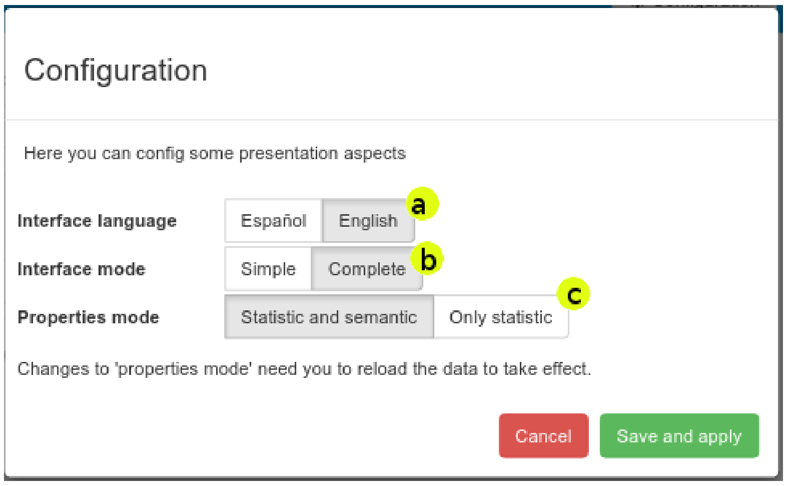


Enter the instance name whose infobox you want to create:

Page name: David Beckham

Multicat mode: Merge - Sort by frequency

Figure D.2: Screenshot: main page of the prototype.


Configuration

Here you can config some presentation aspects

Interface language: Español [✓ English]

Interface mode: Simple [✓ Complete]

Properties mode: [✓ Statistic and semantic] Only statistic

Changes to 'properties mode' need you to reload the data to take effect.

Figure D.3: Screenshot: config windows of the prototype.

D.4 Template Page (Expert)

This page (Figure D.4) is accessed when the user selects one or more categories, clicks on the "Load data" button and the Expert mode is selected. It shows the generated template in the left side, allowing the user to introduce values for the given properties, and simulates an Infobox in the right side. The elements appearing in this page are:

1. The list of statistical and semantic properties that form the generated template. It can be scrolled, and a pagination system is present. Each property has the following components:
   (a) The name or label of the property in the selected language.
   (b) The frequency of the property (percentage of instances of at least one of the given categories that manifest that property).f
   (c) The frequency of the property on each individual selected category.
   (d) A search button that opens a browser window with a Google Search about the property value for the instance being created.
   (e) A value input where the user can introduce the value for a property or select one from the provided suggestion list (1h).
   (f) A button that adds another value input to the property, so more than one value can be introduced.
   (g) A ranges bar that show the used ranges for that property and their frequencies. When multiple categories are selected, first a bar for the combination of categories is shown, and then one for each individual class.
   (h) A dropdown list of suggestions. Suggestions are grouped by the ranges shown in the top bar, and in this mode, the count of uses is also shown. The provided suggestions change as the user types to propose the value

he/she is looking for. When a value is clicked, it is automatically entered in the value input.

2. A preview of the entity being generated, using for that a visual format similar to Wikipedia infoboxes Each time a value is modified in the left side of the interface, the Infobox is updated in real time. It has the following components:

    (a) The name of the instance being inserted.

    (b) A thumbnail of image of the instance. Doing click on the camera icon allows the user to enter an image URL that will be shown there.

    (c) The list of entered values. If a value is linked, it appears like an hyperlink (blue). The pencil icon at the right side of each value moves the interface directly to were that value was introduced so it can be edited.

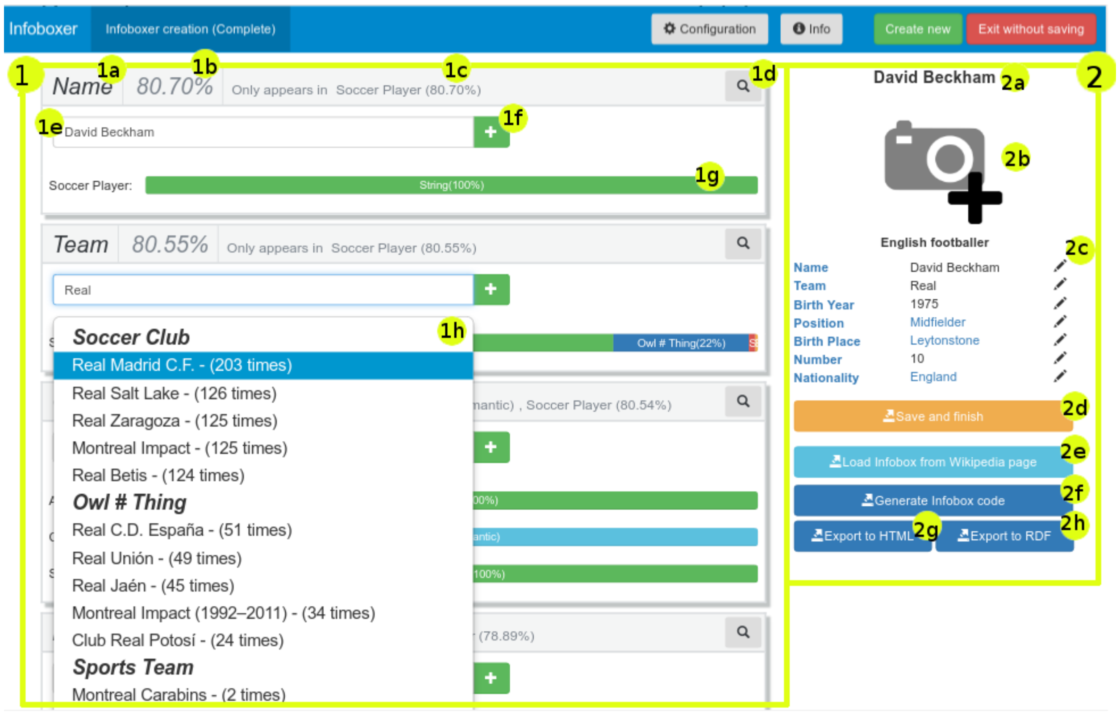
    (d) The "Save and Finish" button generates the Wikipedia Infobox code for the introduced information, saves it to the server, and stops counting the time that takes the user to fill the template. Also, a little survey is shown where the user can express whether the system is easy to use or he/she had any problem.

    (e) When this button is clicked, a Wikipedia Page URL is asked to the user. When it is introduced, the properties are automatically filled according to the information about that page contained in the KB.

    (f) This button generates the Wikipedia Infobox code and shows it to the user.

    (g) This button generates and shows to the user HTML code. This HTML code, when introduced into a website, shows an Infobox similar to the displayed in the prototype.

    (h) This button generates and shows the RDF code belonging to the introduced data.



Name 80.70% Only appears in Soccer Player (80.70%)

David Beckham

Soccer Player: String(100%)

Team 80.55% Only appears in Soccer Player (80.55%)

Real

Soccer Club
Real Madrid C.F. - (203 times)
Real Salt Lake - (126 times)
Real Zaragoza - (125 times)
Montreal Impact - (125 times)
Real Betis - (124 times)
Owl # Thing
Real C.D. España - (51 times)
Real Unión - (49 times)
Real Jaén - (45 times)
Montreal Impact (1992–2011) - (34 times)
Club Real Potosí - (24 times)
Sports Team
Montreal Carabins - (2 times)

Owl # Thing(22%)

David Beckham

English footballer

Name	David Beckham
Team	Real
Birth Year	1975
Position	Midfielder
Birth Place	Leytonstone
Number	10
Nationality	England

Figure D.4: Screenshot: template page (expert) of the prototype.

D.5 Template Page (Basic)

This page (Figure D.5), accessed when the user selects one or more categories, clicks on the "Load data" button and the Simple mode is selected, has the same structure as the Expert mode page, but shows only the basic data. The elements appearing in the page are:

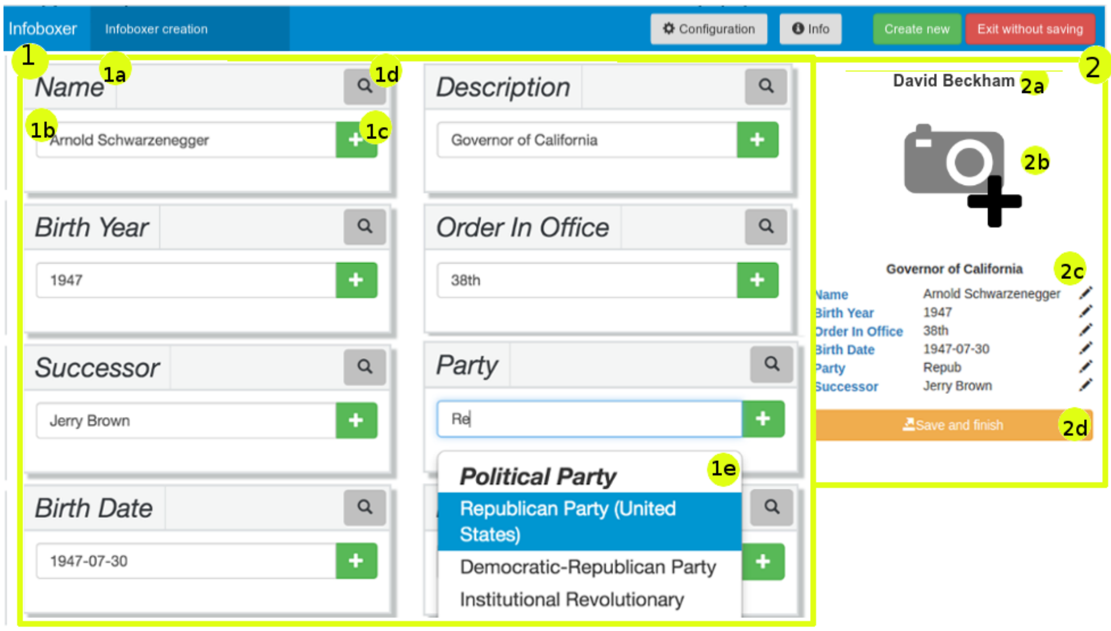
1. The list of statistical and semantic properties that form the generated template. For each property, the following components are shown:
   (a) The name or label of the property, in the language selected by the user.
   (b) An value input where the user can type a value or select one from the suggested values list.
   (c) A button to add another value input, so a property can have more than one value.
   (d) A search button that opens a browser window with a Google Search about the property value for the instance being created.
   (e) A dropdown list that provides suggestions of values, similar to the one in Expert mode, but without displaying the frequency of each value. They are grouped by ranges as in the "Expert mode", but no range bar is shown.
2. A preview of the entity being generated identical to the one shown in the Expert mode. It has the following components:

(a) The name of the instance being inserted.

(b) A thumbnail of image of the instance. Doing click on the image allows the user to enter an image URL that will be shown there.

(c) A list of introduced values with an edit button, similar to the Expert mode.

(d) A "Save and Finish" button that generates the Wikipedia Infobox code for the introduced information, saves it to the server and stops counting the time that takes the user to fill the template. Also, a little survey is shown where the user can express whether the system is easy to use or he/she had any problem.



Name: Arnold Schwarzenegger
Description: Governor of California
Birth Year: 1947
Order In Office: 38th
Successor: Jerry Brown
Party: Re
Birth Date: 1947-07-30

Political Party
Republican Party (United States)
Democratic-Republican Party
Institutional Revolutionary

David Beckham

Governor of California

Name	Arnold Schwarzenegger
Birth Year	1947
Order In Office	38th
Birth Date	1947-07-30
Party	Repub
Successor	Jerry Brown

Figure D.5: Screenshot: template page (basic) of the prototype.