ISSN 1870-4069

Toward a Natural-Language Interface in Ontologies for Disambiguation

Alejandro Solís-Sánchez[1], Pablo Barraza Cornejo[1], Juan Acosta-Guadarrama[1,2], Juan Ferret[2]

[1] Universidad Autónoma de Ciudad Juárez, Mexico
[2] Instituto de Sistemas Filosóficos, USA

{al160509,al164006}@alumnos.uacj.mx, juan.acosta@uacj.mx

Abstract. With natural-language-interfaces-to-ontology (NLI2O) one can query a specific domain's data, stored into that particular conceptual structure, coded into specialized languages such as Ontology Web Language, OWL. To access such data, one requires complex query languages, such as SPARQL. NLI2Os translate human-language queries into SPARQL ones. In this work, we describe the architecture of an NLI2O. It receives a spoken question from an Android device. The spoken query becomes text and goes to a Java Server Page (JSP). The core of our interface runs on a JSP server, which receives the user's text query. It transforms it into a SPARQL query, later used to access, from the ontology to query, the information requested by the user. The interface uses several technologies to achieve that. Those are known as lemmatization, part-of-speech tagging, and tokenization. It also uses knowledge inference from the ontology of interest. It also uses clarification dialogues for disambiguation to answer user questions and revise the query. Besides that, some experiments suggest further studies like the disambiguation of an original text.

Keywords: Natural language interfaces to ontologies, disambiguation, SPARQL, OWL, semantic web.

1 Introduction

The Semantic Web is an extension of the current Web [2], where information maps to a well-defined meaning that can be understood both by computers and humans. The architecture of the Semantic Web that was established by Tim Berners-Lee makes use of ontologies as one of its principal components. An ontology is an explicit specification of conceptualization.

It allows us to understand the structure of knowledge better since it shows concepts and the relationships that exist between them [11]. Natural language[3]

[3] By natural language, we mean what people speak, such as English or any other.

interfaces (NLI) to ontologies translate a Natural Language query into a formal query language, employed to retrieve the knowledge expressed in one of the knowledge representation languages [7].

In the Semantic Web, ontologies store information and represent it through the Web Ontology Language, OWL[4], considered an extension of the *Resource Description Framework* RDF language [18]. Consulting knowledge stored in an ontology requires advanced skills for such purpose; an example is SPARQL (Protocol and RDF Query Language [20]) [14]. For this reason, interfaces that receive a query in natural language and transform it into a SPARQL query are necessary, with which they extract information users request.

The Semantic Web needs interfaces to query ontologies in a language closer to humans'. That facilitates interaction with sites or applications based on its architecture. In this work, we propose an architecture for an interface which, through queries in natural language, allows searching for information stored in an ontology, by using a mobile device for its operation. It is also capable of solving some kind of ambiguities through a simple clarification dialog, which is a promising feature to process non-monotonic languages in future works. The structure of this paper is the following. In Section 2, we add a description of related existing proposals; in Section 3, we describe our project; Section 4 describes the implementation and Section 5 the conclusions.

2 Related Works

There are different architectures proposed for the implementation of natural-language-interfaces-to-ontologies, NLI2O; they use different techniques of natural language processing (NLP) and formal query languages. This current work revisits and extends a preliminary version that appeared in [22], and we include further findings and software as a foundation of our claims[5]. In this section, we describe the architecture of some NLI2Os like FREyA [9], Gingseng [3], QuestIO [23], ORAKEL [4], and AquaLog [16], [22].

2.1 FREyA

FREyA (Feedback Refinement and Extended Vocabulary Aggregation) "is an NLI for querying ontologies which combines usability enhancement methods such as feedback and clarification dialogs," to improve recall and precision [10]. FREyA uses the knowledge available in the ontology to identify the terms found in the query, called *Ontology Concepts*—OC. If they are ambiguous, FREyA generates a clarification dialog in which the user chooses one of several alternatives. In the clarification dialog, the user selects an option that is stored and used to train the system. Starting from the OC, FREyA generates a set of triplets that converts to SPARQL queries to find the answer in the ontology. To

[4] Refer to `https://www.w3.org/TR/owl-features/`.
[5] Further examples and software are available at `http://k-lab.uacj.mx/NLI2O-20/`.

show the results to the user, it first identifies the type of response [9]. Then, it does a syntactic analysis and a search in the ontology to determine the *emphasis* of the question, for each answer type. A *focus* is a word or sequence of words that define a query and disambiguate it, showing what it is looking for [8]. The presentation of FREyA results includes a graph for visualization.

2.2 Ginseng

Ginseng (Guided Input Natural Language Search Engine) is another NLI based on a grammar of queries that extends dynamically through the structure of an ontology to guide users in the formulation of questions in a language similar to English. Based on the grammar, Ginseng translates queries to SPARQL, which allows its execution [3]. It provides query access to any OWL knowledge base. It guides the user to formulate questions, so there is no need to interpret them (logically o syntactically) and does not use any predefined vocabulary. Ginseng only knows the words defined by the currently considered ontologies, and the user has to follow it [3]. That can limit the possibilities of the user in general but ensures that all queries can get answered [3].

Ginseng's architecture consists of three parts: a multilevel grammar, an incremental parser, and an access layer to the ontology. The multilevel grammar consists of a static component that specifies the structures of generally possible query sentences and a dynamic one generated from the utilized ontologies. The static grammatical rules provide the structures and basic phrases for the questions in English. Each ontology loaded onto Ginseng creates the dynamic grammatical rules, and it uses them to extend the part of static syntax [3]. The incremental parser first uses the whole grammar to provide alternatives to the user while inputting the queries, and secondly to store information about how to build SPARQL queries. Finally, they use the Jena framework for the access layer to the ontologies. When the query execution becomes completed, Ginseng shows the generated SPARQL query and the results for the user [3]. The principal characteristic of Ginseng is that, unlike FREyA, queries that the user makes get controlled by a grammar based on the knowledge stored in the ontology. That gives rise to one of its advantages and equally its disadvantage since the grammar gives it control over what the user writes, and therefore, it avoids invalid questions, but at the same time limits the queries the user can make.

2.3 QuestIO System

QuestIO (Question-Based Interface to Ontologies) is an NLI to access structured information. It is domain independent and easy to use without training [23]. It is an open domain (or customizable to new ones with minimal cost), with undefined vocabulary; that is, it is derived automatically from the existing data in the knowledge base [23]. The system works by converting natural language queries to formal SeRQL (Sesame Rdf Query Language, pronounced "circle") questions [1], though other query languages can become used.

The QuestIO working process begins by processing the domain ontology, automatically creating a lexical dictionary from the obtained knowledge. From the ontology, the dictionary is capable of identifying mentions of classes, properties, instances, and property values associated with cases [23].

When QuestIO receives a query, the system does the following:

1. Make a linguistic analysis that consists of a morphological analysis of the text by a tokenizer and tagging tool.
2. Execute the ontological dictionary created at the system's start over the query's text. That creates annotations for all the mentions that the dictionary could identify—classes, properties, instances, and property values of data types.
3. Initiate an iterative transformation process to convert input text into a formal query. First, it separates the input text into tokens, determining each one's part of the discourse and adding annotations with its morphological route. Then, it tries to identify mentions of the ontology resources on the input text for generating a formal SeRQL query.
4. Finally, execute the query on the knowledge base and display the results.

QuestIO is an NLI that, unlike FREyA and Ginseng, uses SeRQL query language, which can be a disadvantage, since the official standard for consulting ontologies in the Semantic Web is SPARQL. Another problem is that it depends on a well-designed ontology for its operation, which generally requires a domain specialist for the ontology's design, making it less accessible to unskilled users.

2.4 ORAKEL

ORAKEL is an NLI to a knowledge base that transforms questions into a logical form [4] since it uses the F-logic language for representation of knowledge [15]. It receives a query in natural language as input, which gets converted to a first-order logic formula. Then, the logic formula gets transformed into the specific query language, which can be SPARQL or the F(rame)-Logic, F-logic, query language [5]. ORAKEL has the following main components.

1. Domain lexicon and general lexicon: a domain specialist creates both.
2. A knowledge base is composed of the domain ontology.
3. FrammeMapper: is a domain specialist that needs to know the underlying knowledge base.
4. Query Interpreter: builds a query formulated by the user to a logical form concerning domain predicates.
5. Query Converter: implemented in Prolog. It receives queries in logical form and translates them to knowledge base language (F-logic).
6. Answer generation.

Unlike other NLI shown in this work, ORAKEL is compatible with F-logic language for knowledge representation. Its main advantage is that it was designed to port NLI between domains efficiently. The disadvantage is that it requires a domain specialist for the lexicon generation.

2.5 AquaLog

Aqualog [16] is a portable question-answer system that takes as input queries expressed in natural language and an ontology and returns answers extracted from semantic markup compatible with the available ontology. Aqualog architecture can get characterized as a cascade model in which a natural language query gets translated into a set of intermediate representations based on triplets. Such triplets shall be compatible with the ontology. The triplet model Aqualog uses is of the form subject-predicate-object [16]. The main Aqualog modules are the linguistic component and the similarity relationship service.

The purpose of the linguistic component is to convert the natural language query to a query in triplets. Aqualog uses the GATE [6] infrastructure and resources to analyze the question as part of a linguistic component. GATE returns a set of syntactical annotations associated with the input query. These annotations include information about sentences, tokens, nouns, and verbs. Aqualog extends the set of annotations returned by GATE, identifying terms, relationships, question indicators (what/who/when/...), and patterns or types of questions.

Aqualog presents a solution that combines different strategies to give sense to a natural language query concerning the universe of discourse covered by the ontology. Using the GATE framework gives it the ability to improve the processing of natural language queries.

Previously analyzed NLI use ontologies to store knowledge, coded in XML files, and extracted through a query language. Except for QuestIO (which uses SeRQL), other interfaces use SPARQL, recommended by the W3C for the Semantic Web[6].

3 Proposed Architecture

The currently developed interface receives a natural language query made by the users in English, translates it to SPARQL [20], and through it, extracts the required information from the user's ontology. Its architecture has two main modules: a) knowledge-generating module (KGM) and b) interface module (IM). The KGM is in charge of generating the knowledge the IM needs to answer the users' natural language queries. In Section 3.1, we describe the KGM, and Section 3.2 describes the IM. In Figure 1, we present the proposed interface's general architecture.

3.1 Knowledge-generating Module

For the IM to be able to answer natural language queries, It is necessary to provide it of knowledge about the structure of the ontology the user wants to query and about the conversational linguistic domain. To facilitate portability to different user ontologies, the KGM is in charge of generating this knowledge, trying to minimize user intervention while configuring the IM.

[6] Refer to https://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation/

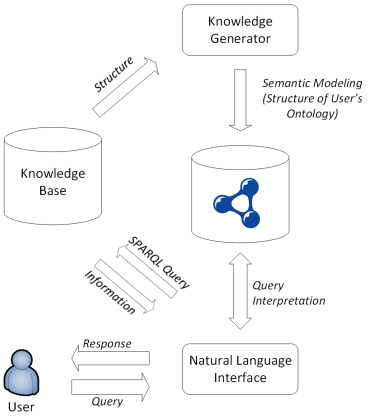

Fig. 1. General Architecture of Proposed Interface.

Firstly, the user indicates to the KGM the ontology they wish to query. Then, the KGM accesses the ontology and analyzes its structure; that is, it identifies the classes, object and data, and the way these elements are related. For this purpose, we used SPARQL query language, coded in OWL [18]. Through SPARQL queries, it can identify the features that make up the ontology's structure and access the stored information, generally modeled individually, that is, as instances of the ontology's classes.

As the next step, the identified elements get modeled semantically by using a semantic representation defined a priori for this purpose. The semantic representation was designed firstly in the software Protégé [25]. Then, to integrate the structure of this semantic representation in the KGM, we used the framework Apache Jena[7], which allows managing ontologies from the Java programming language, in which we designed our proposal.

Next, each of the modeled elements names get associated with synonyms and with words that share the same lemma. That gets done to provide the IM with linguistic knowledge about the domain of the user's ontology. The generated vocabulary becomes part of the semantic model.

Lastly, after generating the semantic model, the KGM stores it in an ontology, created dynamically. This ontology contains the knowledge the IM needs to answer natural language queries formulated by users. In Figure 2, we present the architecture corresponding to KGM, and in Figure 3, we show a fragment of the generated semantic model.

We generated the presented fragment from the ontology Geography.owl[8], used in the evaluation of the FREyA interface by Damljanovic [9]. At the same time, we generated this ontology from the deductive database made by Raymond J. Mooney [24], which contained information about the geography of the United

[7] Refer to `https://jena.apache.org/about_jena/about.html`.
[8] Refer to `https://github.com/danicadamljanovic/freya`.

States and got distributed as example data in Turbo Prolog 2.0. In Figure 4, we show the structure of the ontology Geography.owl.

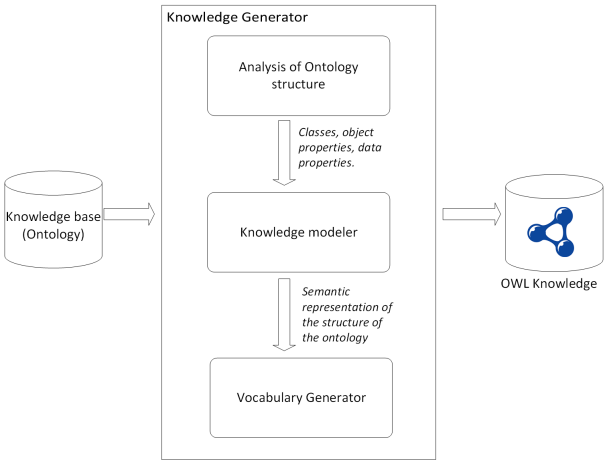

Fig. 2. Knowledge-Generating Module Architecture.

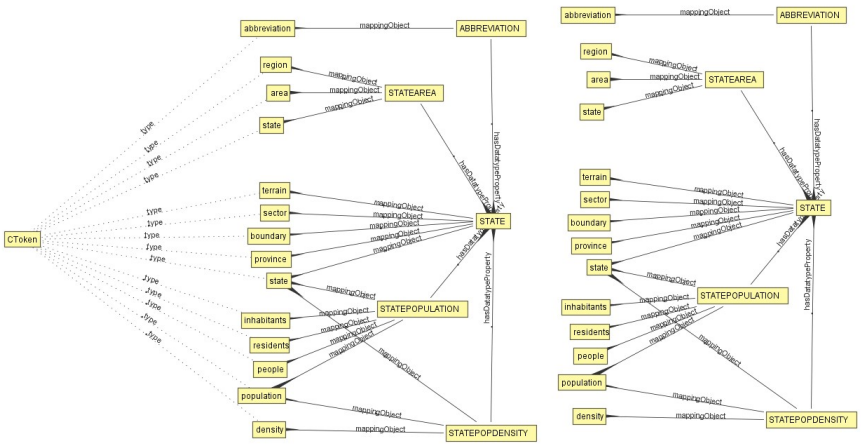

Fig. 3. Generated semantic model fragment.

The classes *CClass, CObjectProperty, CDatatypeProperty*, and *CToken*, make up the semantic representation of Figure 3, besides the object properties *hasDatatypeProperty, mappingObject*, and others.

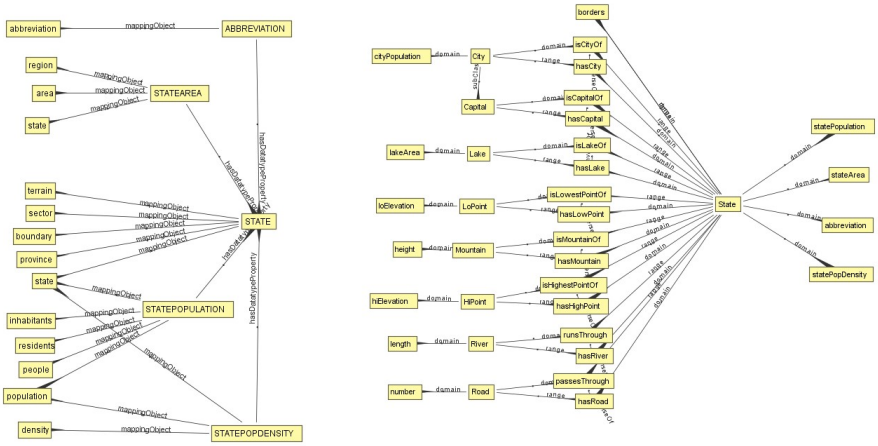

Fig. 4. Ontology structure for Geography.owl.

Figure 3 models the classes *State, Lake, City, Capital, Lake, LoPoint, Mountain, HiPoint, River*, and *Road*, shown in Figure 4 as individuals of the *CClass* class. Figure 3 models object properties *Borders, isCityOf, hasCity, isCapitalOf, hasCapital, isLakeOf, hasLake, isLowestPointOf, hasLowestPoint, isMountainOf, hasMountain, isHighestPointOf, hasHighestPoint, runsThrough, hasRiver, passesThrough* and *hasRoad*, shown in Figure 4, as individuals of the *CObjectProperty* class. Figure 3 models data properties *statePopulation, stateArea, abbreviation, statePopDensity, cityPopulation, lakeArea, loElevation, height, hiElevation, length* and *number*, shown in Figure 4, as class individuals *CDatatypeProperty*.

The class *CToken* is useful for modeling the interface vocabulary. Considering Figure 3, if the user introduced the word "*population*," IM would know that they are possibly referring to individuals *STATE* (CClass), *STATEPOPULATION* (CObjectProperty), or *STATEPOPDENSITY* (CObjectProperty). In this way, the IM uses the knowledge generated to interpret user's queries and generating the SPARQL query.

3.2 Interface Module

The Interface Module (IM) aims at translating the user's natural language query to a SPARQL query, with which it extracts from the user's ontology the requested information. Its operation is in Figure 5. The first step done by IM to generate the SPARQL query is to process the natural language of the user's query.

That phase consists of separating the user's query into a set of words, or isolated *tokens*. Afterward, it tags these tokens according to their grammatical function in the user's query. Lastly, it obtains the lemma for each one of the

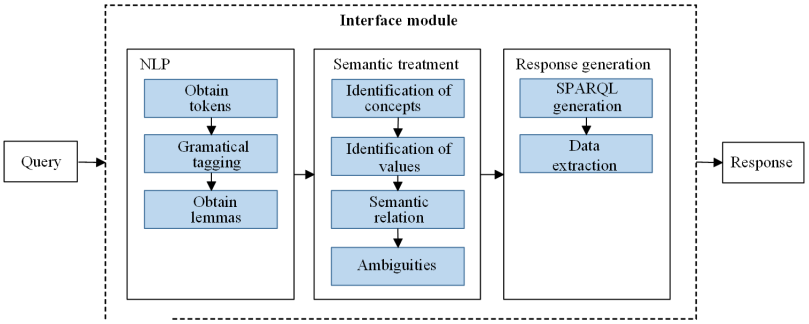

Fig. 5. Interface module operation.

tokens. We achieve this phase by using Freeling [19], a suite of tools for processing natural language which supports different languages, including English and Spanish. Supposing the user introduces the query "*What is the population density of Wyoming?*", the result of this phase is in Table 1, as in [22].

Table 1. Example of natural language processing.

Token	Lemma	PoS	Meaning PoS
What	what	WP	type= interrogative pos= verb
is	be	VBZ	pos= pronoun vform= personal person= 3
the	the	DT	pos= determiner
population	population	NN	pos= noun type= common num= singular
density	density	NN	pos= noun type= common num= singular
of	of	IN	pos=preposition
Wyoming	Wyoming	NP	pos= noun type= proper
?	?	Fit	pos= punctuation type= question mark punctenclose= close

The second step achieved by the IM is a semantic treatment. In this phase, each token, tagged grammatically as a noun or adjective, becomes analyzed. Analysis means to identify if any token maps to a concept, namely, to an individual of the class *CToken* from the ontology generated by the KGM (as it was in the world *population* example at the end of Section 3.1). That gets done through an exact matching of characters. Through the model presented in Figure 3, the IM identifies if the mapping refers to a class, text property, or data property. In case any of these tokens did not map to some element

in the ontology, it searches in the user's ontologies to identify if it is a value. In the previous example, *Wyoming* is a value, since the IM identifies that an instantiated individual from the class *State*. Correctly identifying the mappings and values is essential to generate the SPARQL query. In Table 2, we present the result of this step.

Table 2. Example of identification of mappings and values.

Token	Lemma	PoS	Meaning PoS
What	what	WP	
is	be	VBZ	
the	the	DT	
population	population	NN	Token=http://www.uacj_nlp.com/schema/nlp. OWL#population Class=http://www.uacj_nlp.com/schema/nlp. OWL#CDatatypeProperty Object=http://www.uacj_nlp.com/schema/nlp. OWL#STATEPOPULATION Token=http://www.uacj_nlp.com/schema/nlp. OWL#population Class=http://www.uacj_nlp.com/schema/nlp. OWL#CDatatypeProperty Object=http://www.uacj_nlp.com/schema/nlp. OWL#STATEPOPDENSITY Token=http://www.uacj_nlp.com/schema/nlp. OWL#citypopulation Class=http://www.uacj_nlp.com/schema/nlp. OWL#CDatatypeProperty Object=http://www.uacj_nlp.com/schema/nlp. OWL#CITYPOPULATION
density	density	NN	Token=http://www.uacj_nlp.com/schema/nlp. OWL#density Class=http://www.uacj_nlp.com/schema/nlp. OWL#CDatatypeProperty Object=http://www.uacj_nlp.com/schema/nlp. OWL#STATEPOPDENSITY Token=http://www.uacj_nlp.com/schema/nlp. OWL#population Class=http://www.uacj_nlp.com/schema/nlp. OWL#CDatatypeProperty Object=http://www.uacj_nlp.com/schema/nlp. OWL#in order toTo
of	of	IN	
Wyoming	Wyoming	NP	Individual=http://www.mooney.net/ geo#wyoming Class=http://www.mooney.net/geo#State Label=wyoming@en Individual=http://www.mooney.net /geo#wyomingMi Class=http://www.mooney.net/geo#City Label=wyoming@en
?	?	Fit	

Subsequently, it aims to identify if there is some relationship between any of the tokens. It determines that if, between its mapping, there exists some in common, besides following some *a priori* defined rules. In case of being affirmative, it combines both tokens into a composite one, keeping only the matching or related mappings and discarding the rest of them. The objective is to simplify the generation process of the SPARQL query. As we see in Table 2, the tokens *population* and *density* have in common the STATEPOPDENSITY mapping, which is why the IM combines them into a single token.

Then, the IM processes the identified values. If the latter belong to more than one class, it could cause an *ambiguity issue*. In such a case, *Wyoming* belongs to two of them, *State* (*wyoming*) and *City* (*wyoming*), so it tries to solve the ambiguity identifying the classes that match those of the rest of the queries' elements.

Since the IM determined, on the grounds of Figure 3, that STATEPOPDENSITY is a data property that belongs to STATE, it discards the class *City*. In the case the IM cannot solve the ambiguity, it shows a clarification dialog to the user so they can decide on it. In Table 3, we present the results of such a semantic treatment phase.

The third step is to generate the SPARQL query with the identified elements after the semantic treatment phase, shown in Table 3, and to use such query to extract, from the user's ontology, the requested information. To generate the

Table 3. Example of the semantic treatment phase.

Token	Lemma	PoS	Meaning PoS
What	what	WP	
is	be	VBZ	
the	the	DT	
population density	population density	NN	Token=http://www.uacj_nlp.com/schema/nlp. OWL#population Class=http://www.uacj_nlp.com/schema/nlp. OWL#CDatatypeProperty Object=http://www.uacj_nlp.com/schema/nlp. OWL#STATEPOPDENSITY
of	of	IN	
Wyoming	Wyoming	NP	Individual=http://www.mooney.net/ geo#wyoming Class=http://www.mooney.net/geo#State Label=wyoming@en
?	?	Fit	

SPARQL query, it follows a series of a priori defined rules. For example, based on Figure 3 and Table 3, the IM identified that STATEPOPDENSITY is a data property related to the STATE class. Such elements are defined in the user's ontology by the IM as <http://www.mooney.net/geo#statePopDensity> and as <http://www.mooney.net/geo#State>. Also, *Wyoming* is an individual of the class *State*. For that reason, the IM generated the following SPARQL query, where the variable that forms part of the *Select* clause comes up out of the combined token *population density*, which becomes transformed into *?population_density*. In Figure 6, we show results given by the IM using the generated SPARQL query.

```
Select ?population_density
Where {
        <http://www.mooney.net/geo#statePopDensity>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://www.w3.org/2002/07/owl#DatatypeProperty>.
        <http://www.mooney.net/geo#wyoming>
        <http://www.mooney.net/geo#statePopDensity>
        ?population_density.

        <http://www.mooney.net/geo#wyoming>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://www.mooney.net/geo#State>
}
```

population_density
0.20830059

Fig. 6. Result of the query "*What is the population density of Wyoming*" using the ontology Geography.owl.

4 Implementation

In this section, we describe the implementation of the NLI proposed in this paper.

We describe the software components and hardware devices where they locate. The NLI is based on a client-server architecture, uses an Android app as

a graphical interface and an application server where the query processing gets done. The server works through Java Server Pages (JSP) [12] as a technology for its implementation, which is why we use Apache Tomcat [26], a Java-based application container.

The display diagram of Figure 7 shows the software components and hardware devices used. The system implements the following software components: An Android application, an event controller for inter-component interaction, a knowledge base, a natural language processing component, and another to generate and execute SPARQL queries on the knowledge base.

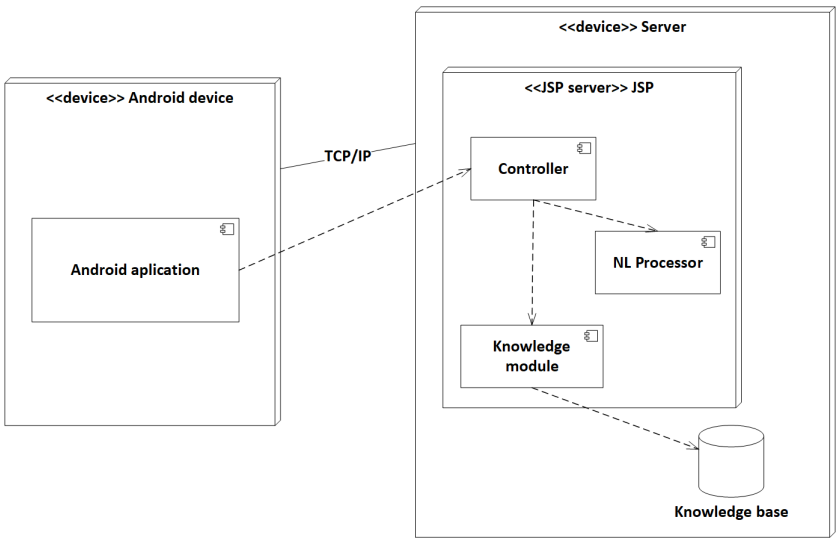

Fig. 7. Display diagram.

The general process for every input query is according to the following steps:

1. The Android app receives the query in a voice phrase as input, converts it into text, and sends it to the controller for processing.
2. The controller receives the query and sends it to the NL processor.
3. The NL processor receives the text query, and out of it, it obtains the tokens, grammatical tags, and lemmas. Finally, it sends the controller a set of all of them.
4. Once the controller receives the set of tokens, tags, and lemmas, it sends it to the knowledge module.
5. The knowledge module is in charge of generating SPARQL queries and executing them on the ontology to get a response. Lastly, it sends the response back to the controller.
6. The controller sends a response to the Android application to show it to the user.

For the implementation of each component, we used several development tools. We implemented the Android app with the Android Studio IDE, using native libraries to convert voice queries to text. One can run it in devices that use Android as an operating system.

The controller is JSP technology [13] implementation. Receives and makes requests to other components to manage all of the system's processes, from receiving the query to sending the answer to the Android application.

The NL processor is implemented in the Java language.After reviewing several natural language processing tools like the open-source library Freeling [19], the library Apache OpenNLP[9], and the Stanford CoreNLP [17] tools, we chose Freeling to implement this component since, unlike the others, it has better support for the English and Spanish languages.

The knowledge base is in ontology form, and in OWL [18] and RDF [21] files, to access it, we use the SPARQL query language. We implemented the knowledge module using the Java programming language and Apache Jena, which is a Java framework for querying knowledge bases stored in RDF and OWL structures and allows managing the ontology at runtime.

The NLI allows selecting the previously created ontology it is desired to query. Using the framework Jena, the NLI accesses the ontology and analyzes its structure to reply to the user's natural language queries subsequently.

5 Conclusions

With the upcoming of Semantic Web, the use of ontologies has taken importance, since they intended to use as a knowledge representation medium to facilitate the interoperability of information on the Web. The NLI2O are adequate tools to access knowledge stored in the ontologies. Its development could facilitate user resource localization, communication between informatics applications, information lookup, disambiguation, and other activities on the Web.

In this work, we propose the architecture of an NLI that uses knowledge bases on ontologies, which receives voice queries from an Android device. Then, it converts them to text, applies natural-language and knowledge-representation-processing techniques to generate the corresponding SPARQL queries. Finally, it extracts information from the ontology.

Even though the modeling and semantic processing by the NLI has allowed answering queries formulated in natural language by users, some areas of opportunity were detected.

As future works, we proposed improving the knowledge representation system, as well as the NLI semantic processing and F-logic. We also suggest upgrading the natural language processing, so the NLI is capable of answering queries that make comparisons, negotiations, automatic disambiguation, dates, and numbers. Subsequently, we propose evaluating the performance of the NLI against FREyA using the ontology Geography.owl and its respective corpus of queries called Geoquery 250 and Geoquery 880.

[9] Refer to `https://opennlp.apache.org`.

As future long-term works, we propose addressing complex problems related to natural language features. Some of them are anaphora, and intersentential ellipsis, to mention a few. The anaphora is when we refer to a previously mentioned entity, and its resolution is essential to answer questions that relate to the same object in different forms. The ellipsis occurs when we omit one or more words in a sentence and are understood by having been mentioned earlier.

References

1. Aduna, J.B., Aduna, A.K.: SeRQL: A second generation RDF query language. SWAD-Europe Workshop on Semantic Web Storage and Retrieval (2003)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (May 2001)
3. Bernstein, A., Kaufmann, E., Kaiser, C., Kiefer, C.: Ginseng: A guided input natural language search engine for querying ontologies. Department of Informatics, University of Zurich, Switzerland (2006)
4. Cimiano, P.: ORAKEL: A natural language interface to an F-Logic knowledge base. In: Meziane, F., Métais, E. (eds.) Natural Language Processing and Information Systems. pp. 401–406. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
5. Cimiano, P., Haase, P., Heizmann, J.: Porting natural language interfaces between domains: An experimental user study with the ORAKEL system. International Conference on Intelligent User Interfaces, Proceedings IUI (2008)
6. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: An architecture for development of robust HLT applications. In: 40th Annual Meeting on ACL. pp. 168–175. ACL '02, Association for Computational Linguistics, USA (2002)
7. Damljanovic, D.: Natural Language Interfaces to Conceptual Models. Ph.D. thesis, The University of Sheffield (2011), `http://etheses.whiterose.ac.uk/1630/2/Damljanovic{%}2C{_}Danica.pdf`
8. Damljanovic, D., Agatonovic, M., Cunningham, H.: Identification of the question focus: Combining syntactic analysis and ontology-based lookup through the user interaction. 7th LREC (May 2010)
9. Damljanovic, D., Agatonovic, M., Cunningham, H.: Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) The Semantic Web: Research and Applications. pp. 106–120. Springer, Berlin, Heidelberg (2010)
10. Damljanovic, D., Agatonovic, M., Cunningham, H.: FREyA: An interactive way of querying linked data using natural language. In: García-Castro, R., Fensel, D., Antoniou, G. (eds.) The Semantic Web: ESWC 2011 Workshops. pp. 125–138. Springer, Berlin, Heidelberg (2012)
11. Guarino, N., Oberle, D., Staab, S.: What is an Ontology? In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 0–17 (2009)
12. Hunt, J.: Java Server Pages, pp. 361–370. Springer, London (2002)
13. Jayson Falkner, K.W.J.: Servlets and JavaServer Pages^{TM}: The J2EE^{TM} Technology Web Tier. Addison-Wesley Professional (2003)
14. Kaufmann, E., Bernstein, A.: How useful are natural language interfaces to the semantic web for casual end-users? In: Aberer, K., Choi, K.S., Noy, N., Allemang, D., Lee, K.I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) The Semantic Web. pp. 281–294. Springer, Berlin, Heidelberg (2007)

15. Kifer, M., Lausen, G.: F-Logic: A higher-order language for reasoning about objects, inheritance, and scheme. In: Proceedings of the 1989 ACM SIGMOD. pp. 134–146. '89, ACM, New York, NY, USA (1989)
16. Lopez, V., Pasin, M., Motta, E.: Aqualog: An ontology-portable question answering system for the Semantic Web. In: Gómez-Pérez, A., Euzenat, J. (eds.) The Semantic Web: Research and Applications. pp. 546–562. Springer, Berlin, Heidelberg (2005)
17. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: ACL System Demonstrations. pp. 55–60 (2014)
18. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview. World Wide Web Consortium (February 2004)
19. Padró, L.: Analizadores multilingües en FreeLing. Linguamatica 3(2), 13–20 (December 2011)
20. Prud'hommeaux, E., Seaborne, A., W3C, Laboratories, H.P., Bristol: SPARQL Query Language for RDF. World Wide Web Consortium (January 2008)
21. Schreiber, G., Amsterdam, V.U., Raimond, I., BBC: RDF 1.1 Primer. World Wide Web Consortium (February 2014), `https://www.w3.org/TR/rdf11-primer/`
22. Solís, A., Florencia, R., Acosta, J., López, F.: Interfaz de lenguaje natural para deducción de información almacenada en ontologías . Research in Computing Science 147(6), 189–205 (2019)
23. Tablan, V., Damljanovic, D., Bontcheva, K.: A Natural Language Query Interface to Structured Information. In: The Semantic Web: Research and Applications: 5th ESWC, Tenerife, Canary Islands, Spain, June 1–5. Springer (2008)
24. Tang, L.R., Mooney, R.J.: Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing, pp. 466–477. Machine Learning: ECML 2001, Springer Berlin Heidelberg (2001)
25. Tudorache, T., Nyulas, C., Noy, N.F., Musen, M.A.: Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web. Semantic Web 4(1), 89–99 (2011), `10.3233/sw-2012-0057`
26. Vukotic, A., Goodwill, J.: Introduction to Apache Tomcat 7, pp. 1–15. Apress, Berkeley, CA (2011)