TESIS DE LA UNIVERSIDAD DE ZARAGOZA

2013

134

Carlos Bobed Lisbona

Semantic Keyword-based Search on Heterogeneous Information Systems

Departamento
Informática e Ingeniería de Sistemas

Director/es
Mena Nieto, Eduardo

ISSN 2254-7606

Prensas de la Universidad
Universidad Zaragoza

Universidad Zaragoza
1542

Tesis Doctoral

SEMANTIC KEYWORD-BASED SEARCH ON HETEROGENEOUS INFORMATION SYSTEMS

Autor

Carlos Bobed Lisbona

Director/es

Mena Nieto, Eduardo

UNIVERSIDAD DE ZARAGOZA

Informática e Ingeniería de Sistemas

Semantic Keyword-based Search on Heterogeneous Information Systems

Carlos Bobed Lisbona

Tesis Doctoral

Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Octubre 2013

"There's a sign on the wall but she wants to be sure
'Cause you know sometimes words have two meanings"
Stairway to Heaven - Led Zeppelin

"Sanity can be the toll -
leading to the core of your soul."
Avantasia - Avantasia

"I've got to keep going, be strong
Must be so determined and push myself on"
The Loneliness of the Long Distance Runner - Iron Maiden

Acknowledgements

First of all, I want to thank my advisor, Eduardo Mena, who, apart from being there *anywhere* and *anytime* to help me with my thesis[1], has had the patience needed to get the best out of my efforts.

This work would not have been possible without the priceless help of my colleagues of the SID group which are also my co-authors: Sergio Ilarri, Raquel Trillo, Jordi Bernad, and the latest acquisitions, Fernando Bobillo, Roberto Yus, and Guillermo Esteban. Thank you all for the efforts and all those long nights trying to reach all those unforgiving deadlines. Thanks also to all my laboratory mates (I will not try to name all of them, as it has been quite a long time and I do not want to miss anyone) for all the great moments shared, specially at coffee breaks.

To finish with the academic staff, I also want to thank Francisco Serón, whose phone calls, along with Eduardo's ones, have become one of my strongest fears (you both will fight later for whom scares me the most, calm down ☺).

After this, where to start? I want to thank my friends, Luis Carlos and Estefanía, Raúl and Sheila, Santi and Javi (Mr. Paquito), for all those moments of laughs that have reminded me that there is life beyond the thesis. Thanks also to my handball mates, after quite a few ball hits I have finally got my PhD (it was not a joke, I was really taking a PhD ☺).

Last but not least, I want to thank my parents (Javier and Conchita) for all their support; my brothers (Javier and Jorge), always there for suggesting me new games which waste my time with ☺; and, specially, my girlfriend, Elena, because of all the patience she has had with me, encouraging me to go on, and being there in highest and lowest moments, I love you.

[1]Specially anytime behind the email: at first, I thought he had some kind of bot answering all the emails I sent him.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The work presented in this thesis belongs to the broad context of *Information Systems*, understanding them as systems that help users to fulfill their information requirements. Delving into this context, our work is closely related to *Semantic Search*, *Semantic Web*, *Knowledge Representation*, and even *Ontology Engineering*. In particular, this thesis focuses on exploiting the different mechanisms that *Semantic Web* technologies have provide us with to bridge the gap that there exists between the spread keyword-based interfaces and the formalisms that are used to access information.

Taking into account the semantics of the different elements that take part in the search process, our approach is able to access and integrate data from different heterogeneous underlying information systems using plain keywords as initial input. We focus on this kind of input as the popularity of keyword-based interfaces has grown along with the spread of Web Search engines. Its simplicity allows users to express their information needs easily; however, they introduce the high cost of ambiguity.

Our work aims at reconciliating this ease of use of keyword-based interfaces with the expressive power of structured query languages. This issue has been studied by other research groups, achieving solutions that are attached to the underlying information system by both the query language and the data model used. In this thesis, we generalize the problem and provide a flexible solution that enables us to get rid of these limitations. Our work builds on previous research works of our group on keyword senses disambiguation, a crucial step in our approach as we will see.

In this chapter, we first present the motivation for this thesis. Then, we detail the different semantic levels that our proposal takes into account to achieve its goal. Finally, we present the structure of this thesis.

1.1 Motivation

In the last few years, with the upcoming of the World Wide Web, a huge amount of information has made been available for users in several forms. Moreover, this information is more easily accessible than ever thanks to the increasing levels of connectivity that the users are provided with: On the one hand, users can access to the Internet via broadband Internet connections at their homes; on the other hand, technological advances of mobile devices and wireless networks (e.g., 3G, Wi-Fi) allows them to be connected all the time via their smartphones.

All this information has a sheer potential value... if filtered and accessed properly according to the user's needs. The excess of information can be as harmful as the lack of it, as users might be overwhelmed and not be able to process it. Besides, this information might be behind different sources or sites, thus forcing users to actively search for the appropriate place to search for particular data.

The information systems that effectively provide users with this information are of such a different nature that users would have to be aware of each of their particularities to be able to exploit them. For example, compare the direct use of a Web Search engine with a Linked Data [BHBL09] endpoint: In the case of the former, users can pose their queries directly expressed in terms of keywords; while, in the case of the latter, users need to know both the underlying data schema and the query language itself to be able even to build a query. This heterogeneity becomes greater when we consider any possible information service as source of potential relevant information for the user (e.g., location-dependent services, databases exported via Web Services, etc.).

As the amount of information and the heterogeneity of the systems holding it is so high, the integration of such information systems should be performed externally, providing methods to access them transparently as it occurs, for example, in federated database systems [HM85,SL90]. The trend to integrate all the possible services can be easily seen in the Google search main page: The user's search can be posed almost transparently to different search services (e.g., search for Web pages, images, maps, etc). The actual search service used gives a global meaning to the kind of searches that are performed, but at least they are all integrated in one site. Regarding the integration of information systems, the use of ontologies and mappings to the underlying systems's data schemas has been of great help [MI01]. They can be used to provide a global view of the integrated systems, and to store information needed to actually access them.

Apart from integrating these systems, we have to provide users with an easy method to express their information needs. This method needs to be independent of any particular system (independent of its data model, its query language, and its query processing model) in order to be adaptable enough to integrate them under its unifying view. Regarding this issue, keyword-based interfaces have spread thanks to its adoption by the most popular Web Search engines. They provide a simple way to express queries, but it comes at the cost of lack of expressivity, and ambiguity.

So far, several approaches have addressed the problem of performing keyword-search on different information systems. However, they are usually bounded to its data model and query languages, which makes them non-flexible solutions. Moreover, they do not provide a single entry point to the different information services that might be relevant for the user's needs.

Therefore, in this thesis, we propose an approach to integrate all these heterogeneous information systems under the unifying view of a semantic keyword-based search that:

1. Provides users with an easy way to express their queries that they are used to, i.e., keyword-based search;

2. Avoids the ambiguity of plain keyword-based search by discovering the exact meaning that users have in mind when posing their queries;

3. Adapts itself to the answering capabilities of each of the underlying systems.

Our system encapsulates the different systems attached to it, adapting their query languages, data models, and query execution models, and providing users with a simple way of expressing their searches.

1.2 Overview of the Approach

As we have introduced in the previous section, our approach aims at providing the benefits of keyword search, while avoiding ambiguity and integrating different heterogeneous information systems. To do so, we advocate for a semantic keyword-based search, where the semantics of the input are well established firstly, to then access only the semantically relevant data.

For this task, several approaches (e.g., [RS06, TGEM07]) advocate starting with the discovery of the meaning of each keyword among the different possible combinations. These approaches consult a pool of ontologies (which offer a formal, explicit specification of a shared conceptualiza-

tion [Gru93, Gru95]) and use disambiguation techniques to discover the intended meaning of each user keyword. So, plain keywords can be mapped to ontological terms (concepts, roles, or instances).

In this thesis, we delve into that line and present a system that performs semantic keyword-based search on different data repositories. Our system:

1. Discovers the meaning of the input keywords by consulting a generic pool of ontologies and disambiguates them taking into account their context (the rest of the keywords in the input set); i.e., each keyword in the input has influence on the rest of the keyword's meanings.

2. Then, as a given set of user keywords (even when their semantics have been properly established) could represent several queries, the system finds all the possible queries using the input keywords in order to precisely express the exact meaning intended by the user. This is done considering different formal query languages (the use of formal languages avoids ambiguities and expresses the user information in a precise way), and avoiding inconsistent and semantically equivalent queries with the help of a Description Logics (DL) reasoner [BCM+03]. During this process, our system considers the addition of *virtual terms*. These virtual terms represent missing keywords that users had in their mind but did not input[1]. This way, our system can explore further meanings when the user has given an incomplete input.

3. Finally, once the user has validated the generated query that best fits her/his intended meaning, our system routes the query to the appropriate structured data repositories that will retrieve data according to the semantics of such a query.

The architecture of our system is flexible enough to deal with different ontologies, formal query languages, and query processing capabilities of underlying data repositories. In the following section, we discuss the different semantic aspects that are taken into account in our approach.

1.3 Semantic Levels of Our Approach

The main feature of our approach is that it is completely semantics guided. In Figure 1.1, we can see how the three main steps align with the different semantic levels that are taken into account during the whole process.

[1]For example, a user looking for movies whose genre is "horror" could enter "horror movie", omitting the keyword "genre".

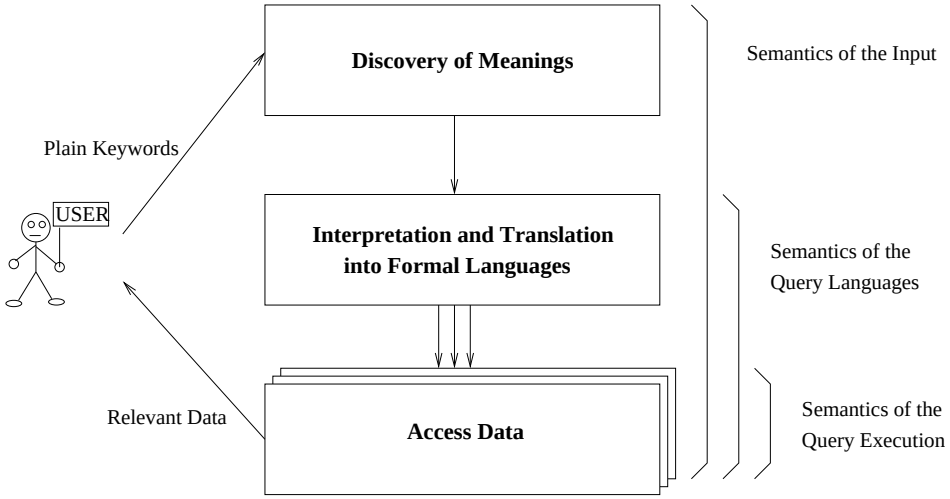


Figure 1.1: Main steps of our approach and the semantic levels involved.

Semantics of the Input

The semantics of the input set of keywords are treated at two different levels:

- First, our system discovers and disambiguates the meaning of each of the keywords that conforms the input. For instance, the keyword "book" could mean "a kind of publication" or "to reserve a hotel room". To determine the meaning of each input keyword, it takes into account its context (the rest of the keywords in the input set); i.e., each input keyword has influence on the rest of the keyword's meanings.

- Second, once each keyword has an exact meaning, our system considers the semantics of the whole set of keywords to build possible interpretations according to the different query languages of the integrated systems. The use of ontologies and formal languages to express the information needs enables our system to get rid of the ambiguity of the input composed by plain keywords.

Semantics of the Query Languages

In our approach, the syntax of the different query languages that are used is left aside to focus on the semantics of their different operators. On the one hand, via the use of extended grammars, we make it possible to specify

query languages taking into account their operators: The arguments that they take as input, their returned values, their properties, etc. On the other hand, we extend further this semantic specification by using Description Logics [BCM+03] to express different semantic conditions that both the operators and their operands must satisfy to build not only syntactically valid queries, but also semantically.

Semantics of the Query Execution

Last but not least, our system considers the different semantics that are behind the different execution models offered by the systems. Different user information needs might require different types of queries and execution schemas, each with its own semantics. For example, information needs about static knowledge require a snapshot query execution (e.g., a user looking for the list of taxi companies in a city); while information needs about volatile knowledge require a continuous query processing as the answer might be continually changing (e.g., a user looking for a cab nearby in a rainy day requires considering continually their position -user and cabs positions- as the answer gets obsolete quickly).

These three semantic levels (input, query languages, and query execution ones) are integrated in our approach to develop a highly flexible system that enables users to perform keyword-based searches over heterogeneous information systems. Our approach is capable of adapting itself to the query capabilities of the underlying systems, providing users with a single entry point to all of them, while retrieving the appropriate answer for their information needs.

1.4 Structure of the Thesis

This thesis is composed of eight chapters, including this one. In Chapter 2, we present the technological context of this thesis, focusing on what ontologies are and what benefits their use provides us with; and on the differences among the informal and formal query languages relevant to this thesis. We finish Chapter 2 with an analysis of the main works related to this thesis.

In Chapter 3, we present a lightweight approach to perform semantics-oriented keyword search on Linked Data repositories. Then, we introduce the problems of translating queries expressed using the keyword query model into other more expressive query models, and present QueryGen, our approach to achieve a generalized keyword interpretation where the semantics of all the elements involved in the process are taken into account.

In Chapter 4, we explain how QueryGen obtains the meaning of each of the input keywords taking into account their query context. In this step, the information retrieved during the disambiguation process is integrated into a multi-ontology sourced ontology, which contains the user's intended meanings. We explain also how we have used external sources and modularization techniques to enrich the available information.

In Chapter 5, we focus on the query generation process that QueryGen performs to translate the keyword queries into more expressive query languages. We detail how we can specify different query languages using specially annotated grammars, and how QueryGen uses these language specifications to generate the queries guided by the semantics of the input keywords and the operators. We also present how QueryGen uses the integrated knowledge to filter out inconsistent queries, and to react to insufficient inputs by performing a *semantic enrichment* of the input. Last but not least, we present the semantic techniques that QueryGen uses to reduce the search space that a generalized keyword interpretation implies.

In Chapter 6, we explain the solution adopted to make QueryGen capable of handling different underlying data models. It is an architecture based on wrappers, which can be attached on the fly. These wrappers provide QueryGen with information about their underlying information systems (the query languages used, data formats, and execution models). We present and detail two successful use cases already implemented (DBpedia [BLK+09] and LOQOMOTION [IMI06]). As a result of working in the field of location-dependent queries, we also propose two different semantic models for representing location granularities that extend the semantics of the queries that can be handled with LOQOMOTION.

In Chapter 7, we test QueryGen from a qualitative and a quantitative point of view. In particular, using a third-party query set, we evaluate the semantic capabilities of our approach regarding the discovery of the user's intended meaning. Then, we focus on the performance of the system, paying special attention to execution times and the impact of the query reduction techniques used by QueryGen.

Finally, in Chapter 8, we present the conclusions as well as our main contributions; we finish with the future research topics.

Chapter 2

Technological Context

In this chapter, we present the technological background needed to understand this thesis. Firstly, we give an overview of what *ontologies* are, their underlying formalism, and their representation languages. Then, we present a brief summary of query languages and give some overview of the ones that are specially relevant to our system. Finally, we provide an overview of the different systems that are related directly to the work presented in this thesis.

2.1 Knowledge Representation

In this section, we present the main semantic tools used along the whole thesis and their relationships. In particular, we use ontologies to handle the semantics of the underlying data models, choosing the Description Logics [BCM+03] as underlying formalism. Description Logics languages come along with reasoners, which allows, among other tasks, to check the consistency of the model. Finally, we present three representation languages that are closely related to each other: RDF [MM04], RDF-S [BG04], and OWL [HKP+12]. The three of them are W3C recommendations for different tasks in the context of the Semantic Web [BLHL01, SHBL06].

2.1.1 Ontologies

Ontologies have their roots in Philosophy, where *ontology* means a systematic explanation of being. In the context of Computer Science, and, in particular, in Artificial Intelligence, they are used as models of the reality,

providing the computers with descriptions that they are capable of understanding. Gruber defined them as follows [Gru93, Gru95]:

> *An ontology is an explicit specification of a conceptualization.*

A definition which comprises two of their main characteristics:

- They are conceptualizations, abstract models formed out of a reality we want to model.
- They are an explicit specification, where all the elements of the ontology must be clearly defined.

Then, Borst [Bor97] extended this definition to add two another important characteristics:

> *Ontologies are defined as a* formal *specification of a* shared *conceptualization.*

Both definitions were merged by Studer et al. [SBF98] in the following definition:

> *An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.*

There are several approaches to categorize them according to different dimensions: The objective of the conceptualization [MVI95], the amount and type of structure of the conceptualization and the subject of the conceptualization [vHSW97], or their level of dependence on a particular task [Gua98]. Nevertheless, from a practical point of view, we focus on the categorization given in [LM01], where ontologies are classified in lightweight and heavyweight ones according to the information needs to express and the richness of its internal structure (see Figure 2.1). Thus, we would have:

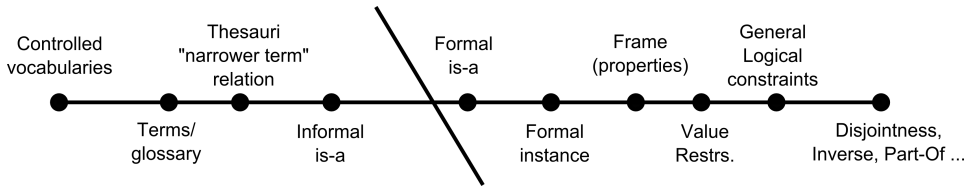


Figure 2.1: Ontology categorization: lightweight and heavyweight ones (taken from [LM01]).

- Lightweight ontologies
  - Controlled vocabularies: They provide a finite list of terms, each of one has an unambiguous interpretation (e.g., catalogues).
  - Glossaries: They provide a list of terms that come along with their meanings specified as natural language statements (e.g., dictionaries).
  - Thesauri: They provide a list of terms with additional semantics in their relationships among each other. Typically, they provide information about *synonymy* (equivalence) and hierarchical relationships between terms, such as *hypernymy* ("broader than") and *hyponymy* ("narrower than").
  - Informal is-a hierarchies: They organize the terms using a general notion of *generalization* and *specialization*. They are not a strict subclass hierarchy, but provide a way to organize concepts that are strongly related together. For example, the general category "apparel" could include a subcategory "women" (which should more accurately be titled women's apparel) which then includes subcategories accessories and dresses.
- Heavyweight ontologies
  - Formal is-a hierarchies: They organize the terms adopting the formal notion of generalization/specification. In this hierarchies, if B is a subclass of A and we have an instance of B, then it is also an instance of A. This interpretation is needed to exploit inheritance in the model. These ontologies only include class names.
  - Formal is-a hierarchies with individuals: They are formal is-a hierarchies that include individuals of the domain, via formal instance relationships.

  - Frames (properties): They include classes (frames or concepts) and their properties (slots or attributes), which can be inherited by classes of the lower levels of the formal is-a taxonomy.
  - Ontologies that express value restrictions: They are ontologies that allow to place restrictions on the values that can fill a property, forcing the instances to belong to a determined class.
  - Ontologies that express logical restrictions: The most expressive ones. They make it possible to specify first-order logic constraints between terms using expressive ontology languages. Moreover, further detailed information about concepts (e.g., disjointness) and properties (e.g., inverse ones, symmetry, transitivity, etc.) can be expressed.

For further details on this topic, we refer to [GPFLC04]. There, the interested reader can find (among other interesting readings) an in-depth discussion on the different definitions and categorizations of ontologies.

From a modeling point of view, and focusing on rich ontologies (i.e., heavyweight ones according to Figure 2.1), we can distinguish three main modeling elements in every ontology independently of the underlying formalism:

- Concepts: They represent classes of objects within the modeled domain. They are usually organized in taxonomies, via subsumption relationship.
- Roles: They represent existing relationships between concepts in the domain, making it possible to describe properties of the concepts.
- Instances: They represent specific individuals of the concepts in the ontology.

The relationship between these elements, the formalism adopted to represent ontologies, and the implementation languages can be seen as a three layer composition. The ontological elements are formally represented within the knowledge representation paradigm selected to represent the ontology. Examples of such formalisms are Frame Logic [KLW95] or Description Logics [BCM$^+$03]; each of which have different computational properties and modeling expressiveness. While Gruber in [Gru93] firstly proposed frames along with first order logic to model ontologies, Description Logics languages have gained popularity due to their adoption by the community for

the Semantic Web (they are the formal base for the Web Ontology Language - OWL [HKP+12]). Finally, each of these underlying formalisms is implemented using different languages which might cover different aspects of them. In Figure 2.2, the formalism and implementation languages adopted in this thesis are presented.

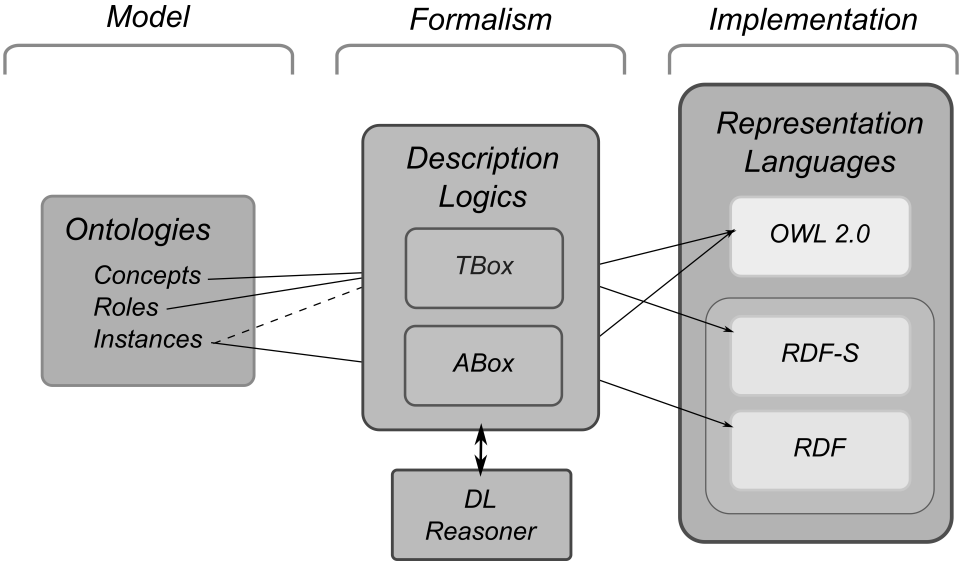


Figure 2.2: Formalism and implementation languages in this thesis.

In the following subsections, we focus on them, giving an overview of both the selected formalism and the representation languages.

2.1.2 Description Logics

Description Logics languages (DLs from now on) are "*formal languages for representing knowledge and reasoning about it*" [BCM+03]. In this section, we briefly overview their definitions[1].

DLs are formed by an intensional layer $\mathcal{T}$ called TBox and an extensional layer $\mathcal{A}$ called ABox. The TBox is composed by a set of terminological axioms. Axioms are formulas of the form $C \equiv D$ or $C \sqsubseteq D$, where C and D are concepts. Concepts are formed by means of: 1) a set of concepts names N_C, conceptualizations of a set of individuals (or instances), for example, $Person$, Car, and Dog; 2) a set of roles N_R, which are binary relations between individuals, for instance, $hasPet$, and $hasChildren$; and 3) construc-

[1]We refer the interested reader to [BCM+03].

tors to define new concepts, such as $\cap$, $\cup$, $\exists$, $\forall$. For example, given that we have as concepts $Person$ and Dog, and $hasPet$ as a role, we can define a new concept to represent people who have a pet dog as $Person \cap \exists hasPet.Dog$. An axiom of the form $C \equiv D$ says that concepts C and D are equivalent, that is, any individual that belongs to C also belongs to D, and vice versa. An axiom $C \equiv D$ is called a concept definition if the left hand of the axiom is a concept name. Axioms of type $C \sqsubseteq D$ represent that concept C is subsumed by D, i.e., any individual in C is in D, but not necessarily vice versa. A *general TBox* is a finite set of axioms. An example of TBox expressing that men are human and fathers are men who have children is:

$$\mathcal{T} = \{Man \sqsubseteq Human; Father \equiv Man \cap \exists hasChild.Human\}$$

An ABox is a set of assertions that describe a specific state of the world represented by the associated TBox. We can express with assertions that John is Mary's father:

$$\mathcal{A} = \{Man(John); Human(Mary); hasChild(John, Mary)\}$$

$John$ and $Mary$ will be constants representing individuals. Let us note that we have not asserted that John is a father, as this is implicitly deduced from the TBox and the ABox. The knowledge representation given by TBox $\mathcal{T}$ and ABox $\mathcal{A}$ is denoted by $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$.

DLs can be classified according to their expressivity, i.e., depending on how many different symbols can be used to express axioms and how those symbols can be combined. For example, a DL with constructors $\sqcap, \sqcup, \forall, \exists, \neg, \top, \bot$ is an $\mathcal{ALC}$ DL, and if the set of constructors is enlarged with $\leq n$, $\geq n$ (unqualified number restrictions), we obtain an $\mathcal{ALCN}$ DL. An interpretation $\mathcal{I}$ is a set $\Delta^{\mathcal{I}}$ and a function that associates each concept name C to a subset $C^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, and each role R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ (see Table 2.1).

DLs are logics and provide a formal framework where rules of inference can be applied to deduce automatically new knowledge from TBoxes and ABoxes. This is done using *reasoners*, which are programs that can perform several different reasoning tasks over ontologies. The most typical reasoning tasks reasoners perform include:

- *Consistency check*: Checks if there exists a logical model satisfying all the axioms in the ontology.
- *Instance retrieval*: Get all the instances of a concept.

$C, D \rightarrow$	$A \in N_C$	| A is a concept name
	$\top$	$\mid \top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
	$\bot$	$\mid \bot^{\mathcal{I}} = \emptyset$
	$\neg C$	$\mid (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
	$C \sqcap D$	$\mid (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	$C \sqcup D$	$\mid (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	$\forall R.C$	$\mid (\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall b, (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
	$\exists R.C$	$\mid (\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b \in C^{\mathcal{I}}, (a,b) \in R^{\mathcal{I}}\}$

Table 2.1: Syntax and interpretation for $\mathcal{ALC}$ DLs.

- *Concept satisfiability*: Checks if a concept can have instances i.e., if it does not necessarily denotes the empty set.

- *Entailment*: Checks if a given fact is a logical consequence which can be derived from the axioms in the ontology.

- *Subsumption*: Checks if a concept/property C can be considered more general than (or subsumes) a concept D.

- *Classification*: Computes a concept/property hierarchy based on the relations of concept/property subsumption.

In this thesis we have used two different reasoners, Pellet[2] [SPG+07] and HermiT[3] [MSH07, MSH09], both of which support OWL 2, the representation language proposed by W3C for ontology specification, which is overviewed in the following subsection.

2.1.3 Representation Languages

As we have seen in Section 2.1.1, and, in particular, in Figure 2.2, once we have selected what to model (the ontology) and the underlying formalism (DLs) to express it, we have to implement it in a representation or implementation language. In this section, we present the most important languages adopted and used in the context of the Semantic Web: RDF (along with RDF-S) and OWL.

[2] `http://clarkparsia.com/pellet`, last accessed October 3, 2013.
[3] `www.hermit-reasoner.com`, last accessed October 3, 2013.

2.1.3.1 RDF and RDF-S

RDF (Resource Description Framework) [MM04] is a language for representing information about resources on the World Wide Web. At first, it was intended for representing metadata (title, date of creation, authorship, etc.) about Web documents; however, by generalizing the notion of resource, it can be used to represent information about *anything* that can be identified in the Web by a URI (Uniform Resource Identifier).

In RDF, the most basic representation data unit is the triplet, $< a\ R\ b >$, which represents an statement where a is the subject, b the object, and R is the property that links them. This simple data model provides flexibility to represent the information about the resources as a graph of nodes and arcs which represent the resources, their properties, and their values.

In Figure 2.3, we can see an example of an RDF graph comprising information about myself. We have a URI representing myself as subject of a set of different statements. Summing up, the nodes of the RDF graph can be: Other resources identified by their own URIs (e.g., my PhD. advisor); blank nodes, which makes it possible to have composite values (e.g., an address) without having to give them an identifier; or literals, typed or not (e.g., my age and my hobby).

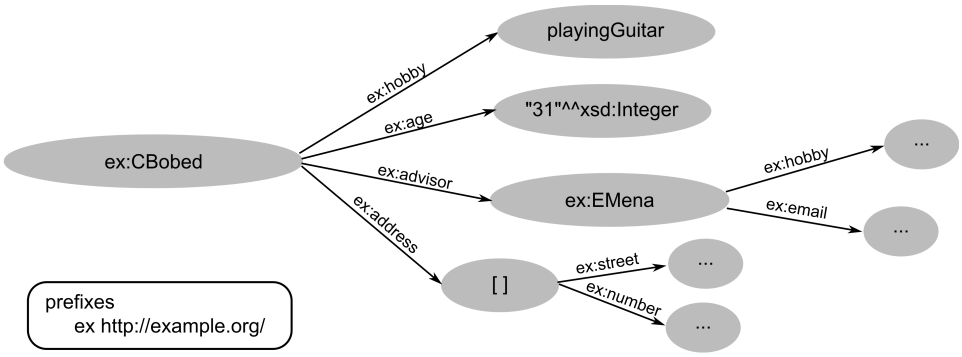


Figure 2.3: RDF example: information about myself.

The structure of the information enables it to be automatically shared and processed by different programs; however, they have to *speak* the same language, i.e., use the same vocabulary. RDF-S (RDF Schema) [BG04] was developed to ease the definition and sharing of these vocabularies that enable interoperability. It allows to define classes, along with their properties (see Table 2.2). Despite of the fact that its expressiveness is quite low, it provides a formal implementation language for simple ontologies.

Classes	*Properties*	*Utility Properties*
rdfs:Resource	rdfs:domain	rdfs:seeAlso
rdfs:Class	rdfs:range	rdfs:isDefinedBy
rdfs:Literal	rdf:type	
rdfs:Datatype	rdfs:subClassOf	
rdf:XMLLiteral	rdfs:subPropertyOf	
rdf:Property	rdfs:label	
	rdfs:comment	

Table 2.2: RDF-S elements.

In the following subsection, we present OWL, which builds on RDF-S, extending further its expressiveness.

2.1.4 OWL: Web Ontology Language

OWL (Web Ontology Language) [HKP+12] is the current W3C recommendation for ontology specification. OWL is based on the DL $\mathcal{SROIQ}(\mathbf{D})$. In this subsection we will only describe the syntax of the language, but more details about the underlying logic, including the semantics, can be found in [HKS06]. OWL provides several syntaxes, among which, we will use Manchester syntax [HP12], specifically designed to be easily understood by humans.

OWL ontologies have five elements: Individuals, concepts (or classes), datatypes (or concrete domains), object properties, and data properties. Essentially, concepts are sets of individuals, datatypes are sets of values defined over a concrete domain (such as integers or dates), object properties are binary relations between individuals, and datatype properties relate individuals and datatypes.

Table 2.3 shows the supported concept constructors in OWL 2, its latest version. Using these constructors, we can build complex concepts from simpler ones inductively. On the other hand, Table 2.4 summarizes the main axioms in OWL 2. The top part of the table (with 7 axioms) contains the axioms concerning the ABox, and the lower part contains the axioms concerning the TBox. Some of the axioms are just syntactic sugar and can be represented using equivalent class axioms (such as disjoint classes, disjoint union of classes, and domain and range axioms). In these tables, C is a concept, R is an object property, n is a natural number, i is an individual, T is a datatype property, v is a datatype value, and D is a datatype.

In order to guarantee the decidability of the logic, there are some restrictions in the role hierarchies axioms and some roles are required to be simple ones. The interested reader may find the formal specification at [HKS06].

A	Atomic/primitive concept
C_1 or C_2	Disjunction
C_1 and C_2	Conjunction
not C	Negation
$Thing$	Universal concept
$Nothing$	Empty concept
$\{i_1, \ldots, i_n\}$	Nominals/Enumeration
R only C	Universal restriction
R some C	Existential restriction
R value i	Value restriction
R self	Self concept
R exactly n $[C]$	[Qualified] Exact cardinality restriction
R max n $[C]$	[Qualified] Maximal cardinality restriction
R min n $[C]$	[Qualified] Minimal cardinality restriction
T only D	Universal restriction
T some D	Existential restriction
T value v	Value restriction
T exactly n $[D]$	[Qualified] Exact cardinality restriction
T max n $[D]$	[Qualified] Maximal cardinality restriction
T min n $[D]$	[Qualified] Minimal cardinality restriction

Table 2.3: OWL 2 concepts.

Figures 2.4 and 2.5 show an example of an OWL ontology (*proyectos.owl*[4]) visualized with our ontology viewer OntView[5]. In this example, we can visualize the following definitions (among others):

```
jefes EquivalentTo
    personas and (ocupacion value "jefe")

superPro EquivalentTo
    proyectos and (miembros min 3 personas)
```

The aim of our viewer is to capture visually the exact meaning of the loaded ontology. To do so, it uses a DL reasoner to classify the ontology and to obtain all the relevant information about it. Depending on the ontology size and amount of visual information that the user wants to be displayed, we can visualize the ontology in two different modes:

- Not expanded (Figure 2.4), where the definitions and expressions are presented in a compact way.

[4] `http://sid.cps.unizar.es/ontology/proyectos.owl`, last accessed October 3, 2013.

[5] `http://sid.cps.unizar.es/OntView/`, last accessed October 3, 2013.

i Types C	Concept assertion
i_1 Facts R i_2	Property assertion
i_1 Facts not R i_2	Negated property assertion
i Facts T v	Property assertion
i Facts not T v	Negated property assertion
SameIndividual i_1 i_2	Equality assertion
DifferentIndividuals i_1 i_2	Inequality assertion
C_1 SubClassOf C_2	Subclass axiom
C_1 EquivalentTo C_2	Equivalent classes
C_1 DisjointWith C_2	Disjoint classes
C_1 DisjointUnionOf $C_2 \dots C_n$	Disjoint union of classes
$[R_1\|T_1]$ SubPropertyOf $[R_2\|T_2]$	Subproperty axiom
R_0 SubPropertyChain $R_1 \dots R_n$	Subproperty chain axiom
$[R_1\|T_1]$ EquivalentTo $[R_2\|T_2]$	Equivalent properties
$[R_1\|T_1]$ DisjointWith $[R_2\|T_2]$	Disjoint object | data properties
$[R\|T]$ Domain C	Domain of an object | data property
R Range C	Range of an object property
T Range D	Range of a data property
R_1 InverseOf R_2	Inverse properties
$[R\|T]$ Functional	Functional object | data property
R InverseFunctional	Inverse functional property
R Transitive	Transitive property
R Reflexive	Reflexive property
R Irreflexive	Irreflexive property
R Symmetric	Symmetric property
R Asymmetric	Asymmetric property

Table 2.4: OWL 2 axioms.

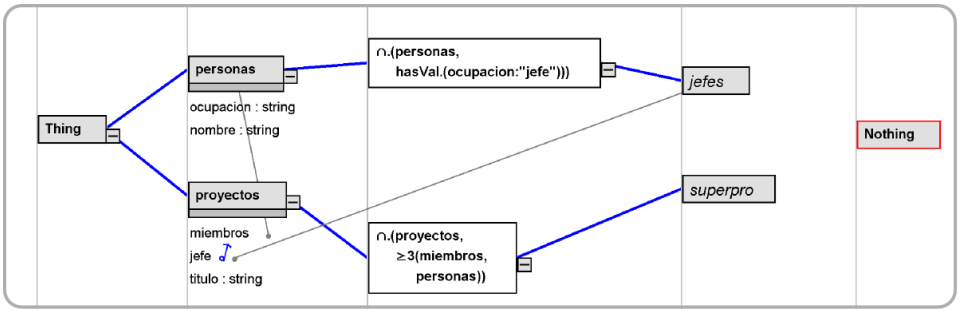


Figure 2.4: OWL example: simple ontology about project management.

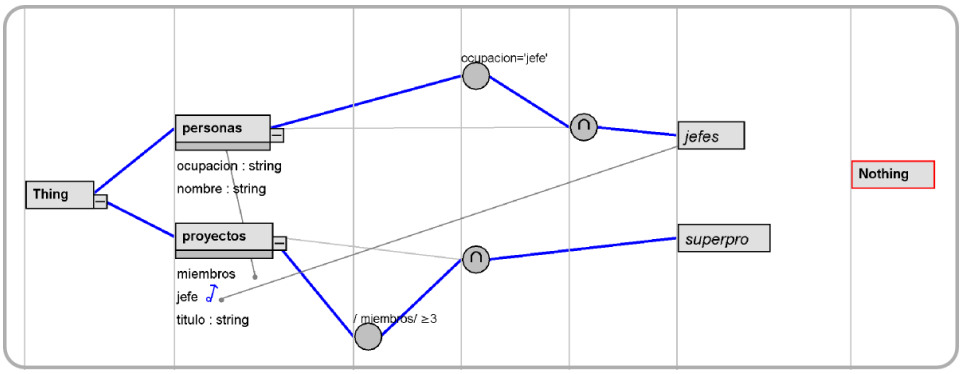


Figure 2.5: OWL example: expanded visualization.

- Expanded (Figure 2.5), where each of the expressions in the ontology are expanded to show their semantics visually.

In the following section, we turn our focus to the notion of query language and present the most relevant ones in the context of this thesis.

2.2 Query Languages

In Computer Science, query languages are computer languages that are used to query databases and information systems. Note the difference that exists between information need and query [MRS08]:

> *An* information need *is the topic about which the user desires to know more, and is differentiated from a* query*, which is what the user conveys to the computer in an attempt to communicate the information need.*

Depending on their formality degree, we can classify query languages as informal or formal ones. Informal query languages are more related to information retrieval tasks, where the semantics of the query are not formally defined. Users express with these query languages their information needs, so these languages imply an intermediate step to establish their semantics (query construction) and adapt the query to the underlying data model. On the other hand, formal query languages have their semantics strictly defined and users express with them queries with a univocal interpretation.

In this section, we present the query languages that appear in this thesis: Informal query languages (natural language and keyword queries), and for-

mal ones (SQL-like languages, SPARQL, and logic based ones –in particular, conjunctive and DL queries).

2.2.1 Informal Query Languages

Under this denomination, we can find our natural languages and the keyword query language, which is a strong simplification of the former.

Natural Language Natural Language (NL) enables users to express their information needs in their own language. Its ease of use makes it always a possible choice for casual users [KB10], but processing it correctly is still an open problem. Among others, NL as query language faces the following problems:

- It is inherently ambiguous: The meaning of the query depends heavily on its context (the *discourse*), and, even when the context has been perfectly established, different aspects such as polysemy or the flexibility of the language might make impossible to interpret correctly the query. Among other linguistic problems, we could find the following challenges (examples taken from [ART95]) when dealing with NL queries:
  - Modifier attachment: When modifiers appear in the sentence, it is not always clear which clause they are modifying. For example, in *List all employees in the company with a driving license*, *"with a driving license"* could modify the company or the employees as well. While one could infer that a company could not have a driving license, this is not straightforward at all. This another example (taken from [PG88]) is even more ambiguous, *List all employees in the division making shoes.*
  - Nominal compound problems: Dealing with English language, nouns are often modified by other nouns and the resulting meaning is quite difficult to be foreseen. For example (based on [PG88], taken from [ART95]), *city department* could mean a department located in a city or a department responsible for the city, *research department* probably means a department carrying out research, and *research system* is probably a system used in research, and not a system carrying out research.
  - Conjunction and disjunction: Users tend to use *and* to denote disjunction instead of conjunction. For example (fitted from [TB83]),

for *List all applicants who live in Zaragoza and Madrid*, all the applicants living either in Zaragoza *or* Madrid should be returned instead of those living in both cities at the same time.

  - Other linguistic features, such as *anaphora* (the use of pronouns and noun phrases to denote entities already mentioned before), *ellipsis* (the use of incomplete sentences, very common in oral communication), quantifier scoping (similar to the modifier attachment problem, but this time concerning logical quantifiers), etc.

- It is language-dependent: The techniques applied to process NL depend directly on the language being processed as they differ at syntactic, grammatical and semantic levels. Assuming that you could process and correctly interpret one language solving the linguistic problems associated to it, moving to another language would require to remake the interpretation process almost from the beginning as the interpretation rules would have change completely.

Natural Language was used firstly as interface to databases [ART95]. This kind of interfaces has evolved into what is currently named *Query Answering* systems, which are not only focused on accessing databases, but on accessing different information systems. In particular, in the last few years, the Semantic Web community has turned its attention to this kind of techniques to be used to query semantic resources [LUSM11]. While they are still far from being perfect, there has been a lot of research in the Natural Language Processing (NLP) field that would help us and vice versa.

Keyword Query Language Keyword queries are a simplification of the queries that can be expressed using Natural Language. They consist of a set of plain keywords that represents the user's information need. For example, a user could express *horror movie* to ask for the latest movies that correspond to the horror genre.

The success and adoption of keyword-based search interfaces have come along with the success of the main Web search engines, such as Google, which adopted it as their main query language. The different search techniques used in the field of Information Retrieval, such as the *bag of words* representation of Web documents, make keyword queries especially easy to answer statistically (using different ranking methods) while keeping the process scalable enough to deal with huge amounts of information. Moreover,

users have found in keyword queries a quick and easy way to express their information needs.

However, the ease of use of keyword search comes from the simplicity of its query model, whose expressivity is low compared with other more complex query models [KB10]. Moreover, keyword queries are in fact projections of the user's actual information need. This leads to a much more ambiguous context, where polysemy and lack of information are always present. Revisiting some of the examples for the NL queries, a user could write the following queries[6]:

- *List all employees in the company with a driving license* could be expressed as *employees driving license*, which although gets rid of the modifier problem, introduces new ambiguity problems: Must the returned employees have a driving license? Must they not? Is there any other kind of license and we should retrieve the employees that are currently driving?

- *List applicants who live in Zaragoza and Madrid* could be expressed as *applicants Zaragoza Madrid*: Which applicants should we retrieve? Those who are living/working in Zaragoza, in Madrid?

However, despite of its inherent ambiguity, keyword-based search interfaces have been adopted by different information systems other than Web search engines as the benefits that they provide in terms of user-friendship and language independence are worthy enough to do so. In this thesis, we aim at overcoming their drawbacks with the help of semantic techniques.

2.2.2 Formal Query Languages

We now turn our attention on formal languages, which make it possible to express the information needed unambiguously thanks to the adoption of different formalisms.

2.2.2.1 SQL-like Languages

The notion of SQL-like language is quite broad. SQL-like languages are languages whose syntax resembles the syntax of SQL (Structured Query Language) [ISO11b], which is the most extended language for querying and managing data stored in relational databases [Cod70].

[6] We assume three keywords for the examples as the average number of keywords used in keyword-based search engines "is somewhere between 2 and 3" [MRS08].

SQL has its formal foundations on relational algebra [Cod71,EN11], and consists of a *data definition language* (to create and manage the database schema) and a *data manipulation language* (to insert, update, and query the data in the database). Focusing on SQL as a query language, and independently of the underlying schema, a SQL query has the following general structure:

```
SELECT ListOfProjections
    FROM ListOfTables
    WHERE ListOfConditions
```

where:

- *ListOfProjections* is the list of attributes that have to be retrieved as an answer for the query.
- *ListOfTables* is the list of the tables that are involved in the query.
- *ListOfConditions* is the list of the conditions that the attributes have to meet to form part of the answer. This conditions can include different kind of relational operators between tables such as the different kinds of JOIN operators that exist.

For example, assuming that the appropriate tables exist, the examples from the previous section would look like the following in SQL:

- The list of the employees of a given company that have a driving license:

```
SELECT EmployeesTb.employeeID
FROM EmployeesTb, CompaniesTb
WHERE CompaniesTb.companyID = exampleCompanyID
    AND EmployeesTb.employeeID = CompaniesTb.employeeID
    AND EmployeesTb.hasDrivingLicense = TRUE
```

- The list of applicants for a position that live in Zaragoza or Madrid:

```
SELECT ApplicantsTb.applicantID
FROM ApplicantsTb
WHERE ApplicantsTb.applyFor = examplePositionID
    AND (ApplicantsTb.livesIn = Zaragoza
        OR
        ApplicantsTb.livesIn = Madrid)
```

As we can see in the examples, the values that compose the answer are unambiguously defined in the query, instead of having to interpret them.

Along all its versions, SQL has been extended with numerous extensions such as object-oriented capabilities with different semantics [BG08, The13], and geospatial capabilities [Sto03, ISO11a]. In this thesis, we use the SQL-like query language used by LOQOMOTION [IMI06] to show how our system can handle this kind of languages (see Section 6.3.4).

In the following subsection, we take a look into SPARQL, which, although it can be considered an SQL-like language, is relevant enough to have its own section in this thesis.

2.2.2.2 SPARQL

SPARQL query language is the W3C recommendation for querying RDF data [HSP13]. It resembles SQL in its syntax, although their queries are expressed in terms of *pattern matching* over the RDF graphs, instead of dealing with relational tables. There are four different types of SPARQL queries:

- SELECT, which retrieves directly the answer to a *query pattern*. This answer is conformed by the used variables and their *bindings* to the answer values.
- CONSTRUCT, which constructs a RDF graph specified by a template (also in terms of *query patterns*).
- ASK, which checks whether a *query pattern* has a feasible solution.
- DESCRIBE, which, given a resource (URI), returns an RDF graph containing information about it.

In particular, we focus on SELECT queries as they allow us to ask directly for facts and data. They have the following structure:

```
'SELECT' ( 'DISTINCT' | 'REDUCED' )? (Var+ | '*')
    DataSetClause*
    WhereClause
    SolutionModifier
```

where:

- *Var* is the list of free variables that conform the answer.

- *DataSetClause* is equivalent to the FROM clause in SQL. It establishes the graph against which the query is posed. In RDF repositories, we can have the data partitioned into different named graphs (as if they were different databases). If omitted, the default graph is used.
- *WhereClause* is the condition that the data retrieved as answer must meet. It is given in the form of a *query pattern*, where the free variables declared in the *Var* list are used to form a graph along with the properties that should relate them. The answer is conformed by all the possible data value tuples that fulfil the pattern.
- *SolutionModifier* is composed by different functions that can be applied to the returned tuples (e.g., ORDER BY, LIMIT, etc.).

The examples in the previous sections would be expressed in SPARQL in the following way:

- The list of the employees of a given company that have a driving license:

```
PREFIX ex:  <http://sid.cps.unizar.es/Example/>

SELECT ?employeeID
WHERE {?employeeID ex:worksFor ex:exampleCompany.
        ?employeeID ex:hasDrivingLicence ?driver.
        FILTER (?driver = 'TRUE') }
```

- The list of applicants for a position that live in Zaragoza or Madrid:

```
PREFIX ex:  <http://sid.cps.unizar.es/Example/>

SELECT ?applicantID
WHERE {?applicantID ex:appliesFor ex:examplePosition.
        {?applicantID ex:livesIn ex:Zaragoza.}
            UNION
        {?applicantID ex:livesIn ex:Madrid. }
        }
```

As we can see in the examples, SPARQL queries resemble SQL queries where the schema has been blurred to the limit of representing everything via RDF tuples. This, along with the use of predefined schemas, provides us with higher levels of flexibility to achieve schema and data interoperability, among other benefits.

2.2.2.3 Logical Languages

In this section, we present two different query languages that adopt different subsets of Logic as their founding formalism: Conjunctive queries and DL queries.

Conjunctive Queries Conjunctive queries are a subset of first order logic queries that have been thoroughly studied in the context of databases. A large part of relational algebra expressions can be expressed as conjunctive queries and vice versa [CM77] (in particular they are included into the select-project-join queries in relational algebra). They are defined as follows:

Definition 2.2.1 *A* conjunctive query *is an expression of the form*

$$(x_1, \ldots, x_k).\exists x_{k+1}, \ldots, x_m.A_1 \wedge \ldots \wedge A_r$$

where $x_1, \ldots, x_k$ *are called* distinguished *(free)* variables*,* $x_{k+1}, \ldots, x_m$ *are called* undistinguished *(bound)* variables*, and* $A_1, \ldots, A_r$ *are* atomic formulae*.*

If the set of distinguished variables is empty, then the query is a *boolean conjunctive query*. Conjunctive queries are often extended with the union operator to build more complex queries. Besides, a formula comprising only $\wedge$ and $\vee$ operators can be rewritten as union of conjunctive queries after converting it into disjunctive normal form [CM77].

Using conjunctive queries, the examples from previous sections would be expressed as follows:

- The list of the employees of a given company that have a driving license:

$$\{x\}.worksFor(x, exampleCompany) \wedge hasDrivingLicense(x, true)$$

- The list of applicants for a position that live in Zaragoza or Madrid:

$$\{x\}.(appliesFor(x, examplePosition) \wedge livesIn(x, Zaragoza)) \\ \vee (appliesFor(x, examplePosition) \wedge livesIn(x, Madrid))$$

This kind of queries covers a frequently used feature of SPARQL, i.e., they are its basic graph pattern. So, several of the works related to this thesis which adopt RDF as their underlying data model focuses on the use of conjunctive queries to interpret the user's input as we will see in the next section.

Description Logics (DL) Queries DL queries [BCM+03] are concept expressions that are used to interrogate a DL knowledge base, which is done usually via a DL reasoner (see Section 2.1.2). In particular, we will use the DL queries to perform instance retrieval, although they can be used also to query the T-Box of the knowledge base to extract information about the concept hierarchy.

They are built with the concept construction operators allowed by the DL language used (see Table 2.3), which defines their expressivity. Being DL concept expressions as they are, their satisfiability according to a given ontology can be assessed directly with the help of a DL reasoner, property which is exploited in this thesis to assess the satisfiability of the possible user's queries.

Using Manchester syntax [HP12], the queries of previous sections would be as follows:

- The list of the employees of a given company that have a driving license:

$$\begin{aligned} &Employee\ and\ worksFor\ value\ exampleCompany \\ &\qquad and\ hasDrivingLicence\ value\ true \end{aligned}$$

- The list of applicants for a position that live in Zaragoza or Madrid:

$$\begin{aligned} &Applicant\ and\ appliesFor\ value\ examplePosition \\ &\qquad and\ livesIn\ some\ \{Zaragoza, Madrid\} \end{aligned}$$

Regarding their relationship with conjunctive queries, they have different expressivity, but they complement each other:

- DL queries could extend their expressivity by using the free variables mechanism to retrieve projected values and other information along with the instances that belong to the DL expression.

- Conjunctive queries could use DL queries as their atomic formulae to obtain all the expressivity of the underlying DL language instead of having to deal with atomic predicates.

In the following section, we move into a presentation of the works that are the most related ones to this thesis.

2.3 Systems Related to QueryGen

In this section, we present the most relevant works to QueryGen. Due to the nature of QueryGen, almost any information system that accepts keywords as input would be related to it. From a general point of view, any system performing a search follows three main steps: Query construction, data retrieval, and presentation of results. Out of these three steps, the first one is crucial because the more accurate the system is able to capture the user's information need, the more precise results it will retrieve. However, the importance of this first step is usually underestimated by adopting unstructured query models (i.e., keyword bag), making the quick access to huge amounts of data and the ranking of results the most important steps of the whole process.

To keep it manageable, we focus on systems that attempt at capturing and interpreting the actual information requirement that is behind the user's input before accessing the underlying data. We have grouped them in three groups according to the approach adopted and their target data model: Keyword interpretation on semantic data, keyword search on relational databases, and question answering. The first group contains the works that are the most related ones to the approach presented in this thesis, and, therefore, they are analyzed thoroughly. The analysis of the other two groups is performed in a less detailed way, as they are not so related to our work as the first group ones.

2.3.1 Keyword Interpretation on Semantic Data

In this section, we group systems that use semantic data to perform the keyword interpretation. They are presented according to their grade of relatedness to our work, being QUICK [ZZM+09] the most related approach to ours. Most of them [GMM03, ZWX+07, WZL+08, THL11] exploit directly the RDF graph structure to perform different subtasks of the search (such as interpreting the possible keyword query or augmenting the retrieved results with relevant ones), which is in fact a data-driven solution. In [FA11] they propose a similar approach to keyword interpretation but they introduce the context of the user's search (the knowledge about previous queries) to focus the whole search process. SemSearch [LUM06] and QUICK [ZZM+09] adopt a different approach by using query templates to perform the keyword interpretation. The former uses a set of predefined query templates, while the latter derives them from a lightweight ontology provided as domain definition.

However, as we will see in the following, the expressivity of these approaches is quite restricted (subsets of conjunctive queries, predefined sets of queries, etc.). Moreover, although the authors in [GMM03] advocate for establishing firstly the meaning of the input keywords, the techniques that these approaches use basically rely on text indexes (some of them slightly semantically enhanced). Moreover, some of the approaches are constrained to the use of a single ontology [ZWX+07, ZZM+09], or use an off-line built one [WZL+08, THL11]. As we will see along this thesis, our system achieves greater levels of flexibility by adopting a multi-ontology solution for the discovery of the keywords meaning and a flexible semantic mechanism to adopt potentially any query language. Besides, none of these approaches supports reasoning capabilities for query reduction and inconsistent query filtering, as opposed to our system. Regarding these approaches, our approach works at a higher semantic level, as it exploits the background knowledge not only to build new queries but also to infer which ones are satisfiable and to avoid the generation of inconsistent queries.

TAP and Semantic Search

TAP [GM03, GMM03] is a platform to publish and query semantic data that was developed in the early days of the Semantic Web. In particular, its three main objectives were: 1) to provide a simple and predictable query interface to access semantic data, 2) to provide the different data publishers with mechanisms to reconciliate their possibly different vocabularies, and 3) to establish a trust network that allows to validate the trustworthiness of the different accessed data services.

Note that, although the first objective seems to have been correctly addressed with the wide adoption of SPARQL endpoints, the other two objectives remain being important research fields. Anyway, its authors developed an application over TAP called Semantic Search, that was meant to be a demonstration of how the Semantic Web could help and improve the current search methods.

In brief, the search process was divided in two steps: 1) finding a *denotation* (a resource) for the provided search terms, and 2) retrieve data that is related to the *anchor* nodes by exploring the relations in the data graph. The mapping between the input keywords and their denotations was done via the TAP search interface, which performed a syntactic match with the different resources and, when there were more than one possible term for each keyword, presented them to the user to select the specific resource. Moreover, they limited the amount of the possible search terms to at most

two ones.

Once they had the nodes in the graph that corresponded to the denotations, they applied different heuristics to retrieve related resources and enrich the search results. With one anchor term, they focused on retrieving the resources that were connected to it by different relationships, applying different selection criteria such as the amount of resources retrieved so far via that relationship, or the provenance of the data. With two anchor terms, before applying this heuristic, the possible subgraph connecting them was to be selected.

SemSearch

One of the first systems whose goal is building formal queries from keywords in the area of the Semantic Web is SemSearch [LUM06]. They offer a Google-like user interface where the user has to mark the main subject of the search and mark the rest of the keywords as mandatory or optional ones. Then, the input is matched to semantic entities by means of text indexes. The subject keyword must be mapped to a concept entity (the focus of the user query, i.e., the type of the expected search results); otherwise, the system applies several fixed heuristic rules to interpret it correctly depending on the number of input keywords (for example, with two input keywords, if the subject keyword matches an instance and the keyword matches a property, the search results are the values of the matched property for the matched instance).

The result of the matching process is passed to the Semantic Query Layer, where the system applies several predefined query templates to interpret the keywords and build formal queries (in particular, they use SeRQL query language, although SPARQL could also be used). This template-based approach fixes the possible interpretations and, as not all the possible queries are considered in those templates, the system could fail generating the user's intended query. Regarding Semantic Search on TAP, SemSearch goes one step further in the keyword interpretation process supporting complex queries (more than two keywords, although it is done via predefined templates attached to a single query model); however, they do not consider the vocabulary mismatch problem (the indexes SemSearch uses for the semantic entity mapping are preprocessed on the data repository to be accessed).

SPARK

SPARK [ZWX+07] relies on graph construction on a semantic model equivalent to RDF-S to achieve a proper keyword interpretation. To do so, they focus on obtaining a query graph out from the keyword query. They equate *semantic query* to *a query graph with constrained object nodes and property arcs*, which is directly mapped to conjunctive queries.

The interpretation performed in SPARK is constrained to just one given domain, which is provided in the form of an ontology. The resources of this ontology are indexed to perform the *term mapping*, which associates each of the input keywords to one or more resources (due to the possible ambiguity). They use morphological techniques (i.e., substring, stemming, etc.), and expands the mapping space semantically using general dictionaries such as WordNet [Mil95].

Once the terms are mapped, the *query graph construction* step explores the RDF data to construct complete query graphs applying *Minimum Spanning Tree* algorithm. If there are missing edges needed to connect the mapped terms, SPARK adds them by consulting the underlying semantic model. The result is a set of SPARQL queries that are ranked according two different perspectives: According to the keyword input, and according to the semantic model. Regarding the previous approaches, SPARK enables a higher level of expressivity (although it is constrained to simple conjunctive queries) as they can calculate all the interpretations that can be derived from the underlying RDF data. However, this comes at the cost of not scaling well with large data repositories. Moreover, they just work on one domain, and it has to be previously indexed (SemSearch also indexed the semantic entities, but it was multi-domain oriented).

Q2Semantic and SemSearchPro

Adopting a similar approach as SPARK, the works of Tran et al. [WZL+08, TWRC09, THL11] also advocate for graph construction on the RDF data to perform the keyword interpretation. Concerned by the scalability problems of general graph-approaches such as SPARK, they perform a clustering operation on the RDF data to "obtain a graph structure that corresponds to a only a summary of the original ontology", a lightweight ontology. Once they have it, they focus on building the top-k queries that are the possible interpretations for the input keywords. Thus, they adopt a data-driven approach to obtain the semantics that are behind the keywords/ontologies used. The techniques were introduced in Q2Semantic [WZL+08], were de-

tailed and improved in [TWRC09], and finally a compilation of the whole pipeline was presented under the name of SemSearchPro in [THL11].

These approaches, instead of working directly with the underlying RDF graph as SPARK did, perform an offline preprocessing step to build the lightweight ontology out from the RDF data. The authors advocate for this dynamic construction instead of working with predefined ontologies due to the fact that large scale scenarios at Web scale have to deal with dynamic evolving generic data, and in that scenarios they argue that "a schema cannot be defined completely a priori but must also evolve with changes in usage requirements, and with changes in the underlying data". This semantic model is built exploiting the structure similarity of the different RDF resources, which allows to group different elements. In SemSearchPro, it is obtained by applying the notion of *bisimulation*, originating from the theoretical analysis of state-based dynamic systems. As the authors state, "intuitively speaking, two vertices are bisimilar when they share the same structure found in the data graph". The expressive power of this semantic model is lower than RDF-S.

The interpretation process in these systems is comprised by two main steps: A keyword-to-resources mapping step, and a exploration and ranking step. The first step in Q2Semantic is similar to SPARK's *term mapping*. It searches for the keywords in the RDF literals, which are enriched with Wikipedia's terms to try to fill the gap between the RDF repository's and user's vocabularies. In [TWRC09] they improve this matching process by adding a keyword index that also takes into account the types of the resources that the keywords are matched against. The second step is performed on the built summarized graph/semantic model, which reduces the search space with respect to generic-graph approaches. They employ a top-k exploration algorithm [TWRC09] that starts from the matched elements and explores iteratively the graph looking for all the distinct paths from these elements. While traversing the graph, they score the paths according to different factors (such as the popularity of the graph elements, among others) to obtain a ranking of the created queries and focus the search. Eventually, paths are formed between the initial resources and are added as candidate queries. In both systems, the queries to be built are constrained to a restricted type of conjunctive queries, which are directly translated into query graphs in SPARQL.

They achieve a more efficient interpretation method than SPARK. Moreover, they get rid of the need of an ontology describing the domain, as they build their own. However, this is done offline and it depends completely on the underlying data. Finally, they still rely mainly on syntactic techniques

for the initial matching (in spite of being partially enhanced), and are attached to just one data model (RDF with a subset of conjunctive queries as query language).

CoSi

CoSi [FA11, FGA11] adopts a summary graph solution as SemSearchPro does, using also the top-k approach to obtain the most probable interpretations for the input keywords. The main difference between both systems relies on the information that their summarization graphs hold.

CoSi introduces the use of user's query history to achieve a better semantic interpretation. Instead of building their own summarization graph from scratch directly from the underlying data, CoSi relies on a provided schema graph (with an expressivity less than RDF-S) to build a *context-aware summary graph*, which includes a query history dependent weighting function. This function takes into account the locality of the resources (*region factor*), and the query history (*historical impact factor*) to weight the nodes and edges during the top-k exploration.

The interpretation process is similar as the performed by SemSearchPro, with an extra step to update the weights in the summarization graph. First, the keywords are matched to the underlying resources using an inverted index. Then, CoSi uses a graph exploration algorithm to generate the top-k interpretations. After this, the updating of the weights is carried out. The top-k exploration algorithm is also adapted to deal with dynamic weight values and to detect early termination conditions.

QUICK

Finally, QUICK [ZZM+09] adopts a schema-driven approach to perform the keyword interpretation, instead of a data-driven one as the previous systems did. Moreover, they involve the user in the keyword interpretation by allowing him to build the desired query incrementally.

QUICK works on a predefined domain, which is provided by an RDF-S ontology. It uses this ontology to build the complete set of all possible semantic queries for the given set of input keywords. To do so, first, QUICK obtains the possible query patterns for that given schema without considering the input keywords. These *query templates* are compositions of schema elements, and their expressivity is limited to acyclic conjunctions of triple patterns. Then, the actual *semantic queries* are build by binding keywords to the appropriate query templates. The keyword matching is done using a

full-text index, extended with synonyms. Then, to help users to select the intended query, they propose an algorithm to iteratively build it by selecting different subqueries of the intended one.

Working at schema-level makes QUICK not to be so dependent on the underlying data. However, QUICK still needs to build an enriched text index to perform the initial keyword matching, which does not take into account all the possible semantic aspects of the keywords. Moreover, their expressivity is constrained to just one data model (RDF with acyclic conjunctions of triple patterns as query formalism). Finally, they are constrained to just one domain at a time (their query search space grows directly with the size of the schema/s considered).

2.3.2 Keyword Search on Relational Databases

There are also some works in the area of databases to provide a keyword-based interface for databases, i.e., translating a set of keywords into SQL queries, such as BANKS [BHN+02,ABC+02], DBXplorer [ACD02], and DISCOVER [HP02]. However, they focus on how to perform keyword searches efficiently on the relational data, overcoming the problem of having the relevant data distributed among several tables (mainly due to normalization). Moreover, as emphasized in [BDG+11], most of these works rely only on extensional knowledge obtained by applying IR-retrieval techniques, and so, they do not consider either the intensional knowledge (the structural knowledge), or the semantics of the input keywords.

The graph-based approach adopted by BANKS [BHN+02, ABC+02] has strongly influenced another works such as SPARK or Q2Semantic. In this system, the database is modeled as a directed graph with each tuple being a node in that graph. The foreign-key-primary-key links are the edges between the different nodes. With this built graph in memory, BANKS uses disk resident indexes to search for the nodes that exactly match the search keywords. Once it has obtained them, it builds join-trees that connect all the matched nodes. These trees are rooted in a *information node*, which can be restricted to be from a selected set of nodes of the graph.

Also considering the database as a graph of interconnected tuples, DBXplorer [ACD02] and DISCOVER [HP02] work with specially designed data structures stored in the same database. The former one builds a *symbol table* which indexes the database associating keywords with their locations within it. This makes it possible to determine where the query keywords appear efficiently (i.e., the tables, columns or rows, depending on the granularity level chosen). Then, it works out all the subsets of tables that, when

joined, might contain rows with all the keywords. Finally, for each of these combinations, they build an SQL statement to retrieve the rows with all the keywords. DISCOVER also relies on a keyword index (*Master Index*) at row granularity to perform the initial keyword search on the database. Then, it works at row level to create *candidate networks*, sets of tuples that cover the input keywords. As the authors state, this allows DISCOVER to consider more solutions than DBXplorer does (e.g., solutions that include two tuples from the same relation).

More recently, Keymantic [BDG+11] proposes an approach that is the most related work to ours in this field. In this case, authors focus on mapping the keywords to entities of the relational schema of a database and interpret them as an SQL query to enable keyword based search over databases without having to process the extensional data, which is an important improvement when we only have access to the database schema. Moreover, when matching each input keyword to the database elements, they have also into account the influence that rest of the input keywords has in the query meaning. Then, once they have obtained the set of most feasible mappings, they apply a greedy algorithm to derive the whole set of possible Select-Project-Join (SPJ) queries to be posed to the database. This approach also helps the user to understand the database schema in a exploratory way. However, as we will see, our approach is more flexible as it is capable of: 1) obtaining the semantics of the keywords without specifying a target schema, 2) interpreting the queries into different query models, taking into account the semantics of all the elements (keywords, query language, operators, etc.), and filtering the inconsistent ones; and, finally, 3) accessing different underlying data models considering the previously well-established semantics.

2.3.3 Question Answering Systems

Finally, we overview the broad field of Question Answering systems [ART95, LUSM11]. Although this kind of systems is traditionally more related to the processing of Natural Language (according to [HG01], their goal is "to allow a user to ask a question in everyday language and receive an answer quickly and succinctly, with sufficient context to validate the answer"), in essence, QueryGen shares their objectives.

According to the classification given in [LUSM11], our system would fall into the category of ontology-based semantic QA systems, taking keywords as input. They are traditionally attached to a specific knowledge domain and/or underlying data source/model , which guides the translation process.

Using several different techniques, we tackle some of the traditional problems that these systems face [LUSM11]:

1. The vocabulary mismatch between the user's vocabulary and the underlying repository. As we will see, QueryGen maps the vocabulary of the user to the vocabulary of the data sources (provided that the data sources are semantically described, and the ontology is made available).

2. As we have seen in Section 2.2.1, Natural Language is inherently ambiguous and, of course, language-dependent. When using keywords, in spite of introducing ambiguity due to the lack of expressivity of the keyword model, we provide users with a multilanguage way of expressing their information needs. Moreover, our approach disambiguates the different possible interpretations due to polysemy (and, in our case, also due to the lack of expressivity of the keyword model).

3. Many QA approaches present domain-specific limitations, a limitation that can be overcome by using techniques to consult and analyze a dynamic pool of ontological sources.

4. QA approaches usually need a well formed input to achieve good results (complete sentences). In this thesis, we deal with this lack of information by retrieving semantic information relevant to the query.

5. To the best of our knowledge, no QA approach exploits the semantic knowledge not only to interpret the query, but to filter inconsistent queries.

However, we have to bear in mind that the premises which QA systems and QueryGen build on are quite different, and we are aware (and we are working on it) that we can introduce several techniques from these systems to improve the whole semantic keyword-search process that we are presenting in this work.

Chapter 3

Semantic Keyword-based Search

In this chapter, we firstly present a lightweight approach to perform keyword search guided by the semantics of a domain on Linked Data repositories. Then, we move onto our main approach, where we introduce the problem of interpreting the keywords of the users, and give an overview of our approach to it. This problem (*keyword interpretation*) is an ill-posed one due to different difficulties, such as the lack of expressivity of the keyword query model, the ambiguity introduced by the polysemy of the keywords, and the possible omission of implicit keywords. Thus, we advocate for a *semantic keyword-based search*, which takes into account the semantics of all the elements that participate in the process to reduce the impact of the main problems of keyword query model.

3.1 Lightweight Semantic Keyword Search on Linked Data

In this section, we give an overview of the lightweight semantic keyword search we have developed to enhance the underlying knowledge of Embodied Conversational Agents (ECAs) [CSPC00]. ECAs are graphical interfaces capable of using verbal and non-verbal modes of communication to interact with users in computer-based environments. The appearance of these agents varies depending on the application scenario: They might be as simple as just an animated talking face, displaying simple facial expressions; or they can be as complex as to have a sophisticated 3D graphical representation, with complex body movements, and emotional and facial expressions.

In [BEM12,BEM13], we studied the possibility of using the sheer amount of information available behind Linked Data endpoints to enhance the information handled by this kind of agents, and, in particular, we focused on the case of DBpedia. The conversational nature of the interaction with the ECAs introduced an important constraint: We could not abuse of disambiguation dialogues to avoid annoying and distracting the user with them. As the speech recognition by itself might need disambiguation questions, we only could use as input the plain keywords that an ECA recognized and forwarded us without further information. Thus, having no control on any user's feedback made us adopt a pragmatic approach.

This approach is based on defining externally the search domain, which provides a view on the underlying data (similar to views on databases). This way, we allow the ECA to exploit the structure of the underlying data to focus its searches, only analyzing semantically related resources. The search domain is defined by using an annotated ontology provided by the administrator of the system (see Figure 3.1). This ontology provides an adaptable view on the underlying data and has to be aligned to the ontology that describes the actual data repository. In fact, although it can be built from scratch, we advocate for using ontology extraction techniques [JCS+08] to obtain a module and, then, make our system work directly with a subontology of the repository's one.

Once it has the Domain Ontology, our system offers two different but complementary kinds of search depending on the user's input (see Figure 3.1):

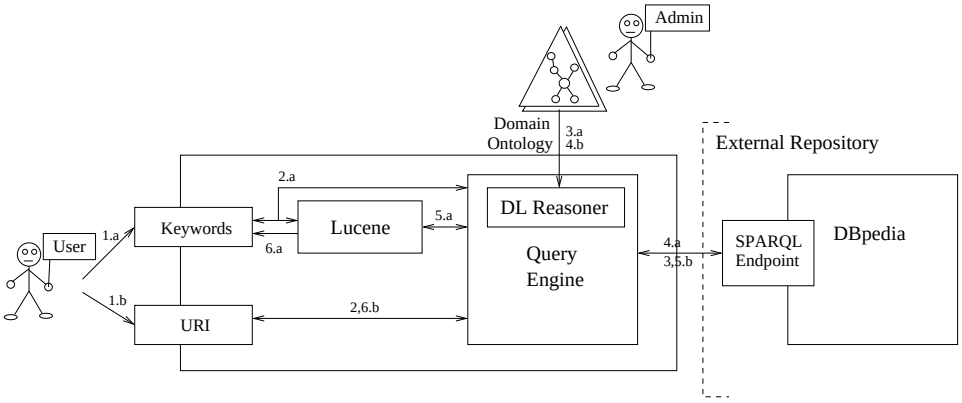


Figure 3.1: Our system provides two complementary search services: a) Keyword-based and b) URI refining services.

1. *Keyword-based Search.* This search service takes plain keywords as input (step 1.a) and checks out whether the search has been processed before (step 2.a). The system has an internal Lucene[1] repository that serves as a cache memory to alleviate the workload of the external public endpoint (which is not under our control and might have limited availability). If the search has not been performed before, the keyword query is forwarded to our Query Engine which consults the concept hierarchy of the ontology to build a focused SPARQL query (step 3.a). This hierarchy includes only the objects we want to be searched, that is, the objects that we define in the search domain. If it is too large, there exists the possibility of specifying a class of the Domain Ontology to serve as top node of the focused search. Once the Query Engine has built the query, it poses the query to the public endpoint to perform the actual search (step 4.a). When the results are retrieved, they are stored in our Lucene repository to cache them for future searches (step 5.a). The Lucene repository acts as a cache and provides the system with relevance measures and ranking on the results. Finally, the results (a set of URIs) are returned and ranked according to their relevance (step 6.a).

2. *URI Refining Search.* The results of the previous search service is a ranked set of URIs, which are presented to the user so s/he can explore them. When the user selects a URI (step 1.b), it is directly forwarded to the Query Engine (step 2.b). The Query Engine consults the data repository to obtain the type of the object behind the URI (step 3.b). Then, it consults the definition of its type (step 4.b) to build a set of specialized queries for that type of object (it consults the relevant properties[2] to retrieve the appropriate data and suggest other related objects) with the help of a DL reasoner. Finally, it forwards these queries to the data endpoint (step 5.b) and returns the data (step 6.b). This step is not cached as these more specialized queries are not so time consuming as a general search over the whole domain.

Notice that the Lucene repository is only used to cache and rank the results obtained from the actual query on the external Linked Data repository. We assume that the results can be cached as the Linked Data repositories are in fact quite stable in time (e.g., there was a lapse of seven months from

[1] `http://lucene.apache.org/core/`, last accessed October 3, 2013.

[2] They are marked as being relevant during the ontology definition, so the Query Engine can be aware of them.

the release of version 3.6 of DBpedia to the 3.7 one, and a year between 3.7 and the latest one, 3.8). Anyway, our system can be easily adapted to another scenarios where data are more volatile by deactivating the caching mechanism.

We refer the interested reader to [BEM12, BEM13], where the details about the used annotations and the adaptation of the system to deal with DBpedia is thoroughly described. This approach, however, does not superseed the main approach that is presented in this thesis, as it stills does not consider the actual semantics that are behind the input keywords. In fact, the benefits of this approach comes precisely from considering the semantics of the domain to filter the results that lie within it. As we will see in the following, there are much more semantic information which we can take profit from to really perform a semantic keyword-based search.

3.2 Keyword Search vs. Semantic Keyword-based Search

The usage of keyword search has spread in the last years thanks to its simplicity and its adoption by the main Web search engines. Common users have found in it an easy way to express their information needs, defining their searches just by giving a plain set of keywords and letting the system do all the work for them. However, the ease of use of keyword search comes from the simplicity of its query model, whose expressivity is low compared with other more complex query models [KB10], as it can be seen in Figure 3.2 [KB10, FCOO12].

This implies that the queries that users can pose to the search systems are limited by this lack of expressivity. In fact, keyword queries are simplifications of the queries that really express the user's information need. Thus, there might be a gap between the posed query and the retrieved information that must be filled by the users, e.g., when talking about Web searches, users usually have to browse the returned Web pages looking for the needed information.

Moreover, polysemous words introduce ambiguity in the queries that cannot be solved without the intervention of the user. For example, behind the keywords *'apache attack'*, a user might be looking for information about the Apache helicopter or about how to secure an Apache server. One could argue that the ambiguity in this example is due to the lack of input keywords, but experience tells us that the average number of keywords used in keyword-based search engines "is somewhere between 2 and 3" [MRS08],

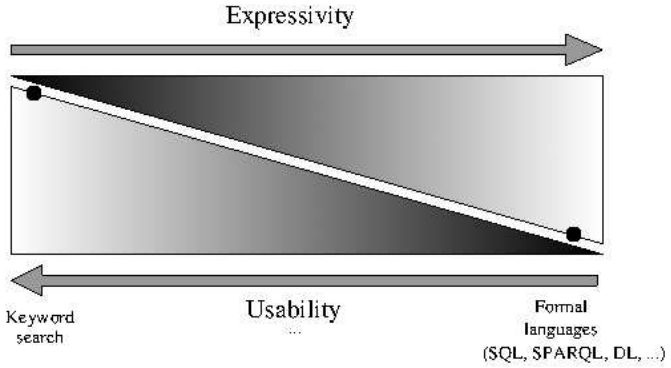


Figure 3.2: Trade-off of between the expressivity of the querying language and its usability (taken from [FCOO12], and adapted from [KB10]).

which points out another problem of keyword search: Users tend to omit important keywords, as they consider them implicit in the query.

On the other hand, the use of expressive formal languages (such as SQL or SPARQL) is far from being easy for common users. Moreover, to effectively use formal languages, the user must have previous knowledge of the underlying schema and data s/he is accessing. Thus, the sweet spot would be to mix the expressivity of formal languages with the ease of use of keyword queries, while making the user unaware of the data sources being accessed to solve her/his information needs.

To reach this sweet spot, we advocate for a *semantic keyword-based search*, a keyword-based search process in which semantics of both keywords and query languages play a crucial role during the whole search process. Our objective is to discover and solve the user's information need taking as starting point a set of input keywords. We divide this task into three sub-objectives:

- To discover the exact meaning of each of the keywords in the set of input keywords.
- To give them an interpretation and express it into a formal language to capture the information need accurately.
- To access the proper information system/s transparently to the user, taking into account the different characteristics that the accessed systems might exhibit.

The first objective, the discovery of each keyword's meaning, allows us to work during the whole process with keywords with well-established semantics, which we call *semantic keywords*. The second one implies structuring a bag of keywords into a structured query, a process which is named *keyword query interpretation* [FA11]. The achievement of the last objective is strongly helped by having the information formally expressed, allowing our system to access semantically even to non-semantically enhanced data sources.

3.3 Generalized Keyword Interpretation

In its simplest form, an information system can be characterized as a tuple $IS_m =< Q_m, D_m, AS_m, f_m >$ where:

- Q_m is the query model and defines the rules to build queries that the information system can answer. We will denote as Q'_m as the set of possible queries that can be built within the query model Q_m.
- D_m is the dataset that the system IS_m can access to provide answers to the different queries.
- AS_m is the set of possible answers that the information system can provide.
- $f_m : Q'_m \rightarrow P(D_m)$ is the function of the system that maps a query q_i to the answer set AS_i. It can involve different subsystems, processes, or even people activities.

Note that AS_m will usually correspond to $P(D_m)$, i.e., the power set of D_m, but could not be the case as in Question Answering systems, which can answer directly *true* or *false*. For example, for a Web search engine based on keyword search, $Q_{WSEngine}$ would be easily defined as the well known *bag of keywords* model, $D_{WSEngine}$ would be the whole set of URLs in the Internet, $AS_{WSEngine}$ would be $P(D_{WSEngine})$, and $f_{WSEngine}$ would be the process that allows to retrieve the URLs of the documents that are relevant for the input. Thus, with this characterization, we will abstract completely from the way that the information system computes the answer set, and the possible ranking schemes that could be applied to it. We will also consider that a query model is defined by its associated query language, that is, the operands and operators that are available to build up queries in that query model, and their given semantics.

To perform a keyword interpretation for a target information system IS_m, we have to provide a relation $K_{int} : Q'_{kwd} \rightarrow P(Q'_m)$ that allows us to structure the keyword query taking into account the target query model. Ideally, if both query models would had the same expressivity, this relation would be a function, that would map a keyword query into just one formal query. Unfortunately, due to the lack of expressivity of the keyword query model, there will usually be far more than one interpretation for a given set of keywords.

Moreover, we do not want just to adopt a single target query model, but also to generalize the process to be able to perform an interpretation into several different query languages. To do so, we want to provide a translation process that works at a meta-description level, i.e., it takes the description of a target query language and performs the interpretation taking into account this description. The description of the target query models will be provided via the specification of their query languages in the form of augmented grammars, where the properties of their operators are explicitly stated. Therefore, we are looking for a generalized interpretation relation $K'_{int} : Q'_{kwd} x Q_m \rightarrow P(Q'_m)$ that allows us to adapt the keyword query model to any more expressive query model.

The search space for such a generalized keyword interpretation process depends on the number of operands of the target query language and their semantics, as they will define the means in which we can combine the input keywords to form a formal query. Orthogonally to other approaches that relies on ranking schemes and top-k solutions, we advocate for taking into account both the semantics of the input keywords and of the operators of the query models to reduce this search space drastically.

In the following section, we describe our approach to obtain this interpretation relation, which is focused on the semantics of all of the elements that participate during the process.

3.4 QueryGen: Architecture of the System

In this section, we present the whole pipeline that allows us to go from plain keywords to semantic queries expressed in formal languages and, finally, access to data stored in different data repositories. The semantics behind the input set of keywords, the semantics of the different query languages, and the different semantics of the access models are taken into account to provide a flexible and efficient way to perform a semantic keyword search on heterogeneous information systems. In this process, we differentiate three

main steps (see Figure 3.3):

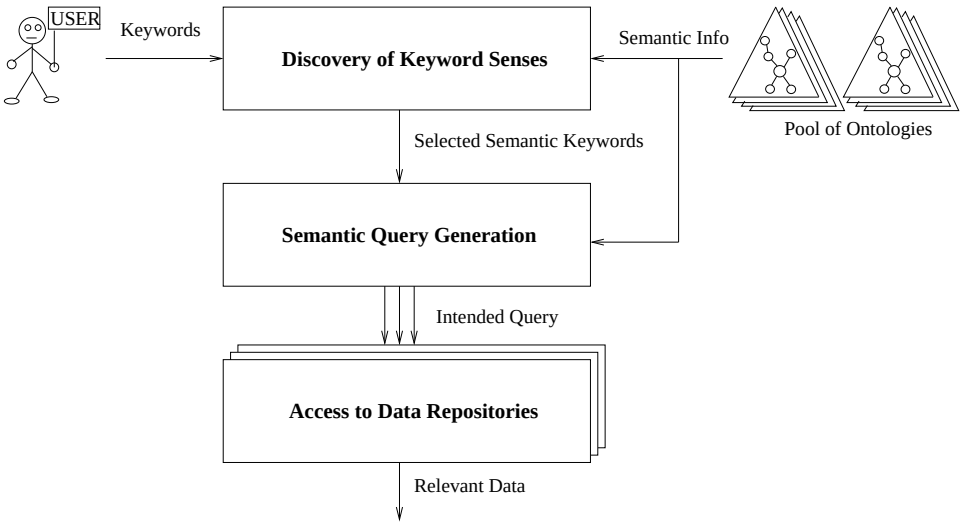


Figure 3.3: An overview of the whole process: from plain keywords to data access.

1. *Discovery of Keyword Senses*: First of all, our system obtains the exact semantics of the input keywords to transform them into *semantic keywords* (keywords with well-defined semantics). To do so, it consults a pool of ontologies to discover and extract the possible meanings of each keyword. During this step, the selected meanings for each keyword that are similar enough are integrated and merged to avoid redundancy (when a *synonymy* value that is calculated is above a threshold [GdM09]). Then, our system applies different disambiguation techniques to establish the meaning for each keyword by taking into account its context (the possible meanings of the rest of the keywords). This disambiguation process can be run fully automatic (implying that our system will work with the most feasible interpretation for each of the keywords in the input set taking into account the rest of the input); or it can be semi-automatic, where the user might be required in this step to validate that the actual offered meanings are satisfactory (the different possible meanings for each keyword are ranked according to their meaning probability, and presented to the user to allow him/her to choose the appropriate ones). This disambiguation process allows the system to map the input keywords to ontological terms, namely *concepts*, *roles* and *instances*.

However, this is only a first step towards obtaining the semantics of the input. Several queries might be behind a given set of keywords, even when their semantics have been properly established individually. For example, given the keywords "fish" and "person" meaning "a creature that lives and can breathe in water" and "a human being", respectively, the user might be asking for information about either biologists, fishermen, or even other possible interpretations based on those individual keyword meanings.

2. *Semantic Query Generation*: The output of the previous step is a set of keywords which has its meaning properly attached, which we call *semantic keywords*. The ontological information that has been considered for obtaining the meaning of each keyword comes along with each of them. Our system automatically integrates this information and, then, automatically builds a set of formal queries which, combining all the keywords, represents the possible semantics that could be intended by the user when s/he wrote the list of plain keywords. The semantic keywords obtained in the previous step are combined according to certain annotated abstract grammars (there is one for each query language made available to our system). These grammars lack syntax sugar and define how to combine the operators of a query language with *typed gaps*, i.e., they specify which kind of queries can be built using concepts, roles, and instances in the corresponding query language (e.g., *And concept concept*). The result is a set of *abstract queries* that the system materializes into a list of actual queries by substituting the typed gaps by input keywords. Finally, the set of (syntactically correct) generated queries are *semantically* filtered using a DL reasoner and the integrated information.

   When no query satisfies the user, our system performs a semantic enrichment of the input by adding *virtual terms*. They are generic typed gaps (to be replaced by concepts, roles, or instances) that represent the keywords that the user might have omitted, but without whom the intended query cannot be built. In a new query generation step, our system treats them as regular typed gaps but, instead of being replaced by input keywords, they are substituted by terms obtained from the ontologies which the input keywords were mapped to (during the previous discovery step). Thus, any query that the user could have in mind will be generated as a candidate interpretation as long as the available query languages are expressive enough.

This query generation process has both a syntactic and semantic dimension: It generates only syntactically correct queries according to the grammar of each of the query languages, and it takes into account the semantics of the operators of each language and the semantics of the keywords to avoid generating either duplicated or incoherent queries. This process is performed in parallel for each available query language as their expressivity can differ from each other.

3. *Access to Data Repositories*: Finally, once the user has validated the generated query that best fits her/his intended meaning, the system forwards it to the appropriate underlying structured data repositories (databases, Linked Data endpoints, etc.) that will retrieve data according to the semantics of such a query. This is not a trivial task, as our system must be capable of adapting itself to their different query processing capabilities and access methods, and to their different data models and formats of the retrieved data.

   This is done via *Adapters*, an evolution of the notion of *wrappers* used in OBSERVER [MI01]. These *Adapters* encapsulate both the access methods and the actual syntaxis of the query languages and data formats, allowing QueryGen to abstract from them. Thus, we can add new information systems to feed QueryGen just by implementing and registering an appropriate Adapter in the system.

In the following chapters, we include a detailed description of each of these three main steps.

3.5 Summary of the Chapter

In this chapter, firstly, we have presented an approach that allowed to perform semantics-oriented keyword search on Linked Data repositories. This lightweight approach was motivated by the actual restrictions that the interaction with ECA's and the parsed natural language interface imposed. Anyway, it gave us an important insight on how to exploit already available Linked Data repositories in a flexible way, without incurring on overloading the external endpoints.

Then, we have moved onto our main proposal: We have introduced the problems of translating queries expressed within the keyword query model into other more expressive query models, the so-called *keyword interpretation* process. The current approaches that tackle this problem restrict

themselves to one target query and data model, due to the ill-posed nature of the problem. This is a limitation that we wanted to get rid of.

Thus, we propose a generalized keyword interpretation approach where the semantics of all the elements involved in the process are taken into account. Firstly, the actual semantics behind of each of the keywords is discovered and associated separately. Then, using the semantic descriptions of the target query languages, our system generates the possible interpretations for the *semantic keywords* in the target query models. Finally, our system, via the use of *Adapters*, is able to access to the registered underlying information systems.

Chapter 4

Discovery of the Semantics of the Keywords

In this chapter, we explain how QueryGen obtains the meaning of each of the input keywords, which is the first step towards the correct input interpretation. First, we present the discovery and disambiguation method that our system uses to obtain the keyword meanings. In this step, the system builds the *senses* that are behind each keyword, which are used all along the process. Then, we explain the inner structure of the disambiguation module in detail to see how QueryGen keeps all the information updated in an efficient way. Finally, we introduce the method by which the system integrates all the information obtained during the process to make it available to the query generation module.

4.1 Disambiguating the Input Keywords

To fully understand our approach, and before giving any further details, we have to introduce the exact meaning of *sense* in our system: A sense is the precise meaning of a keyword in a context, i.e., its surrounding keywords determine which meaning this keyword has. In particular, a sense is represented by a tuple formed by the term itself, an ontological context that comprises a list of possible synonyms (with their URIs) and ontological information about the term, and a description in natural language. Each ontological context is built by integrating information from different ontologies. Figure 4.1 shows some possible senses for user keyword *star* retrieved from online ontologies.

So, the first step that our system performs is to discover and build these

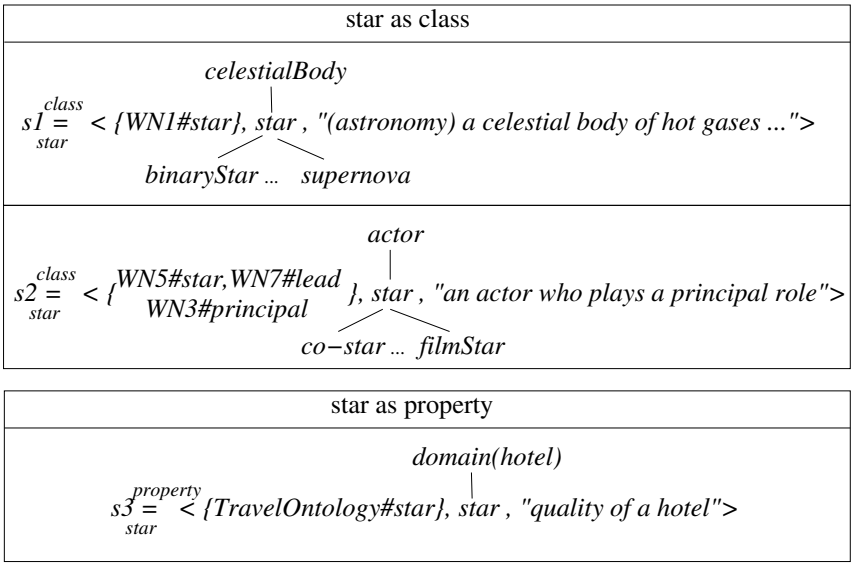


Figure 4.1: Possible senses for keyword *star*.

senses for the plain input keywords. This discovery of the semantics behind each one of the input keywords is done by taking into account their individual possible semantics as well as the possible semantics of its context (the rest of keywords), following the proposal in [TGEM07]. In particular, this process is divided into three substeps (see Figure 4.2):

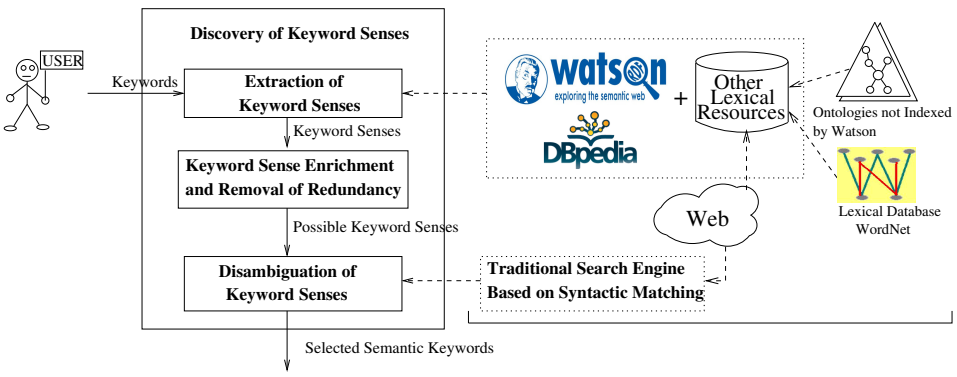


Figure 4.2: Discovery of keyword senses.

- Extraction of Keyword Senses: The system extracts out the possi-

ble meanings of each keyword from a dynamic pool of ontologies (in particular, it queries Watson [dBG+07], DBpedia [BLK+09], WordNet [Mil95], and other ontology repositories to find ontological terms that syntactically match the keywords - or one of their synonyms). The system builds a sense for each matching obtained, and then, the extracted senses are semantically enriched with the ontological terms of their synonyms by also searching in the ontology pool. The result is a list of candidate keyword senses for each user keyword. In Figure 4.1, three possible senses (two as a class and one as a property) retrieved for user keyword *star* have been shown.

- Keyword Sense Enrichment and Removal of Redundancy: As the obtained senses were built with terms coming from different ontologies, they could represent the same semantics. An incremental algorithm is used to align the different keyword senses and merge them when they are similar enough. To assess the sense similarity, our system calculates a *synonymy probability* that considers both linguistic and structural characteristics of the source ontologies: The linguistic similarity is calculated considering the different labels of each term as strings; and the structural similarity is calculated recursively exploiting the semantics of the *semantic keywords* (their ontological context, see Figure 4.1) until a certain depth. Finally, both similarity values are combined to obtain the resultant synonymy measure[1].

  Senses are merged when the estimated *synonymy probability* between them exceeds a certain threshold[2]. Thus, the result is a set of *different* possible senses for each user keyword entered.

- Disambiguation of Keyword Senses: A disambiguation process is carried out to select the most probable intended sense of each user keyword by considering the possible senses of the rest of keywords. The senses are compared by combining [GM09]: a) a Web-based relatedness measure, that measures the co-occurrence of terms on the Web according to traditional search engines such as Google or Yahoo!, b) the overlap between the words that appear in the context, and the words that appear in the semantic definition of the sense [BP03], and c) the frequency of usage of senses (when available, as in WordNet annotated

[1] The formulae for the synonymy for each type of senses (concepts, roles and instances) can be found in [TGEM07].

[2] In [GdM09], the authors proposed several strategies to obtain this threshold and validated them via thorough experimentation.

corpora). Thus, the best sense for each keyword will be selected according to its context. Note that this selection can require the user's feedback to select the most appropriate sense for each keyword in a semi-automatic way.

This discovery and disambiguation algorithm, which has been summarized here, is thoroughly described in [TGEM07], and has been applied successfully to very different tasks such as ontology matching [GM08], the integration of senses in semantic repositories [GdM09], or the construction of multi-sourced ontologies [BMT12].

So, the result of this disambiguation is a set of possible senses for each keyword, and the probabilities of each of them to be the correct one according to the rest of input keywords. In the following section, we explain how our system manages this information to expedite further searches while adapting itself to changes in the source ontologies (ontology evolution).

4.2 Architecture of the Disambiguation Module

As we have seen in the previous section, the objective of the first step in QueryGen is to obtain *senses* out from the input keywords. As it might be a costly process, we also want QueryGen to store and manage these senses efficiently to speed up following interactions. In Figure 4.3, the inner architecture of the Disambiguation Module is shown. There are two main components:

1. *Multi-Ontology Senses Library:* When the user inputs its keyword query, the Multi-Ontology Senses Library is consulted with it. This library contains an index of sets of keywords with their possible meanings in the form of senses.

   An intelligent agent, *Librarian*, decides when to build a new entry for the senses or to update and integrate the possibly existing ones. If the *Librarian* has to disambiguate the meaning of the keywords ($\{k_i\}$) or to widen the semantic information that it has for each one of them, it can use the Disambiguator and request the help of the user to choose the most appropriate meanings ($\{S_i\}$). More details can be found in Section 4.3.

2. *Ontology Library:* Once the system has the senses attached to the input keywords, the Ontology Library stores an ontology associated to the set of senses. This ontology is integrated gathering all the semantic

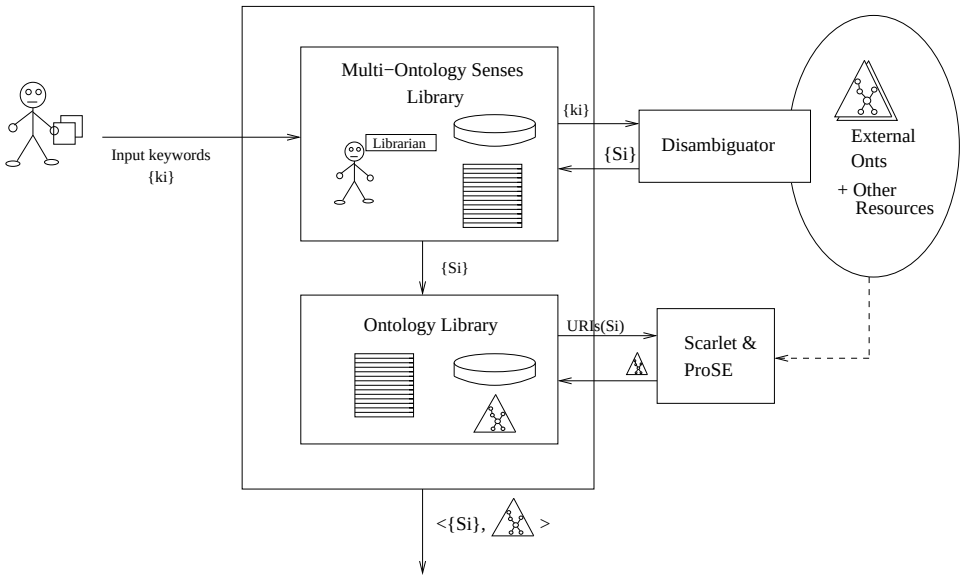


Figure 4.3: Overview of the architecture of the Disambiguation Module.

information regarding the senses together, which is extracted from the ontologies referenced in them. In this step, external services can be required to extract and discover more information (in particular, our system uses Scarlet [SdM08] and ProSÈ [JCS+08]). In Section 4.4, the different possibilities to integrate these ontologies are detailed.

In the rest of the section, we present both modules in detail. In the case of the Ontology Library, we also explain how the ontological information stored in the senses is exploited to integrate the ontologies used in QueryGen.

4.3 Multi-Ontology Senses Library

The Multi-Ontology Senses Library is composed of two main blocks (as shown in Figure 4.4), and an intelligent agent *Librarian* which takes care of both. The first block is the Disambiguation Storage. It contains the different results of disambiguating a set of keywords along with their different probabilities of being the proper interpretation. When a set of keywords is looked up, it returns a probability-ordered list of tuples formed by the probability and a list of corresponding references to the senses. In this search, the information about the synonyms of the keywords is taken into

account to avoid missing any possible interpretation. The library tracks the senses with unique IDs, so homonym senses cannot be mistaken (otherwise, the disambiguation process would be useless).

The other block is the Sense Library. It is an storage for the senses maintained up to date by the *Librarian*.

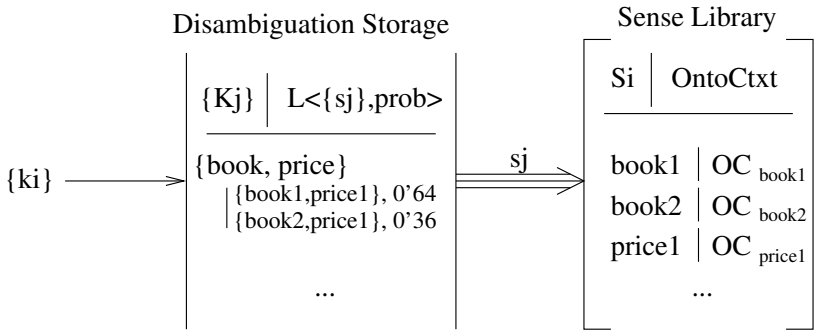


Figure 4.4: Organization of the information managed by the agent *Librarian*.

The system can consult this senses library in two ways: Using sense references or plain keywords as input. When using the former only the Sense Library is accessed, while when using the latter the Disambiguation Storage is. When working with keywords as input, if they do not fully match any keyword set, then partial coverages are considered, and, if the access fails again (or the user declines all the offered interpretations), then the *Librarian* starts a new disambiguation process and the newly obtained senses are inserted. To insert a new sense:

1. The *Librarian* obtains possible synonyms of the newly built sense (the one to be inserted).

2. Then, it looks in the ontological contexts of the already inserted senses for matches in the list of possible synonyms in order to avoid duplicates in the Sense Library. The result is a set of senses which are possibly equivalent.

3. In a parallel way, it checks whether the new sense and the candidates to be equivalent to it are so. If not all of them are, the new sense is inserted with a new unique ID. Otherwise:

   - The *Librarian* integrates the senses and inserts a new one in the Sense Library.

- All the references to the former sense are updated and the ontologies in which the sense participates are tagged as obsolete.

By inserting new senses and updating the existing ones, the system can minimize the effects of the evolution of the source ontologies, as the *Librarian* can take care of the senses being up-to-date.

4.4 Ontology Library

The different senses stored in the Multi-Ontology Sense Library and their different possible combinations can lead to different knowledge sets about the domain they are attached to. The Ontology Library is the component responsible for writing down that multiontology knowledge and storing it in different local ontologies to be used for interpretation purposes. Each one of these ontologies is associated to the senses which have taken part in their creation, and is updated as the senses evolve in time (when introducing new senses in the Multi-Ontology Sense Library, the already existing senses can be affected and modified). In Figure 4.5, the information stored is shown. As in the Disambiguation Storage previously explained, the integrated and stored ontologies are associated to a set of references to their conforming senses, and so, the system is able to take into account partial coverages of the inputs when retrieving the ontologies.

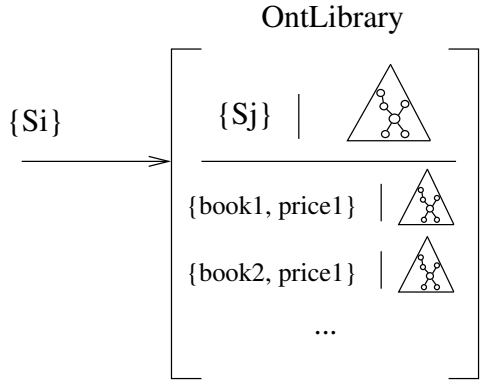


Figure 4.5: The ontologies are stored and indexed by their originating senses.

Together, the Multi-Ontology Senses Library and the Ontology Library, can be seen as a real implementation of the system envisioned in the position paper [Ala06]. In the rest of the section, we explain how the Ontology

Library exploits the information retrieved during the disambiguation process to integrate these ontologies, and present an example of the process.

4.4.1 Integrating the Ontological Information

As we have seen before, the result of the disambiguation step is a *semantic keyword* (a keyword and its selected sense) for each keyword input by the user. These semantic keywords have inner information about the *ontological context* of themselves, this is, the semantic information that has been consulted during their construction and that defines them. This information is multiontology-sourced and has been merged during the keyword disambiguation step. In this section, we present the different levels of information that are used to enrich and complete the information about the semantic keywords in the final ontology, and then we give an insight of the method that is used to integrate it.

4.4.1.1 Ontological Information Considered

Once the input keywords and their corresponding senses have been disambiguated, a multi-sourced ontology can be integrated. In Figure 4.6, the situation that our system confronts is depicted. The information retrieved to obtain the semantic keywords is stored in their ontological contexts, and, in principle, they are isolated from each other even when they might share sources. Thus, the relationships between ontological contexts can be stated at different levels:

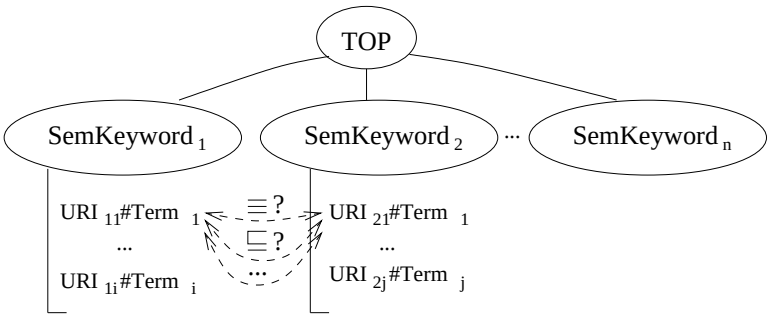


Figure 4.6: The system has to discover the possible missing relationships.

- Sharing a term from an ontology: It is very frequent that, when disambiguating a set of keywords, the same ontology is consulted and

different ontological contexts share terms as they affect different keywords definitions.

- Sharing source ontologies: When two ontological contexts do not share ontological terms, they might still share source ontologies. This has to be taken into account as the keywords that are being disambiguated might belong to the same domain.

- Sharing concepts: Even when neither a term nor ontology is shared, there might exist further relationships such as synonymy between the terms of different ontologies.

So, to avoid the possible isolation, and to find out the possible relationships and enrich the information in the final ontology, the system considers three levels of ontological information, as suggested in [BM10]:

1. *Semantic keyword information.* It is the skeleton of the resulting ontology, and, as it contains the original URIs of the terms involved in each sense definition, it allows to extract more information from the original source ontologies. This information is asserted altogether in one knowledge base and enriched with the information further retrieved. If no more information were available, the resulting ontology would have to be considered as lightweight one (shallow ontology). If two ontological contexts share an ontological term, it is stated at this information level.

2. *Automatic modularization and ontology reuse.* To obtain the original definitions of the ontological terms involved in the ontological contexts, the system uses ontology modularization techniques [dSM06]. In brief, given a set of terms of an ontology, these techniques enable us to obtain a module that is equivalent to the whole ontology regarding what can be inferred about the given set of terms. This allows our system to obtain the complete definitions of the terms and all the existing relationships between terms coming from the same ontology (intra-ontology relationships).

3. *Simple inter-ontology relationships.* Finally, the system can discover inter-ontological relationships using Scarlet [SdM08]. Scarlet is able to obtain disjointness, inheritance and named relations information (i.e., roles that have the pair of concepts as domain and range or vice versa). It also gives an explanation about how the relationship has

been found, and our system only includes it if the reasoning path uses ontologies included in the ontological context.

The possible redundancies in the information (information of different levels may overlap) of the resulting ontology are automatically removed with the help of a DL reasoner. By asserting all the information and classifying it, we obtain a final version of the ontology.

4.4.1.2 Insight of the method

We now turn our attention to the integration algorithm itself. To achieve a better understanding of the algorithm, the general structure of the integrated ontology is shown in Figure 4.7. The semantic keywords set $\{SK_i\}$ is the result of the keyword disambiguation process, and the input of this step. The algorithm is shown in Algorithm 1. It assumes the existence of a global storage that is used to track the source ontologies and the terms that appear in the ontological contexts of the senses (*ontologyURIStorage*), an extractor that encapsulates the modularization techniques, and a reasoner that is finally used to filter out the possible redundancies.

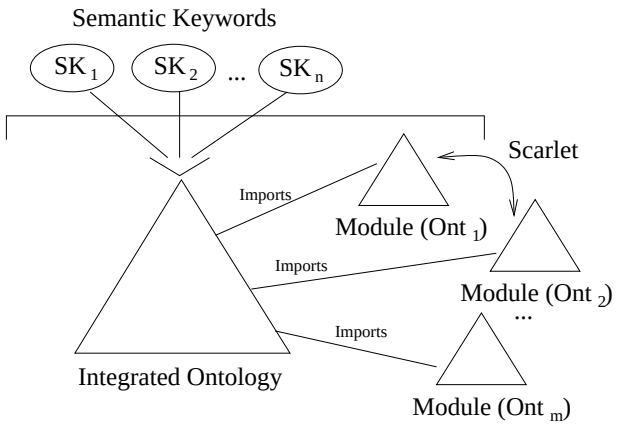


Figure 4.7: Structure of the integrated ontology: our system consults different information sources to enrich and integrate our resulting ontology.

First, after initializing the storages (lines 2-3), our system translates each of the semantic keywords into OWL (lines 4-6) and stores their translation. During this process, all the URIs of the source ontologies are registered in the *ontologyURIStorage*, and, along with each of them, a list of all the terms that are defined in each ontology is stored. These lists are the signatures

Algorithm 1 Integration Algorithm

```
1: procedure sensesToOnt (Array semKeywords, Extractor extractor)
2: ontologyURIStorage.clear()
3: translations.clear()
4: for all sk in semKeywords do
5:     translations.add(translateSense(sk, ontologyURIStorage))
6: end for
7: for all uri in ontologyURIStorage.getOntologies() do
8:     // each module is stored separatedly
9:     tmpModule = extractor.getModule(uri, ontologyStorage.getSignature(uri))
10:    writeDownModule(tmpModule)
11: end for
12: writeDown(translations) // we write down all the translations in the main ontology
13: writeDown(importAxioms) // we write the import axioms to include the modules
14: // we finally use Scarlet to obtain inter-ontology relationships
15: scarletInformation = filter(Scarlet.obtainRelationships(ontologyURIStorage));
16: writeDown(scarletInformation)
17: reasoner.classify(newlyIntegreatedOntology)
18: reasoner.writeDown(ontologyWithoutRedundancies)
19: end procedure
```

(i.e., the list of terms that the module must include) that the extractor uses to obtain each corresponding ontology module (lines 7-11). In particular, our prototype uses ProSÉ [JCS+08], although the method is designed to work with other module extractors. Then, the translations are written in the main module together with the *import* axioms that will include the different modules in the final ontology (lines 12-13).

In this stage, Scarlet is used to discover possible relationships between terms in different ontology modules (lines 15-16). Scarlet can discover relationships following paths between different ontologies, establishing the relationships between terms of two ontologies through relationships with terms of another ones. To avoid introducing ontologies that our system has not processed, we filter out paths to only accept direct relationships (source and target terms belong to ontologies that have been registered in the *ontologyURIStorage*). Finally, we use a DL reasoner to classify the ontology (lines 17-18) and then write down the classified ontology. This allows us to: 1) eliminate redundant axioms, and 2) detect possible inconsistencies in the integrated ontology and inform the user about them.

4.4.2 Example of Ontology Integration from Keywords

QueryGen, apart from using the integrated knowledge to give an interpretation of the input keywords (as we will see in the following chapter), allows to access directly to the integrated ontologies and use them in different ways.

In this way, a user can be provided with an adhoc integrated ontology that comprises the concepts that s/he provided in the form of plain keywords. Moreover, QueryGen could also be used as a source ontology search engine, where the main search unit would be the *sense* of the input keywords. The returned senses contain the ontologies where they appear with the intended meaning.

In our implementation, thanks to the use of ProSÉ, we can specify further constraints on the nature of the modules to be extracted depending on the use we have in mind for the integrated ontology. Following the guidelines proposed in [JCS+08], we can choose to obtain *upper modules* (UM) or *lower-upper modules* (LUM), among others. In brief, the LUMs will contain the sub terms of the terms included in the specified signature, while the UMs will also contain their super terms. In particular, if the ontology is integrated to give a starting point to build up your own ontology, we advise to use the UMs as the modules usually contain more information. For QueryGen's purposes, we use the UMs to capture also the definitions of the terms, and therefore, be able to filter the inconsistent queries, as we will see later.

To illustrate the potential of these other uses of our ontology integration, we chose an example taken from the field of e-commerce. Let us imagine a user that has an e-book store in the Web. This user wants to have the contents on her/his page annotated semantically to allow the Web robots to index them. However, he is not an expert and does not know any ontology about e-commerce. He introduces the input keywords *book* and *offer* because he wants to start a sales campaign.

Figure 4.8 shows the intermediate results for these keywords, while Figures 4.9 and 4.10 present a visual representation of the final ontology. In our prototype, we have used a controlled set of ontologies to trace and repeat experiments. The set for this integration task contains the test collection OWLS-TC4[3] plus the ontology *schema.org*[4]. The ontology obtained for this input can be consulted at `http://sid.cps.unizar.es/ontologies/integration_book_offer.owl`.

Note that both semantic keywords share source ontologies (schema.org, for example), which is exploited by our system to obtain a richer module from them. The user can now use this ontology to add semantic annotations to her/his site, and, as the integrated ontology keeps the original sources, the robots visiting the site will understand these annotations.

[3] `http://projects.semwebcentral.org/projects/owls-tc/`, last accessed October 3, 2013.

[4] This ontology is supported by Google, Yahoo, and Microsoft, in their Web Search engines.

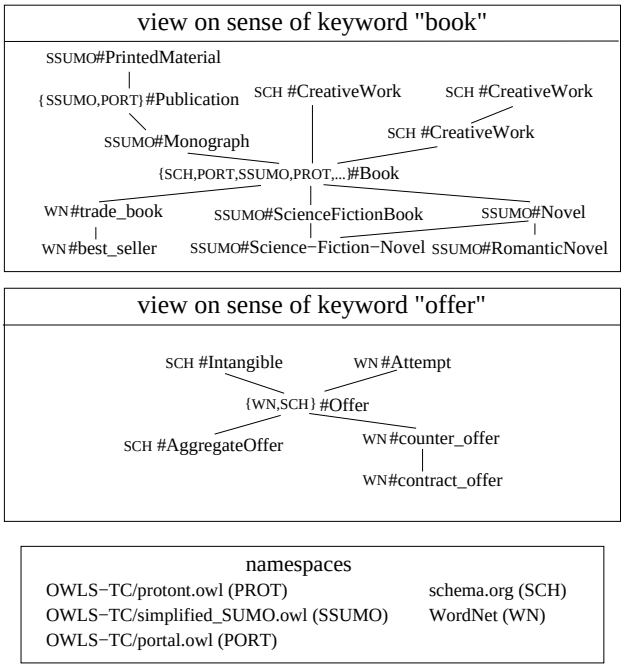


Figure 4.8: Excerpts of the senses obtained for “book” and “offer”.

4.5 Summary of the Chapter

In this chapter, we have presented the details of how QueryGen uses the disambiguation techniques presented in [TGEM07] to obtain the meaning of the input keywords, which is the first step to obtain a correct keyword interpretation. The inner architecture of the module makes it possible to speed up the different disambiguation steps (as we will see in Chapter 7). Moreover, the Multi-Ontology Senses Library presented allows to keep the meanings updated, adapting them to the changes in the source ontologies and, thus, providing an automatic evolving mechanism.

The information retrieved during the disambiguation process is used to integrate a multi-ontology sourced ontology, which contains the user’s intended meanings. This integrated information is further enriched using external sources and modularization techniques to fill the possible semantic gaps existing in the senses. Finally, apart from being used for the keyword interpretation process, these integrated ontologies can be written down and be used for different purposes, as we have seen in the presented example.

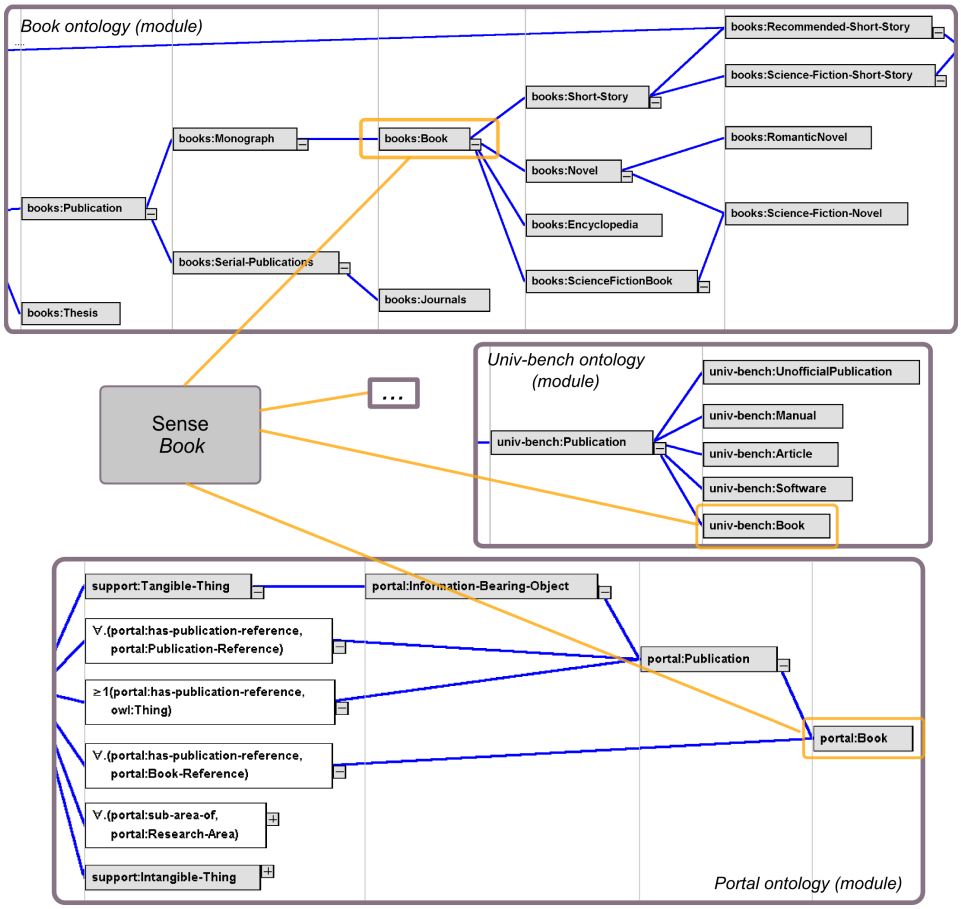


Figure 4.9: Integrated ontology for “book” and “offer” (part 1).

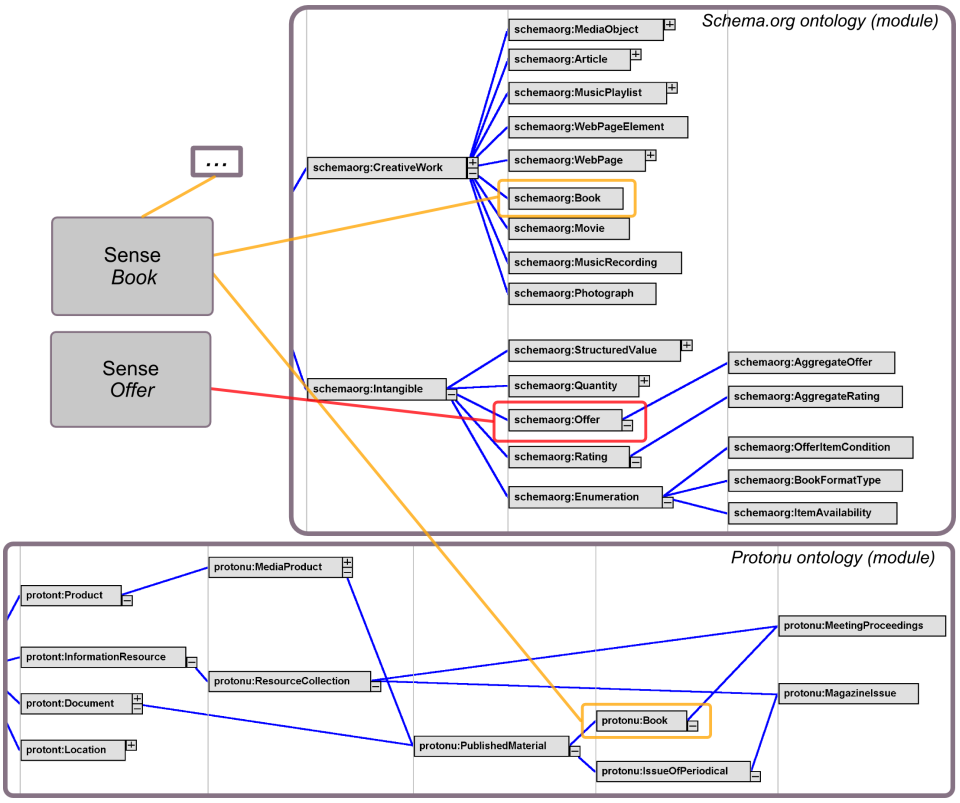


Figure 4.10: Integrated ontology for “book” and “offer” (part 2).

Chapter 5

Semantic Query Generation

In this chapter, first, we overview the main steps that the Semantic Query Generation module in QueryGen performs to give the semantically possible interpretations of the input keywords. Secondly, we focus on how query languages are specified to be used by QueryGen. This is done via special grammars, which comprise semantic information about the operands of the query language and about how they can be combined to build formal queries out from a set of input tokens. Then, we detail how QueryGen uses these specifications to build the possible queries (interpretations) in the different available query languages. After this, we explain how QueryGen filters out the queries that are semantically inconsistent and attempts to discover possible missing information in the user's input. Finally, as the search space of the possible queries is quite large, we explain the different semantic techniques QueryGen applies to reduce the candidate queries shown to the user.

5.1 Overview of the Semantic Query Generation Module

Once the meaning of each keyword has been established, QueryGen automatically builds a set of formal queries which, combining all the keywords, represent the possible semantics that could be intended by the user. The main generation steps are shown in Figure 5.1:

- Analysis Table Constructor: It constructs the analysis tables for the formal query languages that the generator uses to generate the possible queries. This is done off-line and just once for each language made available to our system.

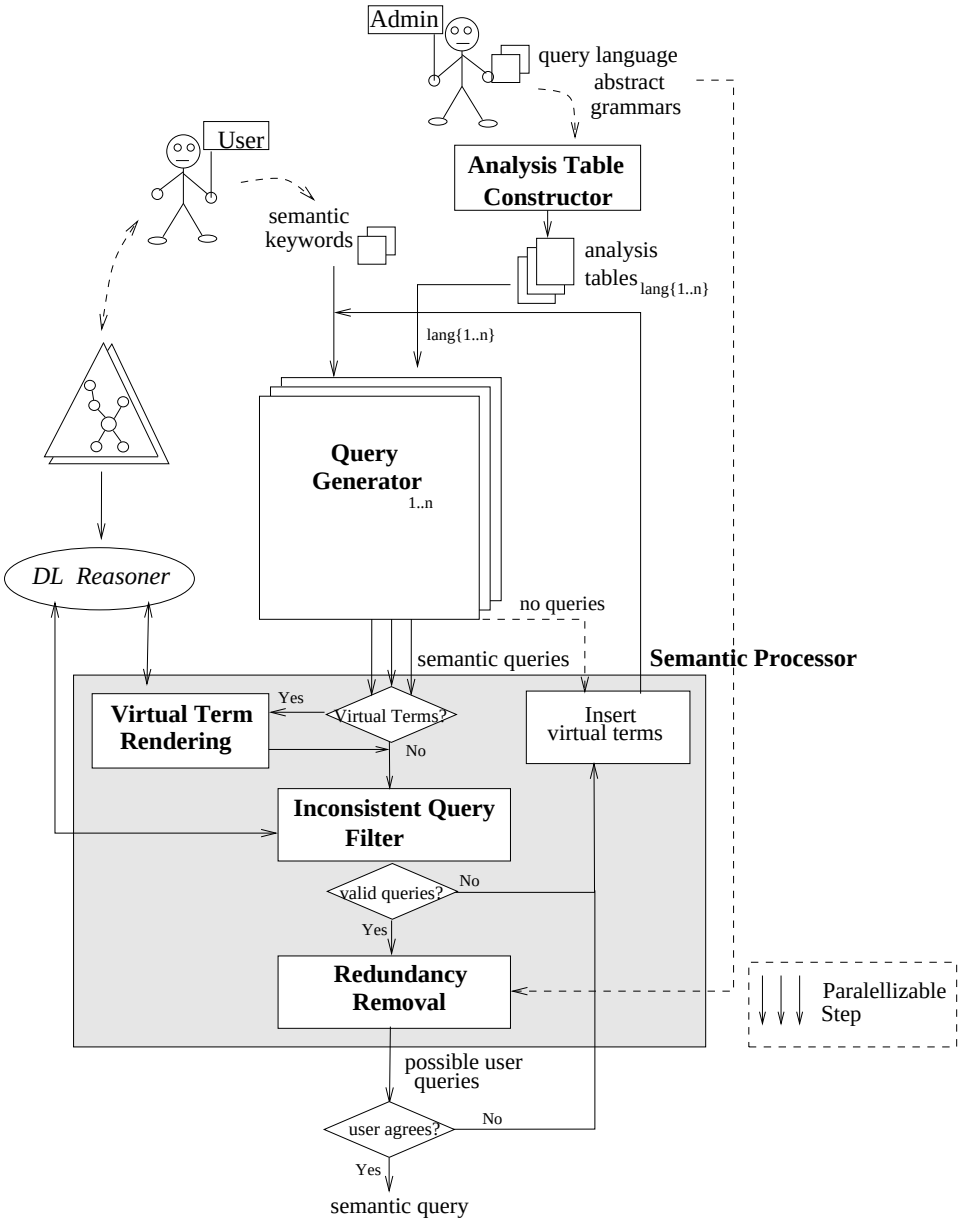


Figure 5.1: Multi-language query generation process.

- Query Generator: It builds the possible queries for each query language according to its syntax. During the generation process, Query-

Gen takes into account the semantics of the different operators to avoid generating semantically equivalent queries.

- Semantic Processor: Once the set of syntactically possible queries is obtained, the system is able to filter out the inconsistent ones with the help of a DL reasoner. During this step, it also performs a semantic enrichment to try to find possible implicit keywords that have been omitted due to the simplistic nature of the keyword query model. To do so, our system adds *virtual terms (VTs)* to the input. They are generic typed terms (they can be generic concepts, roles, or instances) that represent the keywords that the user might have omitted, but without whom the intended query cannot be built. Finally, when the system uses VTs, an extra step is carried out to substitute them with appropriate terms taken from the ontologies which the input keywords were mapped to.

In the following sections, we detail these three main query generation steps plus a section dedicated to the semantic reduction techniques that are used to ease the selection of the intended query.

5.2 Analysis Table Constructor

Our system has to be provided with a specification of the different formal query languages that it will use to express the semantics behind the user keywords. In our approach, the query languages associated to the query models that the underlying systems support are specified using extended context-free grammars. These grammars have their "syntax sugar" removed, and are semantically annotated, on the one hand, to avoid generating duplicated queries; and, on the other, to build the expressions that have to be evaluated to conclude if a query is consistent or not according to the knowledge retrieved by the system. Moreover, instead of working with bare syntactical tokens, we consider three types of tokens: *Concept (C)*, *Role (R)*, and *Instance (I)*, which correspond to the three main types of elements in ontologies. With these grammars, the system builds the analysis tables[1] that are used by the Query Generator (see Section 5.3) to build all the possible queries corresponding to the input keywords for each of the available query languages. Note that these tables are built only once for each new output query language that is made available to the system, and they are used every time a new query is posed to the system.

[1]It builds the Goto and Action tables, as defined in [ALSU07].

5.2.1 Specifying the Query Languages

So, to make a new query language available to our system, its context-free grammar G must be transformed into an abstract context-free grammar $\mathcal{G}'$, where the syntax sugar of such a language has been removed. In this grammar, operators become non terminals, and the right side of their productions are the operands they accept. Thus, the use of these abstract grammars makes the translation process independent of the syntax of the query languages. We define these abstract grammars as tuples $\mathcal{G}' =< \mathcal{Q}, \mathcal{N}, \mathcal{T}, \mathcal{P} >$, where:

- Q is the starting symbol of the grammar, and represents the root of the query.
- $N = \{Op_i\} \cup \{Q_{ps}\} \cup \{R_{types}\}$, with $\{Op_i\}$ containing the set of operators of the query language; $\{Q_{ps}\}$ being a set of auxiliary nonterminals needed to build up the different parts of the queries; and $\{R_{types}\}$ being the types of the returning values of the query language operators. Each element Op_i is a tuple $< Op_{ID}, \{prop_j\} >$, with the id of the operator and its associated properties.
- T is the set of terminals that we work with, and is conformed by C, R, and I (corresponding to concept, role, and instance tokens, respectively), plus the empty token symbol ξ.
- $P = \{< prod_i, localCond_i, globalCond_i >\}$ is the set of productions which define: a) if the left-side nonterminal is an operator, an ordered list of the types of operands it works with, and b) if it is a returning type ($\{R_{types}\}$), the operators that produce the returning values of that type; $localCond_i$ and $globalCond_i$ are expressions which define the semantic conditions that have to be checked to correctly apply the operator.

We define them to be ambiguous on purpose, as we want to obtain all the possible derivations for a given input. To do so, we have modified the classical LR parsing algorithm [ALSU07] to deal with conflicts, as we will see later. These grammars makes the system able to, once it knows whether the input keywords are concepts, roles or instances, build semantically correct interpretations expressed as formal queries in the different query languages that are available. In Figure 5.2, the extended abstract grammar corre-

sponding to a subset of BACK[2] query language [Pel91] is shown.

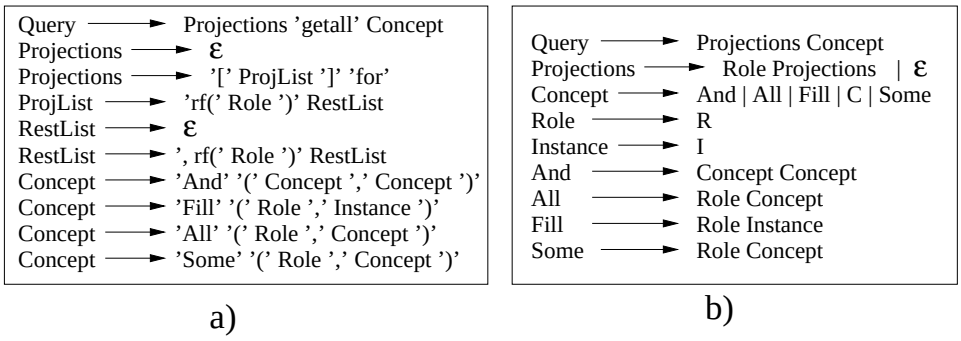


Figure 5.2: a) Simple BACK grammar, and b) the resulting abstract grammar (semantic annotations are not included).

In particular, in the BACK example:

- The initial symbol Q is *Query*.
- $\{Op_i\}$ contains *Projections, And, All, Fill,* and *Some* nonterminals; $\{Q_{ps}\}$ would be empty; and, $\{R_{types}\}$ contains *Concept, Role,* and *Instance* nonterminals.
- T, as we have defined before, contains *C, R, I* and ξ tokens.
- P contains each one of the productions in Figure 5.2.b.

In the rest of the section, we explain the rest of the elements that complete the language definition, namely, the properties of the operators, and the semantic annotations for the productions.

Properties of the Operators

As we have seen in the description of the extended abstract grammars used in our system, there is a list of its properties along with each of the operators. In particular, the properties considered are *associativity*, *involution*, *symmetry*, *restrictiveness* and *inclusiveness*. The first three ones are well known properties, while *restrictiveness* and *inclusiveness* are defined in [BTMI10] as follows:

[2]Although it is obsolete (discontinued since 1998), we use BACK in the examples for didactic purposes as it almost lacks syntax sugar, and supports projections.

Definition 5.2.1 *A binary operator* op *is* restrictive *if*
$\exists f : K \rightarrow C,\ K = \{R, C\} \mid f(x) \sqsubseteq y \Rightarrow op(x, y) \equiv op(x, f(x))$.

Definition 5.2.2 *A binary operator* op *is* inclusive *if*
$\exists f :\ K\ \rightarrow\ C,\ K = \{R, C\} \mid f(x) \sqsupseteq y \Rightarrow op(x, y) \equiv op(x, f(x))$.

From these definitions, and according to the semantics of the operators in BACK, it directly follows that the *And*, *Some*, and *All* operators are restrictive, and *Or* is inclusive.

Following with the specification of the excerpt of BACK language, a summary of the properties of the considered operators is shown in Table 5.1.

Operator	**Properties**
And	associativity, symmetry, restrictiveness
Some	restrictiveness
All	restrictiveness
Fill	none
Projections	associativity, symmetry

Table 5.1: Properties of the operators of BACK language.

In this example, we consider the *Projection* operator as associative as all the roles that are specified in the projections list are applied to the same concept. Thus, it does not matter the order in which we apply them. The same reasoning is applied to consider it symmetric.

In the following sections, we will see how our system takes into account these properties to avoid generating duplicated queries in different steps of the interpretation process (generation and enrichment steps).

Expressions for Semantic Checking

Each production in the grammar can be annotated with semantic expressions that are checked with the help of a DL reasoner. These expressions are built using the operators shown in Table 5.2. There are two types of conditions for each production, *local* and *global* ones, depending on the information that they comprise:

- A local condition $locCond_i$ on a production $prod_i$ details semantic constraints that the nonterminals on the right side of the production must satisfy for the production being eligible to be fired. This is used to perform an extended semantic type checking on the operands locally.

Operator	Meaning
$*	It refers the *-th element of the right side of the production
SubClassOf (cpt_1, cpt_2)	It checks whether cpt_1 is subsumed by cpt_2
SubPropOf ($role_1$, $role_2$)	It checks whether $role_1$ is subproperty of $role_2$
Dom ($role$)	It returns the domain of $role$
Range ($role$)	It returns the range of $role$
Class ($inst$)	It returns the class of $inst$
InstanceOf ($inst$, cpt)	It checks whether $inst$ is instance of cpt
And (cpt_1, cpt_2)	It returns the concept intersection
Or (cpt_1, cpt_2)	It returns the concept union
Satisfiable (cpt)	It checks the satisfiability of cpt
$\wedge$ ($bool_1$, $bool_2$)	It calculates the boolean And of $bool_1$ and $bool_2$
$\vee$ ($bool_1$, $bool_2$)	It calculates the boolean Or of $bool_1$ and $bool_2$
$\neg$ ($bool$)	It calculates the boolean Not of $bool$
Thing	It returns the Top concept, which is the identity element for And operator
Nothing	It returns the Bottom concept, which is the identity element for Or operator
Neutral	It lets the system choose which identity element to use depending on the context

Table 5.2: Operators of the inner specification language to establish what has to be checked about the operators of the final specified language.

- A global condition $globalCond_i$ on a production $prod_i$ provides a semantic translation of the production that allows to translate it into a checkable DL expression (even with non-DL query languages). Using these conditions, QueryGen can build a global DL expression that comprises the semantics of the associated query, and check its consistency according to the retrieved knowledge.

Following with the simplified BACK example, there are no local conditions on the different productions due to the fact that the operators do not impose any special constraint on their operands (apart from their general type -concept, role or instance- which is explicitly stated in the right part of the production). Regarding global conditions, there are several annotated productions, as we can see in Table 5.3.

The first condition tells the system to check the conjunction of the concepts returned by the *Projections* operator and the rest of the query (*Concept*). The concept returned by the *Projections* nonterminal is built according to the second condition: It appends the domains of the different roles

Production	Global Condition
Query → Projections Concept	And($1, $2)
Projections → Role Projections	And(Dom($1), $2)
Projections → ξ	Thing
Concept → And	$1
Concept → All	$1
Concept → Fill	$1
Concept → Some	$1
Concept → C	$1
Role → R	$1
Instance → I	$1

Table 5.3: Global conditions on the productions of the abstract grammar for BACK language.

using *And* operators. With these two global conditions, our system can check the global consistency of the query. This is done by checking whether the domains of all the properties that appear in the *Projections* part of the query are compatible (not disjoint) with the rest of the query, which defines the objects retrieved (the ones which the projections have to be made on).

The rest of the global conditions just tell the system not to touch anything about the returned concepts. Note that there are no global conditions for the productions with the *And, Some, All,* and *Fill* operators because they directly map to DL-expressions and their associated *Concept* can be built directly without any given expression.

Although this example does not contain local conditions, the use case of LOQOMOTION [IMI06] that will be presented in Chapter 6 uses them thoroughly. However, to illustrate the role of local conditions, let us take an operator that calculates percentages of instances between concepts (e.g., percentage of male people). It could be annotated with the expression $SubClassOf(\$2, \$1)$ to express that, to be a candidate operator for two concepts, the second operand has to be subclass of the first one. In this example, to apply the operator that calculates this percentage to the concepts *person* and *male* (e.g., $percentage(person, male)$), our system would check $SubClassOf(male, person)$, i.e., whether *male* is a subclass of *person*. Otherwise, we could be mixing instances of different types, and, thus, applying the operator in a wrong way. Another example would be to force a particular instance/value of an operator to belong to a particular concept, e.g., an *Inside* operator that would need an instance of *location*.

With this semantic annotations, our system is able to generate only both syntactically and semantically correct queries, as we will see in the following section.

5.3 Query Generator

For each query language available, a different generation thread is launched to build the possible queries expressed in such a language. As an example, let us assume that a user enters keywords "person fish" to find information about people devoured by fishes, which are mapped to the homonym terms in the ontology *Animals*[3], and that the previous simplified[4] version of BACK is used as target query language. The query generation process is divided into three main steps, as shown in Figure 5.3.

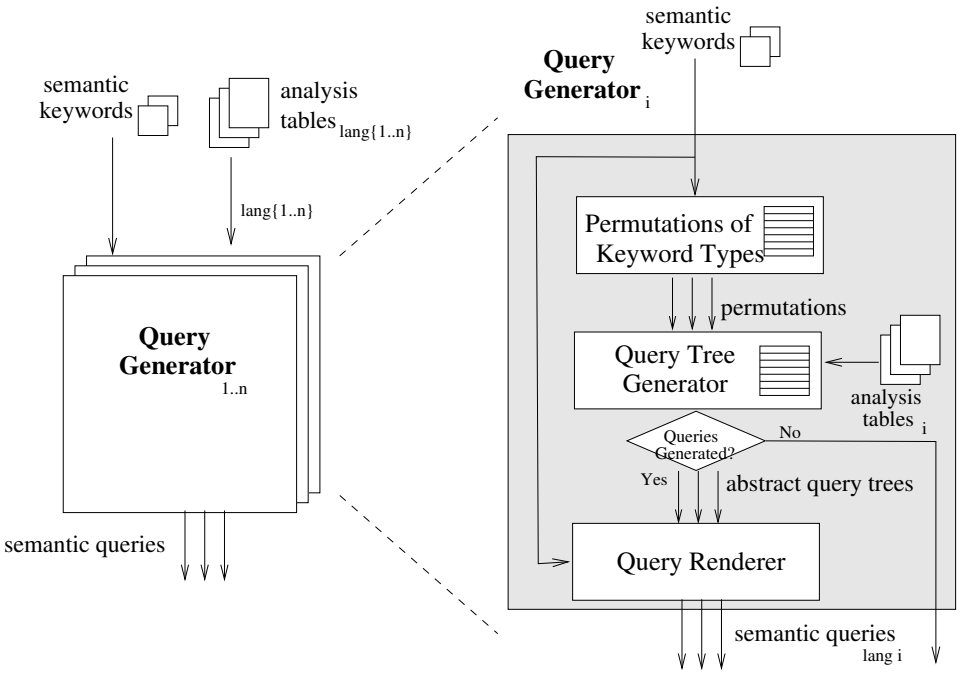


Figure 5.3: Inner structure of the Query Generator.

[3] `http://www.cs.man.ac.uk/~rector/tutorials/Biomedical-Tutorial/Tutorial-Ontologies/Animals/Animals-tutorial-complete.owl`, last accessed October 3, 2013.

[4] The simplified version of BACK consists of the projection operation and four concept defining operators: *And, Some, All,* and *Fill*.

5.3.1 Permutation of Keyword Types

To decouple the generation process of any specific language, its first stage is syntax-based. So, the system firstly obtains all the possible permutations of the types of term (concept, role, or instance) corresponding to the semantics of each input keyword, to discover any syntactically possible query in latter steps. This allows the system to focus on the kind of knowledge represented by the user keywords (which will enable it to perform structural analysis of the generated queries in a schema-agnostic way), and to be independent of the specific order in which they were entered by the users.

For the set of semantic keywords $K = \{k_i\}$, we define $KT = \{t_i\}$, where t_i is the type of k_i (C if it is a concept, R if it is a role, and I if it is an instance). This step calculates all the permutations $KP{=}P_{|KT|}$ of the set KT associated to the list of input semantic keywords. In the example, *Person* and *Fish* are concepts, so the output of this step would be <C,C>, as no more permutations are possible. The results of this step are shared by all the language threads.

5.3.2 Generation of Abstract Query Trees

For each permutation $p \in KP$ obtained in the previous step, the system generates all the syntactically possible combinations according to the syntax of the available query languages. We call these combinations *abstract queries* because they have *gaps* that will be filled later with specific concepts, roles, or instances. These abstract queries are represented as *abstract query trees*, this is, syntax trees where the inner nodes are operators and the leaves are *typed gaps* (concept, role, or instance gaps).

Let us denote by $\mathcal{T}_h^p$, the set of abstract query trees with maximum height h where the terminals C, R, I appear in preorder in the abstract query tree in the same order as in permutation p; h is used to limit the complexity of the generated queries[5]. This set is built using a modified parser that follows *all* the possible paths in the recognition process, performing a bottom-up analysis LR [ALSU07]. Algorithm 2 shows the modifications that we have been performed to the general LR-parsing algorithm. Instead of forcing the ACTION table to have just one action in each cell (as in the classical

[5]In practice, we have never had to put a strict height limit due to the fact that, so far, the languages that we have considered have operators that consume operands. This is, being $L \rightarrow R$ a production describing how to apply a generic operand, we have that $|L| <= |R|$ always holds. When we have that $|L| = |R|$, we apply the involution property of the operator, if applicable, to limit the query height, and, in the last resort, we limit by this h parameter.

algorithm), our system deals with the possible action conflicts by spanning the search space recursively for each possibility.

Algorithm 2 LR Modified parsing algorithm

```
1: procedure generateQueries (List remainingInput, Stack stateStack)
2: // the initial values for the arguments:
3: // remainingInput = input
4: // stateStack contains only s0
5: let currentTerm be the first symbol of remainingInput
6: while true do
7:    // repeat forever, one of the possible actions is to exit
8:    let currentState be the state on top of stateStack
9:    if ACTION[currentState,currentTerminal].size = 1) then
10:      // the algorithm is the usual one
11:      execute the corresponding action
12:   else
13:      for all act in ACTION[currentState,currentTerminal] do
14:         clone remainingInput as remainingInputNew
15:         clone stateStack as stateStackNew
16:         execute act on cloned variables
17:         generateQueries(remainingInputNew, stateStackNew)
18:      end for
19:   end if
20: end while
21: end procedure
```

Following the previous example, with the input <C,C> and simplified BACK as query language, *And(C,C)* would be built as an abstract query. Depending on the expressivity of each query language (i.e., their operators), the system is able to generate different query trees. For example, in Figure 5.4.b we can see two different query trees generated for permutation <R,C,C> using Relational Algebra (RA) as query language; these trees represent queries "$\Pi_R(C \cap C)$" and "$C \bowtie_R C$", respectively.

During the generation of the query trees, the semantics of the operators are considered to avoid generating equivalent query trees, as it will be explained in Section 5.5.1.1, which leads to an important reduction of the query search space.

5.3.3 Query Rendering

During this step, concurrently for each abstract query tree $t \in \mathcal{T}_h^p$, the system creates a set of query trees $\mathcal{Q}^t$. Each $q \in \mathcal{Q}^t$ is the result of using each user keyword k to replace a gap g in t, $g \in \{C, R, I\}$, verifying that the type of k is g. The result of this step for the running example would be *And(Person, Fish)*, i.e., entities which are a person and a fish, which in this

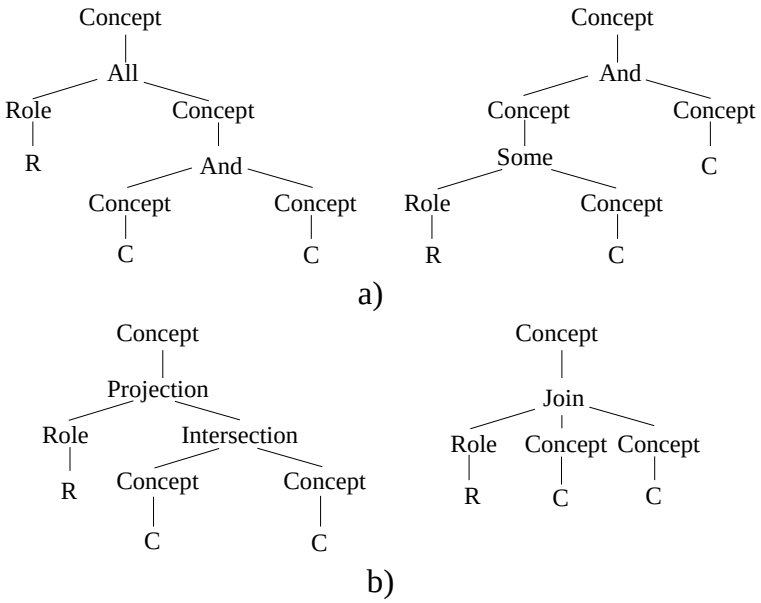


Figure 5.4: Sample abstract query trees for permutation <R,C,C>: a) in BACK, and b) in Relational Algebra (RA).

case we know that do not represent what the user had in mind although, for the system, it is a syntactically possible query to consider.

The decision of introducing an intermediate generation step (abstract query trees) instead of working with the actual keywords from the beginning allows our system to apply the reduction techniques that will be presented in Section 5.5 more efficiently. Intuitively, an abstract query tree is a canonical representation that will generate a set of queries after this rendering step; so, the semantics of the different operators are considered to avoid generating semantically equivalent trees (in particular, our system considers the properties specified in the extended grammar of each of the languages). This leads to a reduction of the number of possible abstract queries and, consequently, to a more important reduction in the number of final possible queries.

Thus, at the end of this step, our system has built all the syntactically possible queries in each of the languages. However, being syntactically correct does not imply that all of them are semantically consistent. In the following section, we present the techniques that our system applies to: a) detect and filter the queries that are not semantically consistent according to

the retrieved knowledge, and b) when no query satisfies the user (or no query has been generated due to an incomplete input), enrich the input semantically to fill the gap between the user's input and her/his information need.

5.4 Semantic Processor

Once the system has obtained all the *syntactically* possible queries, the Semantic Processor comes into play. As aforementioned, it has two main tasks: To check the queries semantically according to the available knowledge; and to perform a semantic enrichment of the input to suggest further interpretations when the intended query cannot be found by the user (e.g., due to an incomplete input). In the rest of the section, we detail these processes.

5.4.1 Inconsistent Query Filtering

During the previous steps, all the user keywords have been combined into queries that are syntactically correct according to the different available query languages. However, some of these queries might not be semantically correct according to the semantics of keywords. Fortunately, we have their semantic information integrated altogether in a local ontology.

Our system takes advantage of capabilities of DL reasoners [BCM+03] to, once the available knowledge has been classified, detect the queries that are inconsistent according to it by testing their satisfiability. In mathematical logic, a formula is satisfiable if it is possible to find an interpretation (model) that makes the formula true [BJ07]. Intuitively, and applied to DLs, a DL expression is satisfiable iff the concept that it defines can have instances. In our example, *And(Person, Fish)* would be removed in this step as it is classified as being inconsistent (*Person* and *Fish* are defined as disjoint classes in ontology *Animals*), and consequently will not lead to any result.

This consistency evaluation is direct when dealing with DL languages as they can be directly translated into concepts and the reasoner can be asked about their consistency. However, when it comes to non-DL languages, we have to tell the system how to check them via the specification of the language. As seen in Section 5.2.1, there are two types of conditions associated to each of the productions, *local* and *global ones*:

- Local conditions: They provide semantic checkings that have to be performed on the operands of the production. A query must hold all the local conditions constraints; otherwise, it must be filtered out as

inconsistent one because there would be any production that should not have been fired. In Figure 5.5, an example of a *percentage* operator is shown.

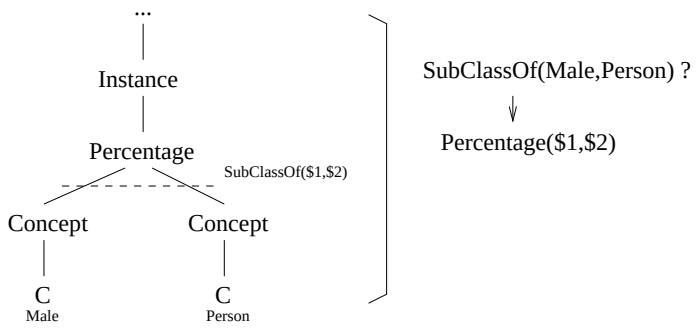


Figure 5.5: Example of semantic checking on the local conditions of a non-DL operator (only local conditions are shown).

The definition of this operator tells the system that their operands must hold that the first one is subclass of the second one to be applicable. In this case, if *Male* was a subclass of *Person* then this part of the query would be locally consistent. All the nodes of a query have to be locally consistent for the query to be considered for global consistency. Note that, otherwise, there would be a part of the query that had been built incorrectly.

- Global conditions: A query holding all the local conditions only probes that it is syntactically correct (by construction), and that all the operators have been correctly applied. However, the query might still be inconsistent due to its whole meaning. As mentioned before, the global meaning is obtained easily for DL-languages, as queries can be seen as concepts and therefore, directly translated and semantically checked. However, for non-DL languages (e.g., SQL-like languages) or for extensions of DL-languages (e.g., the projection operator of the simplified BACK language), this is not directly applicable. Global conditions provide a translation of each of the productions of the language to build a semantic expression that comprise the global meaning of the query.

  Our system translates the query into a checkable DL-expression by traversing recursively its associated query tree and applying the global conditions expressions to each node. This traversal is performed in a depth-first way. During it, our system applies the different condition

operators with the help of a DL-reasoner. Following with the example in the query generation, if we added the keyword *owns* to the input, mapped to the homonym term in ontology *Animals*, the system could form the query *[owns](And(Person, Fish))*, that is, the entities that are owned by a *(Person and Fish)*. In Figure 5.6, an example of the global checking on this query involving a projection is shown.

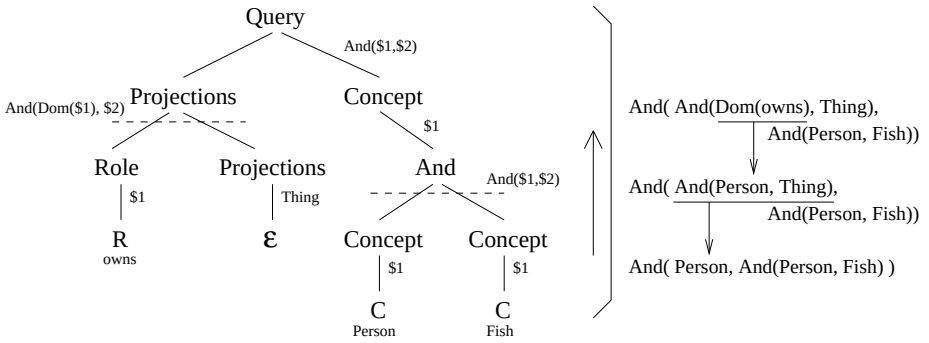


Figure 5.6: Example of the global semantic checking on a DL-query involving projections (only global conditions are shown).

Going from the leafs to the root node, our system is able to form the global expression applying the global conditions on the productions. In particular, the conditions on *Projections* operator establishes that, to be able to ask for the value of a property for the instances of a particular instance, the domain of the property must be *compatible* with the concept (i.e., not disjoint, which can be checked out by evaluating their conjunction). So, the system applies the specification to translate the query tree into *And(Dom(owns), And(Person, Fish))*. Resolving the different operators with the use of the DL reasoner, this expression leads to *And(Person, And(Person, Fish))*, which is inconsistent as we cannot ask for the properties of a concept that is not satisfiable.

Note that all these checks cannot be performed before: Until the *rendering step*, the system is working just taking into account the structure of the queries. It is not until the system substitutes the typed gaps on the abstract queries with the input terms, that the actual query is built (along with its meaning). Due to the size of the query search space, we prioritized its reduction. Removing firstly all the possible abstract queries (each of which results on a set of actual queries after the *query rendering* step)

pruned the search space and lead to a lower number of queries to be checked than working with the actual terms from the beginning.

Finally, the performance of this step is greatly boosted by the fact that the set of generated queries forms a *conservative extension* [CHKS08] of the original ontologies. Once an ontology has been classified, this property makes it possible to evaluate the satisfiability of the queries without reclassifying the ontology, as each query does not assert new knowledge into that ontology.

5.4.2 Semantic Enrichment

When no query either is generated or satisfies the user, our system considers that something could be implicit in the user input. The average number of keywords used in keyword-based search engines "is somewhere between 2 and 3" [MRS08], so there is a high chance that the user might have simplified too much its information need, specially when it comes to expressing complex queries.

To deal with this lack of information, our system adds *virtual terms* (*VTs*) to the original list of user keywords (*Insert Virtual Terms* step in Figure 5.1). These VTs represent possible keywords that the user may have omitted as part of her/his query, as a keyword query is a simplification of her/his actual information need. Then, the previous steps are executed again to generate queries considering these VTs. In our example, the extended inputs considered would be "person fish $VT_{concept}$" and "person fish VT_{role}", which allows the system to build, among others, the enriched abstract query *And(Person (Some(VT_{role}, Fish))*[6] (see Figure 5.7). This process is slightly different to *query expansion* [CR12], as our system works at structural level, aiming at building the exact query, instead of broadening/narrowing the search itself by adding actual keywords to the input.

These queries with VTs have to be rendered again (*Virtual Term Rendering* step in Figure 5.1). Our system replaces any existing VT by compatible terms (i.e., terms of the same type: concept, role, or instance) extracted from the ontologies which the input keywords were mapped to in the disambiguation process (see Section 4.1). To build only semantically correct queries in an efficient way, the system narrows the set of candidate terms by using the ontology modularization techniques described in [JCS+08]. In the example, the previous enriched abstract query is rendered into *And(Person (Some (is_eaten_by, Fish))*, which actually represents

[6]Here, VT_{role} is the VT to be rendered with a compatible role.

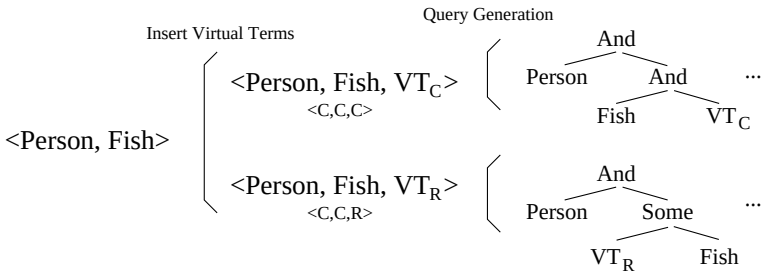


Figure 5.7: Our system enriches the input set of keywords to generate underspecified queries.

the exact semantics intended by the user when s/he entered the keywords "person fish". This query enrichment process can be repeated iteratively by adding new VTs in order to deal with situations where the user did not enter a very descriptive set of keywords.

Note that how, in the example, our system considered only VTs for concepts and roles. At first, we did not considered instances for the semantic enrichment as ontologies themselves usually do not have them (it happens frequently [WPH06]). Moreover, it seemed pretty safe to assume that when a user is looking for information about something very specific, the implicit keywords might be roles or concepts, rather than instances. For example, a user looking for information about terror movies (a possible query might be *And(Movie, Fill(genre,terror))*, will input "horror movies" instead of "movies genre". However, our approach can effectively deal with instances as well, but we advocate for asking the user to explicitly fill the instance value when appropriate instead of showing her/him all the possibilities (which might be even unfeasible, for instance, when we are dealing with datatypes such as strings or integers).

Anyway, a general keyword interpretation process that involves filling a semantic gap will generate many queries. Even when the meaning of each keyword has been perfectly established (and thus, its polysemy avoided), the number of possible interpretations will grow as the number of user keywords increases and the output query language allows to combine them in more ways. Besides, we do not want to limit our approach to provide the most popular results (syntactic search engines do that already), so we aim at not missing any possible interpretation (according to the accessible knowledge). We are aware that this may lead to the generation of a high number of queries (as there are many possible interpretations), and in the following section we propose different semantic techniques to deal with this problem.

5.5 Reduction Techniques

As stated in the previous section, the search space for the possible interpretations of a keyword query into a formal language grows quickly with the number of input keywords and the expressivity of the language. Fortunately, we can reduce this search space by applying several semantic techniques that allows our system to provide the user with an easier-to-handle set of possible interpretations.

5.5.1 Avoiding the Generation of Redundant Queries

Trying to guess the user's information need is a hard task due to the high number of possible interpretations. In our example, for the input "person fish" and one extra VT, there exist 780 syntactically possible queries for simplified BACK and 2832 for simplified DIG[7]. So, it is critical to reduce the number of generated queries. Apart from the number of input keywords, there are two main elements that lead to this high number of possible queries: The expressivity of the output query language, and the semantic enrichment step with VTs performed to fill the semantic gap.

On the one hand, expressive query languages are very valuable, as they are more likely to be able to represent the user's intended query. However, the higher the number of operators, the higher the number of possible queries (the operands can be combined in more different ways). On the other hand, adding new terms to the user's input can help to discover the intention of the user. However, this will also increase the number of possible queries, mainly for two reasons: 1) new possible interpretations appear, and 2) there may be a high number of candidates to replace a given VT (some of them probably irrelevant).

Along this section, we show how our system deals with these two issues. Then, in Section 5.5.2, we present a semantic technique that is applied to further simplify the output of the query generation.

5.5.1.1 Considering the Expressivity of the Query Language

For expressivity of a query language, we understand its set of operators and the possible ways in which they can be combined to form a proper query. The more operators the language has and the more ways to combine them exist, the more queries will be possible. For example, if you add the *Or* operator to a language that had the *And* operator, the number of possible

[7]Simplified DIG is equivalent to simplified BACK plus the *Or* operator.

queries for a user input considering both operators will be larger than the double. There is apparently no way to reduce this number because the different options express different queries. However, we can avoid building equivalent queries along the generation process by considering the semantic properties of the operators. In particular:

- *associativity*: It is used by the *Query Tree Generator* to avoid, for example, building *And(And(c1, c2),c3)* if it has already generated *And(c1,And(c2, c3))*.
- *involution*: It is used by the *Query Tree Generator*, for example, to avoid building $Not(Not(c))$, which is equal to c.
- *symmetry*: It is used by the *Query Renderer* to avoid, for example, building *And(c2,c1)* if it has already generated *And(c1,c2)*.

Apart from those well-known properties, we saw in Section 5.2.1 that our system consider two other properties of the operators: *Restrictiveness* and *inclusiveness*. These properties are used in the *Virtual Term Rendering* step to avoid substituting the VTs by terms that would result in equivalent queries. For example, for the *restrictiveness*, if we have $And(c1, concept)$, all the candidates that subsume c1 can be avoided as *(any of them* And *c1)* would result in $c1$.

Following with the running example, let us suppose that a user enters "person fish" to find information about people devoured by fishes. Adding one VT to that input, the system generates 780 queries using simplified BACK and 2832 queries using simplified DIG, which are reduced to 72 queries ($90,77\%$ reduction) and 364 ($87,15\%$ reduction), respectively, by considering the semantics of the operators. In both cases the intended query *Person And Some(is_eaten_by, Fish))* is among the final results.

5.5.1.2 Reducing the Number of Candidate Terms for Query Enrichment

Usually, users simplify the expression of their information needs when they write keyword queries, which contributes to the semantic gap. Thus, a user may omit terms that form part of the actual information need. As we have seen before, to deal with the problem of possible information loss, some VTs can be added to the input, in order to generate the possible queries as if these VTs were proper terms introduced by the user. Then, the system performs a substitution of those VTs with actual terms extracted from the ontologies considered.

Following this idea, one can think about using each term of the same type as the inserted virtual one. For example, if the VT was a concept, the system could substitute it with all the concepts of the input ontologies (and generate one query for each different candidate concept). However, as the number of queries with VTs could also be high, considering all the terms of the input ontologies to render each VT for each query is too expensive.

In order to reduce the number of candidates for rendering a VT while avoiding losing any possible related term, we apply the modularization and re-using techniques explained in [JCS+08]. More specifically, the system uses ProSÉ, which, given a set of terms of an ontology (*signature*), makes it possible to extract different *modules* that can be used for different purposes. Our system uses the user input terms as signature and extracts a module such that the same information can be inferred from the extracted module as from the whole ontology, as shown in [JCS+08]. This allows the discovery process to focus only on what is related to the input terms.

After applying the modularization techniques for the user query "person fish", the system generates 32 queries using BACK (15 after filtering, 98.07% less than the 780 original possible ones), and 148 queries using DIG (73 after filtering, 97.42% less than the 2832 original possible ones). The intended query is still among the final results, as the system does not miss any possible interpretation. Note that, in the example, we have used a single ontology to depict the impact of the technique. However, the modularization is already applied during the integration of the local ontology. Anyway, applying this technique again, with different parameters for ProSÉ, might lead to a further narrowing of the search space.

5.5.2 Extraction of Relevant Query Patterns

Besides reducing the number of generated queries, the way in which they are presented to the user also makes a difference. Users' attention is a capital resource and they can get easily tired if they are forced to browse over too many options to select their intended query.

In this stage of the search process, recall seems to be crucial as only one interpretation will fit the user's information need. Ranking the generated queries according to their probability of reflecting the user's interest can be an approach to minimize the interaction with the user. However, it is not clear how to identify the query that a specific user had in mind when writing a set of keywords, even though the meanings of the individual keywords in the input have been identified previously. For example, with the input "person fish" the user might be looking for information about people devoured by

fishes, but also about people who work with fishes (e.g., biologists, fishermen, etc.), among other interpretations. Besides, a statistics-based approach may be not suitable for ranking queries, as it would hide possible interpretations that are no popular in the community. Approaches based on semantic and graph distances would also hide possible meanings.

Due to these reasons, we advocate a different and orthogonal approach to ranking. Our approach tries to minimize the amount of information that will be presented to the user by identifying query patterns, and could be combined with any query ranking approach if it is available. The semantic technique that we propose takes advantage of the syntactic similarity of the generated queries and of the ontological classification of the terms that compose the queries. A small example is shown in Table 5.4, where we can see that several queries may have a similar syntactic structure.

Queries	**Patterns**
All(drives, Person) And bus	All(Role, Concept) And Concept
All(drives, Bus) And person	
...	
Some(drives, Person) And bus	Some(Role, Concept) And Concept
Some(drives, Bus) And person	
...	
All(drives, Bus And Person)	All(Role, Concept And Concept)
...	
Some(drives, Bus And Person)	Some(Role, Concept And Concept)
...	

Table 5.4: Queries and patterns generated for "person bus"[8]using BACK.

Thus, the system analyzes the structure of the queries to extract common query patterns which lead to a compact representation of the queries. This is especially useful when the system tries to find out the user's intention by adding VTs. A query pattern shows an expression with the VTs not substituted (i.e., with *gaps*) and the system maintains a list of potential candidates for each gap.

At this point, a new challenge arises: How can the candidates for a gap be organized to facilitate their selection by the user? We advocate the use of a DL reasoner to show the candidates and allow users to navigate through their taxonomy. The interface shows a list of candidate terms and three buttons for each gap in each pattern (see Figure 5.8):

[8]Mapped to homonym terms in *People+Pets*, `http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/people+pets.owl.rdf`, last accessed October 3, 2013.

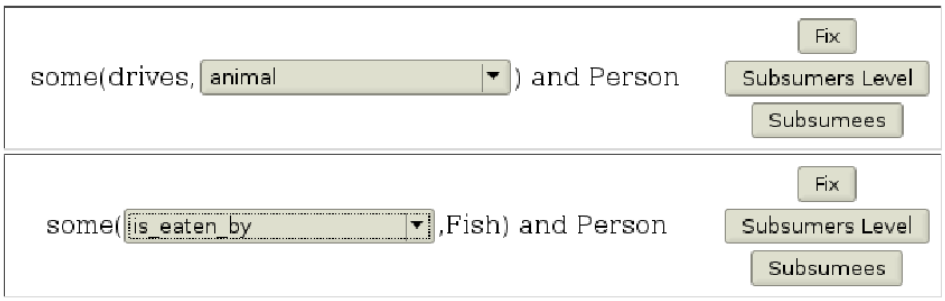


Figure 5.8: Example of query patterns for "person drives" and "person fish".

- *Fix*: Performs the substitution with the candidate term selected.
- *Subsumers*: Enables the user to generalize the candidate term selected.
- *Subsumees*: Enables the user to refine the candidate term selected.

This allows to show a high number of queries in a really compact way and provide a navigation in a top-down style through the terms of the ontology. Thus, the most general terms are initially presented to the user, who can then move through the taxonomy by accessing each time a direct subsumers/subsumees level. Users are allowed to select only terms that are relevant for the corresponding gap, i.e., their selections will never lead to an inconsistent query.

Thus, for example, for the input "person fish", the system is able to show the 15 final queries obtained using BACK under 7 patterns, and the 73 obtained using DIG under 20 patterns. Moreover, this representation allows the system to establish an upper bound on the number of user clicks needed to select their query, which is equal to the depth of the taxonomy of the ontology multiplied by the number of substitutions to be performed (number of gaps).

5.6 Summary of the Chapter

In this chapter, we have presented the core of our proposal for interpreting keywords in different query languages. First, we have detailed how we can specify different query languages through the use of specially annotated grammars, which comprise the semantics of the different operators of the language. These grammars lack syntax completely, and allow QueryGen to be completely syntax agnostic. Moreover, they allow to semantically

describe the language even when it is not a DL query language, thanks to the use of local and global conditions.

Then, we have detailed how QueryGen, with the help of the provided grammars, is able to generate different queries that provide the different possible interpretations of the input keywords. This generation is guided both by the semantics of the input keywords and the semantics of the operators. Our system is able to filter out inconsistent queries with the help of a DL reasoner. Moreover, when the input is insufficient to reach the intended query, our system performs a semantic enrichment of the input that allows to further explore the possibilities taking into account the semantics of the added terms.

As the search space for a generalized interpretation might be quite large, we have presented the reduction techniques that QueryGen applies to limit the generated queries without losing possible interpretations. On the one hand, the semantics of the operators (their properties) are considered all along the process, making it possible to avoid generating queries that would be semantically redundant. On the other hand, QueryGen applies modularization techniques to reduce the search space when using virtual terms. Finally, the similar syntactic structure of the queries is exploited to present them in a compact way (patterns), reducing the shown options and, thus, not overwhelming the user.

Chapter 6

Accessing Data

In this chapter, we describe the strategy adopted by our system to deal with the different characteristics of the underlying information systems. We present an architecture based on *Adapters*, an evolution of the *wrappers* of OBSERVER [MI01], that enables the system to manage the different characteristics that an information system might exhibit (e.g., different types of query processing -snapshot vs. continuous ones-, different strategies to obtain the underlying data, heterogeneous data formats, etc.) in a flexible and extensible way. Finally, we present two different use cases that have been successfully integrated in QueryGen: DBpedia [BLK+09] (its SPARQL endpoint), and LOQOMOTION [IMI06]. Regarding this last system, we also will discuss how we have extended the semantics of its location model.

6.1 Using the Adapters

Recalling the characterization of information systems in Section 3.3, we have focused on how to adapt the keyword query model to the different query models Q_m that are made available to our system. However, when it comes to accessing the actual data in each of the underlying information systems, the characteristics of both the underlying data model D_m, and the answer calculation function f_m have to be taken into account and unified. So, once our system has the intended semantic query, it has to access the underlying data corresponding to such semantics. These underlying data might be stored in different data repositories, with different data organizations, and with different query capabilities (query languages and formats of answers) [VP97]. Moreover, the accessed data repositories might support queries of different nature, e.g., some repositories process only snap-

shot queries, while others might be capable of processing continuous ones, refreshing the answers continually and requiring a different communication schema.

To provide QueryGen with enough flexibility to deal with this data heterogeneity, we advocate for the architecture shown in Figure 6.1, whose main modules are the following:

- Dispatcher: Once the user selects her/his intended query from those generated by QueryGen, the Dispatcher poses the query to the underlying data repositories that are able to process it. It consults the characteristics of their query processing services that their *Adapters* expose, and, depending on them, creates the appropriate communication channels. For example, a snapshot query only implies one answer, while a continuous query needs answer refreshments along the time. Finally, it correlates the data coming from the different systems and presents them to the user.

- Adapter: It wraps the access to the data stored in information systems with a certain data organization (e.g., there is an Adapter for relational databases, a different one for SPARQL endpoints, etc.). It registers itself in QueryGen providing information about the querying capabilities of the accessed information system, and making itself available to the Dispatcher. There is one instance of the appropriate kind of Adapter for each system accessed by QueryGen.

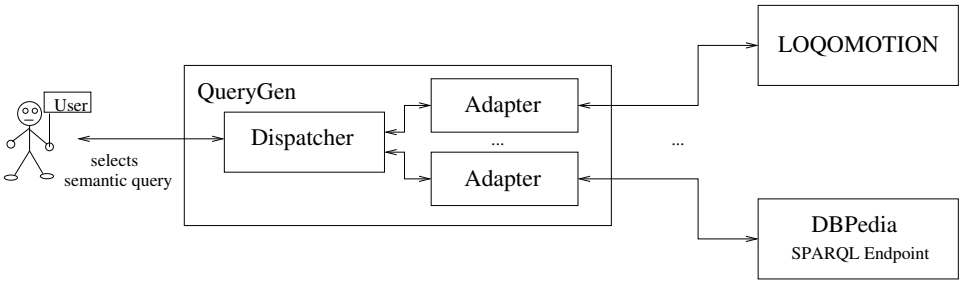


Figure 6.1: Our system can retrieve data from different channels and data models.

Each Adapter Ad_i is characterized by a tuple

$$< \{< lang_{id}, lang_{id}^{spec} >\}, \{Q_{type}\}, \{d_{format}\} >$$

where:

- $\{< lang_{id}, lang_{id}^{spec} >\}$ is the set of languages that the Adapter Ad_i is able to process. When the Adapter registers itself, the Dispatcher checks whether the language has been already added to the system. If it was, it adds Ad_i to the list of systems that are capable of processing $lang_{id}$. Otherwise, it adds the new language to the system, requiring the Analysis Table Constructor to build the information needed to enable QueryGen to use it.

- $\{Q_{type}\}$ is the type of queries that the Adapter Ad_i supports. In particular, QueryGen is able to deal with the following types:

  - Snapshot queries: They are posed and answered once, in a pull way (e.g., a SQL query against a relational DB).
  - Monitoring queries: They are posed once, but the answer is updated continually in a pull way (the Dispatcher is responsible for retrieving the new data). This type of queries is useful when, independently of the frequency at which the answer/data might change, the updates are not critical and, therefore, the system might relax the resources requirements, executing a snapshot periodically (e.g., a Web service providing information about the weather, invoked every hour).
  - Continuous queries: They are posed once, but the answer is updated continually via update events in a push way (e.g., a location dependent query where the position of the mobile objects is continuously changing [IMI06]). It is the underlying system, via its Adapter, who updates the answer at the requested query refreshment frequency.

- $\{d_{format}\}$ is the set of data formats that the Adapter Ad_i is able to offer (e.g., CSV values, RDF/XML, etc).

Adapters are in charge of performing the translation of the syntax-free semantic queries into the final languages that their underlying systems process. This architecture is an evolution of the *wrappers* proposed in OBSERVER [MI01], which adapted the queries to the different answering capabilities of the data repositories. In particular, the main capabilities that our system inherits from OBSERVER are the data integration capabilities and the ability of processing incomplete queries, this is, accessing several sources to obtain an appropriate answer (this is done at Adapters level). The separation into two elements allows us to increase the flexibility and

integrate systems that have different query processing capabilities, not only concerning the language expressivity, but also concerning both the query types and data models.

Note that, independently of the underlying data repository accessed, the system only access to data that corresponds to the semantics stated by the user, which have been maintained all along the process, thanks to the transformation of plain input keywords into a semantic query representing exactly what the user wanted to retrieve.

Finally, an important remark: Adapters also hold information about the data schemas of their underlying information systems. In particular, they must hold information about how to map the input senses to their actual schemas. Regarding this issue, currently, Adapters are hard coded for each of the adapted systems. However, we advocate (and leave it as future work) for using the synonymy measures used for the disambiguation to align the queries with the underlying data.

In the following sections, we present how we have integrated successfully the data access to two very different information systems such as the location-dependent query processor LOQOMOTION [IMI06] (which processes queries continuously) and DBpedia [BLK+09] (as SPARQL Endpoint). In the case of LOQOMOTION, we also present the semantic models that we have developed for extending its semantic capabilities.

6.2 Accessing DBpedia from DL Queries

DBpedia is a huge repository of structured data that provides a semantic entry point to Wikipedia. It uses several different types of information extractors to convert the information in Wikipedia's articles into structured information which is published under the principles of Linked Data [BHBL09] using RDF, SKOS[1], and OWL. This extraction is performed automatically by exploiting the structure of the information stored in Wikipedia. However, the nature of the results of this extraction process differs from the sources in more ways than barely structural and format ones. The articles extracted from Wikipedia, once in DBpedia, become *resources*. Each resource is represented by a URI and has a direct correspondence to its original Wikipedia's article, inheriting its categorization. The Wikipedia's categorization is extracted and included in DBpedia as a SKOS taxonomy (see Figure 6.2).

[1]SKOS Simple Knowledge Organization System, `http://www.w3.org/TR/skos-primer/`, last accessed October 3, 2013.

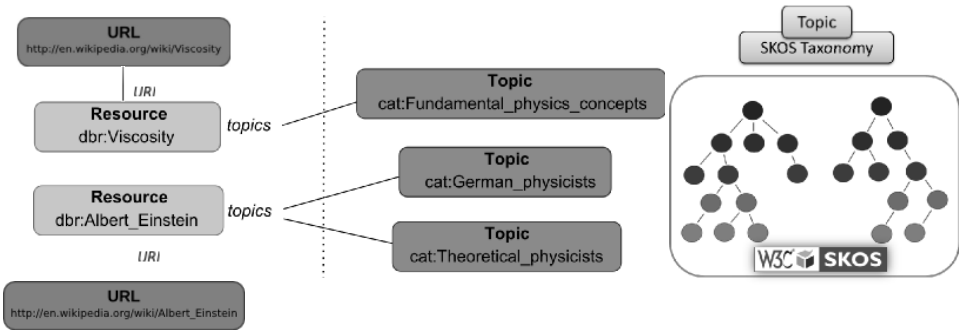


Figure 6.2: *Articles* in Wikipedia become *resources* in DBpedia, inheriting the URI of the article and its categorization.

When moving from the *article world* of Wikipedia to the *semantic resources* in DBpedia, there are objects that might augment their descriptions as the new semantic model can represent more information about them. The articles extracted from Wikipedia, once in DBpedia, become *resources*. Each resource is represented by an URI and has a direct correspondence to its original Wikipedia's article, inheriting its categorization. The whole taxonomy of article categories of Wikipedia is included as a SKOS ontology in DBpedia; thus, DBpedia provides a first view on the resources according to their category (see Figure 6.2).

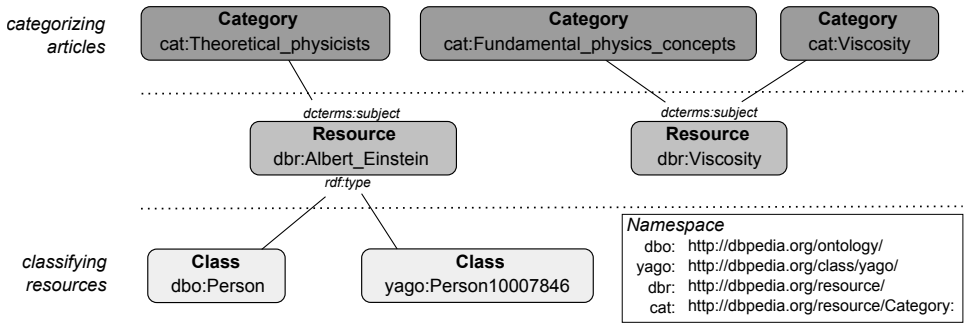


Figure 6.3: DBpedia excerpt of the descriptions of *Albert Einstein* and *Viscosity* resources.

Depending on the content of its corresponding article, a DBpedia resource might also be representing an object (see Figure 6.3). The classification of this object dimension of the resources is done via several general

domain ontologies, being DBpedia Ontology[2] and YAGO[3] the most important ones. In this way, independently of the article categorization, DBpedia offers a second different view based on the nature of the underlying resources. However, this view does not cover all DBpedia. There exist resources that, despite being categorized, do not have these descriptions as they are not defined in the used ontologies, as shown in Figure 6.3.

Summing up, DBpedia organizes knowledge in two major ways: The SKOS categorization, and an ontological classification; and exposes this knowledge through an SPARQL endpoint. In this thesis, we have added this endpoint using the DBpedia ontology as entry point. However, the SKOS categorization could also be used without an adaptation effort.

6.2.1 DBpedia Adapter

The *Adapter* for DBpedia we have developed is characterized by the following tuple:

$$< \{< SBack, SBackGrammar >\}, Snapshot, \{RDF\} >$$

That is, it supports the simplified version of BACK that we have used in the previous chapter, processes snapshots queries, and returns the data in RDF. As DBpedia offers a public SPARQL endpoint to access its data, the Adapter has to translate the DL queries into SPARQL to properly forward the possible queries. To do so, the Adapter traverses recursively the selected query (in fact, its associated query tree) applying the query rewriting rules[4] shown in Table 6.1. For simplicity's sake, only a binary version of *And* operator has been considered, altough the actual algorithm considers that *And* can have more than two operands. Moreover, during the traversal, new variable names are created to bind the resources appropriately.

In these rules, *obtainGraph* is the entry point for the recursion, as it expands the concept expression that is passed as an argument until it reaches the leafs of the query tree. We assume that the underlying RDF repository has materialized, at least, the hierarchy inferences. Note how the translations of the operators that are applied on a role bind the answer resources

[2]The DBpedia Ontology, `http://wiki.dbpedia.org/Ontology`, last accessed October 3, 2013.

[3]YAGO Ontology, `http://www.mpi-inf.mpg.de/yago-naga/yago/`, last accessed October 3, 2013.

[4]We consider SPARQL v1.1, where the filter NOT EXISTS was added. The status of SPARQL v1.1 has changed from Proposed Recommendation to Recommendation during the writing of this thesis. Besides, repositories such as VirtuosoDB currently supports it.

And(C_1,C_2) $\Rightarrow$	[{ ?x *a obtainGraph*(C_1, ?x). ?x *a obtainGraph* (C_2, ?x) }
Some(R_1,C_2) $\Rightarrow$	[{ { ?x *a Dom*(R_1). ?x R_1 ?y }. ?y *a obtainGraph*(C_2, ?y) }
All(R_1,C_2) $\Rightarrow$	[{ ?x *a Dom*(R_1). *FILTER NOT EXISTS* { ?x R_1 ?y . *FILTER NOT EXISTS* { ?y *a obtainGraph*(C_2, ?y) } } }
Fill(R_1,I_1) $\Rightarrow$	[{ ?x *a Dom*(R_1). ?x R_1 I_1 }
C_1 $\Rightarrow$	{ ?x *a* C_1.}

Table 6.1: Rewriting rules applied by the DBpedia Adapter.

to belong to the domain of the role. If such an assumption cannot be made, we should relax these constraints. Otherwise, operators such as *Some* or *All* would return none results unless the direct assertions would have been made (the RDF statements of the resources belonging to the domain of a property would not be present, and therefore any subgraph would be matched). Finally, the obtained graph expression is extended by adding the different bindings needed to retrieve the properties that are considered in the *Projections* operator.

In the following subsection, we present a complete example of accessing data from DBpedia.

6.2.2 A Complete Example with Data from DBpedia

To illustrate how our system works, we will give a complete example from the input of the user to the data retrieved by our system from DBpedia[5]. We restrict the semantics to the different ontologies used in DBpedia for the sake of simplicity in the explanations. As an illustrative example, let

[5]The namespaces used in this section are *dbpedia* `http://dbpedia.org/`, *dbo* `http://dbpedia.org/ontology/`, and *dbpprop* `http://dbpedia.org/property/`.

us assume that a user watched old cartoons starred by a dumb tall black dog many years ago. S/he does not recall its name (in fact, s/he is thinking about Goofy, the Disney character), but s/he wants to know since when this character exists. As s/he cannot provide more specific input, s/he inputs the keywords "Fictional Dog Appearance":

1. In the disambiguation process, our system offers the user several interpretations for each keyword:
   - For "Fictional", one of the proposed meanings is the concept `dbo:FictionalCharacter`.
   - For "Dog", one of the proposed meaning is an integrated sense containing DBpedia URL `dbpedia:resource/Dog`, considered as an instance of *Animal* in the DBpedia ontology.
   - For "Appearance", one of the proposed meanings is the role `dbo:firstAppearance`.
2. In the generation process, the user cannot find the intended query, as no combinations of *FictionalCharacter*, *Dog* and *firstAppearance* represents her/his intended query. However, by considering one VT during the semantic enrichment step, the system can try adding the role `dbpprop:species`. This allows the system to find out the query intended by the user:

   $[firstAppearance](FictionalCharacter\ And\ (Fill\ species\ Dog))$

   which has to be read as "retrieve the first appearance of the fictional characters whose species is dog".
3. The system detects that the query can be processed by the DBpedia Adapter as it supports the language of the query, so it forwards the query to it, which translates the query into the corresponding underlying query language (a SPARQL sentence):

```
SELECT  * FROM <http://dbpedia.org>
WHERE { ?x a dbo:FictionalCharacter.
        ?x dbpprop:species
            <http://dbpedia.org/resource/Dog>.
        ?x dbo:firstAppearance ?y. }
```

which retrieves the first appearance of several fictional dogs (see Table 6.2), among which Goofy's can be found[6].

Character **FirstAppearance**
http://dbpedia.org/resource/Max_Goof http://dbpedia.org/resource/Goof_Troop
http://dbpedia.org/resource/Max_Goof http://dbpedia.org/resource/Fathers_Are_People
http://dbpedia.org/resource/Bolt_(character) http://dbpedia.org/resource/Bolt_(2008_film)
http://dbpedia.org/resource/Goofy *http://dbpedia.org/resource/Mickey's_Revue*
http://dbpedia.org/resource/Spike_and_Tyke_(characters) http://dbpedia.org/resource/Dog_Trouble
http://dbpedia.org/resource/Huckleberry_Hound http://dbpedia.org/resource/Huckleberry_Hound_Meets_Wee_Willie
http://dbpedia.org/resource/Droopy) http://dbpedia.org/resource/Dumb-Hounded

Table 6.2: Results for the first appearance of fictional dogs returned by DBpedia (including Goofy's).

In this example, the system presented an average of five senses for each input keyword, which resulted in 14 query patterns representing 172 queries. If we search Google using the same input ("Fictional Dog Appearance"), it returns 12.800.000 results, without any reference to Goofy in the ten first pages. The first result returned by Google links a list of famous fictional dogs in Wikipedia. But in that list there is no answer to the user query (first appearance of Goofy): S/he has to browse the whole list of 65 dogs to find Goofy (and recall its name, hopefully), and click its page to look for the information inside the text. Notice that the list returned by our system contains the first appearances of dog characters, while the list in Wikipedia links to dog characters pages (not necessarily containing their first appearance). Thus, taking the same input as starting point, our system has performed a semantic search returning the first appearance of fictional dogs, while using a search engine we can just obtain information about dogs.

If we replace in the input "Appearance" by "series", our system is able to answer, for example, in which series Odie and Scrappy Doo appear.

[6]The amount of results that the public SPARQL endpoint of DBpedia provides depends on the current workload. The results presented are from June 21, 2013.

6.3 Accessing LOQOMOTION with Extended Semantics

LOQOMOTION [IMI06] processes continuous location-based queries in a distributed and efficient way using mobile agents. During this thesis, we have extended its query language by adding semantics to the locations it was able to deal with. This bridges the semantic gap that there exists from GPS locations to the user's vocabulary, and extends the semantics of location constraints.

In this section, we firstly present an overview of the network of agents that LOQOMOTION deploys to process the queries. Secondly, we present the notion of *semantic location granules*, and we propose two complementary models for them. Then, we analyze the influence of using location granules on the expressivity of location-dependent queries, and how they can be integrated into the query model of LOQOMOTION. Finally, we present the Adapter that allows QueryGen to process queries using LOQOMOTION. The language used is the SQL-like query language presented in Section 6.3.3, and we will see how, through the use of the local and global conditions, QueryGen is able to perform the semantic checking even with non-DL query languages, such as this one.

6.3.1 LOQOMOTION Architecture

Firstly, we present the basics of LOQOMOTION without considering location granules. No attempt is made to justify the use of mobile agents or the relation between LOQOMOTION and other systems for location-dependent query processing (for details about this, see [IMI06] and/or the survey in [IMI10]). To clarify the explanations, the scenario shown in Figure 6.4, and a query that retrieves the interesting objects within the (inner) moving *query circles* (*relevant areas*) centered on the reference objects *car38* and *policeCar5*, will be considered[7]. In LOQOMOTION there is a static agent called *QueryMonitor*, executing on the mobile device of the user, and three different types of agents (two of them, mobile agents [TIM07,BIM10a]) are in charge of processing the query on the fixed network:

- A mobile agent *MonitorTracker* on the fixed network (initially, on *Proxy6* in Figure 6.4) is in charge of communicating, to the user device, the updated data about moving objects relevant to the query

[7]The areas shown in Figure 6.4 are circles, but this will not be necessarily the case when location granules are used.

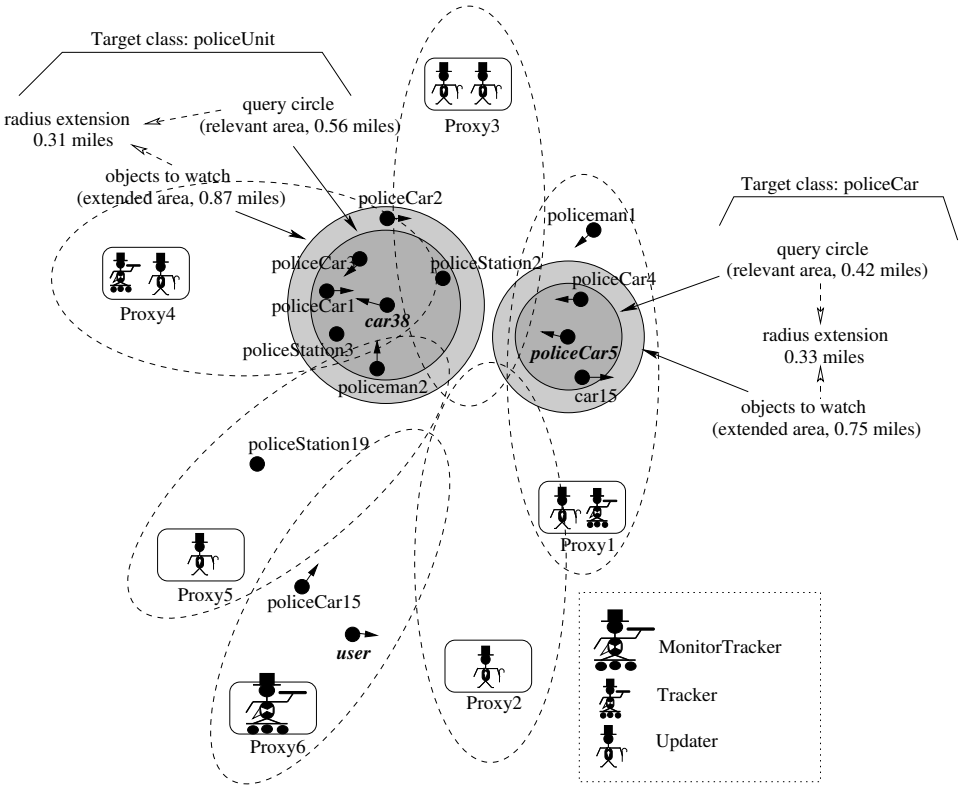


Figure 6.4: Query processing in LOQOMOTION: sample scenario.

(*target objects*). This agent always executes on the proxy corresponding to the proxy area where the user is, following the user, to optimize communications with the mobile user device.

- For each reference object, a mobile agent *Tracker* keeps itself on the proxy that handles the location of that reference object to track it (e.g., in Figure 6.4 the Tracker for *car38* is initially executing on *Proxy4*, as *car38* is within the coverage of that proxy), and computes the area that contains the objects that must be communicated to the Monitor-Tracker (called *extended area*).

- For each proxy whose area intersects the extended area (the *relevant proxies*), a static *Updater* agent is created by such a Tracker (e.g., in Figure 6.4, the Tracker on *Proxy4* sends Updaters to *Proxy3*, *Proxy4*, and *Proxy5*). An Updater is in charge of retrieving the interesting objects within its area by executing *standard queries* (i.e., queries without

location-dependent constraints such as *inside*) on its proxy.

The different mobile agents maintain themselves on the relevant proxies, to keep track of the interesting objects. In this way, they support an efficient continuous query processing, as results can be ready when a refreshment is needed. The use of mobile agents in LOQOMOTION facilitates: 1) tracking the positions of relevant objects efficiently, 2) optimizing the wireless communications, and 3) supporting the distributed query processing efficiently. For further details about this system, see [IMI06].

6.3.2 Semantic Location Granules

The existing work on location-dependent query processing implicitly assumes GPS locations for the objects in a scenario (e.g., [SWCD97, PXK+02, MXHA05, CHCX06, GL06, DTS08, IMI10]). However, some applications do not require location data at GPS resolution, and a coarser representation may be more appropriate for them. For example, a train tracking application would need to just consider in which city a train is currently in, and not its exact coordinates. For such applications, it is useful to define the concept of *location granule* as a set of physical locations [IMB07, ICBM09, BIM10b, IBM11, BBMI13]. This concept is similar to the concept of *place* in [Hig03, HS07, HS09] or *spatial granule* in [BCP09].

However, when we group a set of locations and give them a name, we are implicitly giving them also a meaning. For example, the set of locations that compose Madrid, when grouped under the name *Madrid*, become a city, the capital of Spain. Thus, the grouped locations become a new different entity as a whole, which could be related to other entities in different ways besides spatial relations. So, to model them, *semantic location granules* [BIM10b, BBMI13] are introduced (as defined in [BIM10b]):

Definition 6.3.1 *A* semantic location granule *is a location granule with well-defined semantics, i.e., explicitly stated.*

Their semantics can be modeled using ontologies and, depending on the role that the locations are having in the system, we advocate for two complementary models for them. The former one considers the location granules as instances of a concept *Granule*, and allows us to extend the query model using logical rules on the ABox. The latter one, on the other hand, considers that the granules themselves are concepts, as they subsume a set of locations (which now become the instances). As we will see, this latter model allows the DL reasoner to make intensive use of the TBox to infer the containment relationships.

6.3.2.1 Modeling Semantic Location Granules as Instances

From an object-oriented point of view, one could argue that the location granules are instances of different types of classes, that would be the ones that define their characteristics [BIM10b]. Thus, modeling them in this way, we proposed the ontology in Figure 6.5. In this model, the notion of *semantic granule map* appears explicitly:

Definition 6.3.2 *A* semantic granule map *is a set of semantic granules identified by a common name. It provides the global semantics of the location granules that participate in it.*

Thus, semantic location granules are also grouped to provide an interpretation of the location space, conforming semantic layers.

This ontology is not meant to be complete, but a starting point to be extended and adapted to particular scenarios. The most basic properties that we identified are the following (see Figure 6.5):

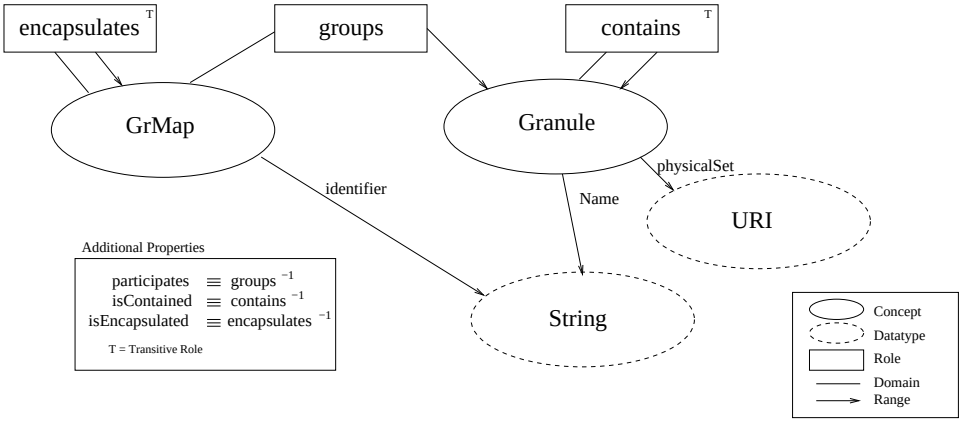


Figure 6.5: Semantic location granules as instances: base ontology.

- *Contains*: It represents the physical inclusion of a granule inside another. For example, the granule *Spain* (the country) contains, among others, the granules *Madrid* and *Barcelona* (the cities). This property permits to establish a spatial hierarchy to organize the granule instances. Its inverse property is *isContained*.
- *Groups*: It represents the relationship that there exists between a granule map and the granules that make it up. For example, the

granule map *Countries* would group the granules *Spain*, *France*, etc. Its inverse property is *participates*. Note that a granule can participate in several granule maps.

- *Encapsulates*: It allows to establish hierarchies between granule maps according to the granularity level. For example, the granule map *provincesOfSpain* could encapsulate *citiesOfSpain*. Its inverse property is *isEncapsulated*.

The rest of the properties are used to identify the granules (*name*) and the granule maps (*identifier*), and to associate a granule to one or several sets of physical coordinates (*physicalSet*). The physical coordinates are also accessible at the semantic level to allow including statements about them in the asserted knowledge and, therefore, to allow the system to perform spatial reasoning (for example, using RCC [GSBM08, RCC92]).

Although this basic ontology may seem too simple, direct benefits can be obtained out of it. For example, even without any additional extension in the semantics, the presentation of results of the queries can be enhanced by exploiting the inclusion relationships to offer different views of the same answer set. Besides, we wanted to keep the model as simple as possible to make it easier to adapt it to the desired semantics.

6.3.2.2 Modeling Semantic Location Granules as Concepts

On the other hand, if we consider that each of the locations themselves are instances, a semantic location granule must be modeled as a concept, as it subsumes geometrically their components [BBMI13].

To formalize the concepts of semantic granules and semantic granule maps with DLs, we will consider that a transitive role named *isContained* and concrete features $loc_{x_1}, \ldots, loc_{x_n}$ are defined in our model. Intuitively, the role *isContained* will be used to express that a granule is geographically contained in another one by subsumption and participation in the relationship, (e.g., $NewYork \sqsubseteq \exists isContained.EEUU$), and $locx_1, \ldots, locx_n$ will be the coordinates of a point. These concrete features allow us to define areas, for example, $loc_x \geq 10 \wedge loc_x \leq 20 \wedge loc_y \geq 20 \wedge loc_y \leq 30$ (if we work in $\mathbf{R}^2$, we write loc_x, loc_y instead of loc_{x_1}, loc_{x_2}).

Definition 6.3.3 *An* area concept $f(loc_{x_1}, \ldots, loc_{x_n})$ *is a concept built with operators* $\cap$ *and* $\cup$*, and a combination of path-free compositions using concrete features* $loc_{x_1}, \ldots, loc_{x_n}$*. An* area concept name A *is a concept name*

such that $A \equiv f(loc_{x_1}, \ldots, loc_{x_n})$*, where* f *is an area concept. The set of names of area concepts is denoted by* N_A*.*

For example, $f(loc_x, loc_y) = \exists loc_x.\geq_{10} \sqcap \exists loc_x.\leq_{20} \sqcap \exists loc_y.\geq_{10} \sqcap \exists loc_y.\leq_{15}$ is an area concept, while $\exists loc_x.\geq_{10} \sqcap \exists loc_x.\leq_{20} \sqcap \exists loc_y.\geq_{10} \sqcap \exists loc_y.\leq_{15} \sqcap City$ is not.

Definition 6.3.4 *Given* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *a knowledge representation and* M *a granule map, a* semantic granule map *is a tuple* $(M, \mathcal{K}, area, semGranule)$*, where* $area$ *and* $semGranule$ *are functions from the set of granules of* M *to the concept names of* $\mathcal{T}$*; that is,* $area$*,* $semGranule\colon M_G \rightarrow N_C$*, such that the following must be satisfied for all* $G \in M_G$*:*

1. $semGranule(G) \sqsubseteq area(G)$
2. $area(G) \sqsubseteq \exists isContained.semGranule(G)$

The concept names $semGranules(G)$ are called *semantic granules.*

Let us show an example to explain the definition. Let M be a granule map with location granules $M = \{ZaragozaGr, AragonGr, MadridGr, SpainGr, FranceGr, PortugalGr, EuropeGr\}$, and let $\mathcal{T}$ be the following TBox[8] of a knowledge representation $\mathcal{K}$:

$$Zaragoza\_Area \equiv \exists loc_x.\geq_{25} \sqcap \exists loc_x.\leq_{30} \sqcap \exists loc_y.\geq_{23} \sqcap \exists loc_y.\leq_{30} \tag{6.1}$$
$$Aragon\_Area \equiv \exists loc_x.\geq_{25} \sqcap \exists loc_x.\leq_{30} \sqcap \exists loc_y.\geq_{20} \sqcap \exists loc_y.\leq_{32} \tag{6.2}$$
$$Madrid\_Area \equiv \exists loc_x.\geq_{15} \sqcap \exists loc_x.\leq_{20} \sqcap \exists loc_y.\geq_{17} \sqcap \exists loc_y.\leq_{23} \tag{6.3}$$
$$Spain\_Area \equiv \exists loc_x.\geq_{5} \sqcap \exists loc_x.\leq_{35} \sqcap \exists loc_y.\geq_{0} \sqcap \exists loc_y.\leq_{35} \tag{6.4}$$
$$France\_Area \equiv \exists loc_x.\geq_{20} \sqcap \exists loc_x.\leq_{60} \sqcap \exists loc_y.\geq_{35} \sqcap \exists loc_y.\leq_{85} \tag{6.5}$$
$$Portugal\_Area \equiv \exists loc_x.\geq_{0} \sqcap \exists loc_x.\leq_{5} \sqcap \exists loc_y.\geq_{5} \sqcap \exists loc_y.\leq_{30} \tag{6.6}$$
$$Europe\_Area \equiv \exists loc_x.\geq_{0} \sqcap \exists loc_x.\leq_{250} \sqcap \exists loc_y.\geq_{0} \sqcap \exists loc_y.\leq_{250} \tag{6.7}$$
$$Zaragoza \equiv Zaragoza\_Area \sqcap City \tag{6.8}$$
$$Aragon \equiv Aragon\_Area \sqcap Region \tag{6.9}$$
$$Madrid \equiv Madrid\_Area \sqcap City \tag{6.10}$$
$$Spain \equiv Spain\_Area \sqcap Country \sqcap \exists vat.=_{21} \tag{6.11}$$
$$France \equiv France\_Area \sqcap Country \sqcap \exists vat.=_{19} \tag{6.12}$$
$$Portugal \equiv Portugal\_Area \sqcap Country \sqcap \exists vat.=_{23} \tag{6.13}$$
$$Europe \equiv Europe\_Area \sqcap Continent \tag{6.14}$$

[8]To keep explanations easier to follow, we represent geographic areas in the TBox by simple rectangles instead of the real geographic limits.

$$Zaragoza\_Area \sqsubseteq \exists isContained.Zaragoza \tag{6.15}$$
$$Aragon\_Area \sqsubseteq \exists isContained.Aragon \tag{6.16}$$
$$Madrid\_Area \sqsubseteq \exists isContained.Madrid \tag{6.17}$$
$$Spain\_Area \sqsubseteq \exists isContained.Spain \tag{6.18}$$
$$France\_Area \sqsubseteq \exists isContained.France \tag{6.19}$$
$$Portugal\_Area \sqsubseteq \exists isContained.Portugal \tag{6.20}$$
$$Europe\_Area \sqsubseteq \exists isContained.Europe \tag{6.21}$$
$$Region, Country, Continent, City \text{ are mutually disjoint} \tag{6.22}$$
$$RedWine \sqsubseteq Wine \tag{6.23}$$
$$RedWine \sqsubseteq \exists isTypical.\exists isContained.Aragon \tag{6.24}$$
$$RedWine \sqsubseteq \exists isTypical.\exists isContained.France \tag{6.25}$$
$$MadridShop \equiv Shop \sqcap \exists isContained.Madrid \tag{6.26}$$
$$AragonWine \equiv Wine \sqcap \exists isContained.Aragon \tag{6.27}$$

We define a semantic granule map, $(M, \mathcal{K}, area, semGranule)$, where $area$ and $semGranule$ are the functions $area(SpainGr) = Spain\_Area$, etc., and $semGranule(SpainGr) = Spain$, etc. We can ensure that this is a semantic granule map since it holds the conditions (1) and (2) of Definition 6.3.4 from axioms (6.8)-(6.14), and (6.15)-(6.21), respectively. Figure 6.6 shows the map corresponding to the modeled area.

Figure 6.6: Sample granule map.

Intuitively, the condition (1) of Definition 6.3.4 says that a semantic granule is not only its geographic area, but it could also have more attributes

($semGranule(G) = area(G) \sqcap C$). For example, *Zaragoza* is an area and a *City*, and *Spain* is an area and a *Country*. The condition (2) allows to establish qualitative relations between granules such as "*Zaragoza* is a city in *Spain*", i.e., $Zaragoza \sqsubseteq \exists isContained.Spain$, or to express the concept "Aragon's wines", $Aragon\_Wine \equiv Wine \sqcap \exists isContained.Aragon$. Note that we do not express the concept Aragon's wine as $Wine \sqcap Aragon$, since *Aragon* is a *Region* and wines are not regions; and similarly, Aragon's wines are not defined as $Wine \sqcap Aragon\_Area$ as wines could not have location information. We have divided the containment relationship in two parts: One to make calculations about areas (quantitative reasoning) using the subsumption relationship, and another one to establish relationships with other concepts (qualitative reasoning) using the *isContained* relationship.

We emphasize that two different functions are needed in the definition of semantic granule map to distinguish the area of a granule from the semantic granule. Usually, granules have attributes which are incompatible. For example, *Zaragoza* is a city and *Spain* is a country, and a city cannot be a country. So, let us suppose that we do not differentiate between the area of a granule and the granule itself. In this case, for example, it would be held that $Zaragoza\_Area \equiv Zaragoza$ and $Spain\_Area \equiv Spain$. Thus, the TBox $\mathcal{T}$ would be inconsistent since $Zaragoza \sqsubseteq City$ and $Zaragoza \sqsubseteq Spain \sqsubseteq Country$, and Zaragoza is not a country.

From this model, a DL reasoner can deduce a number of facts, as we explain in the following.

Proposition 6.3.5 *A reasoner under conditions of Definition 6.3.4 can infer that:*

1. *a granule G is contained in a granule G', i.e., it can be deduced that $semGranule(G) \sqsubseteq \exists isContained.semGranule(G')$*
2. *a granule G intersects a granule G'*

Proof From the definition of the area concept, using concrete domains, a reasoner can infer that $area(G) \sqsubseteq area(G')$, and therefore that:

$$semGranule(G) \sqsubseteq area(G) \sqsubseteq area(G') \sqsubseteq \exists isContained.semGranule(G')$$

In the example above, it can be inferred that *Zaragoza* is contained in *Spain*, $Zaragoza \sqsubseteq Zaragoza\_Area \sqsubseteq Spain\_Area \sqsubseteq \exists isContained.Spain$.

The second statement is obvious since it is equivalent to asking if the area concept $area(G) \sqcap area(G')$ is satisfiable.

Another interesting remark is that the content of a granule only depends on its area. What does this mean? For example, let us suppose that we define two different semantic granules with the same area, $VaticanCity$ and $VaticanCountry$, defined as $Vatican\_Area \sqcap City$ and $Vatican\_Area \sqcap Country$, respectively. We would like that the content of Vatican as a city, $\exists isContained.VaticanCity$, be equal to the content of the Vatican as a country, $\exists isContained.VaticanCountry$, even when a country is not a city. Due to the conditions (1) and (2) of the Definition 6.3.4 and the transitivity of the role $isContained$, we can conclude that in our model $\exists isContained.VaticanCity \equiv \exists isContained.VaticanCountry$, as it is shown in the following proposition.

Proposition 6.3.6 *Let G be a granule under conditions of Definition 6.3.4. Then it holds that*

$$\exists isContained.semGranule(G) \equiv \exists isContained.area(G).$$

And therefore, if G_1 and G_2 are granules such that $area(G_1) \equiv area(G_2)$, then

$$\exists isContained.semGranule(G_1) \equiv \exists isContained.semGranule(G_2).$$

Proof By condition (1) of Definition 6.3.4, $semGranule(G) \sqsubseteq area(G)$, and so $\exists isContained.semGranule(G) \sqsubseteq \exists isContained.area(G)$. Now, let us prove that $\exists isContained.area(G) \sqsubseteq \exists isContained.semGranule(G)$, obtaining the desired result. Using condition (2) of Definition 6.3.4 and the transitivity of role $isContained$, we can deduce

$$\exists isContained.area(G) \sqsubseteq \exists isContained.\exists isContained.semGranule(G) \sqsubseteq$$
$$\sqsubseteq \exists isContained.semGranule(G).$$

The second part of the proposition is trivial.

We are currently extending this model to make it possible to infer automatically another different topological relationships, thus further enriching both qualitative and quantitative reasoning on the model.

6.3.3 Location-dependent Queries with Location Granules

In this thesis, we consider a *location-dependent query* any query whose answer depends on the locations of certain objects (the mobile user and/or

other interesting objects). For example, a query that retrieves the locations of the taxis around a person who is searching for a cab is an example of a location-dependent query. In comparison with other proposals, such as [SDK01], the above mentioned definition of location-dependent query is quite general [IMI10]. We also contribute to extend the expressivity of this kind of queries by enabling the use of the locations at different granularity levels, which extends the range of location-dependent queries that can be expressed. For example, a continuous query such as "retrieve the cars that are within 100 miles of the city where *car38* is, showing their locations with city granularity" (i.e., indicating the city where each retrieved car is) could be submitted to keep track of the interesting moving objects and their current cities.

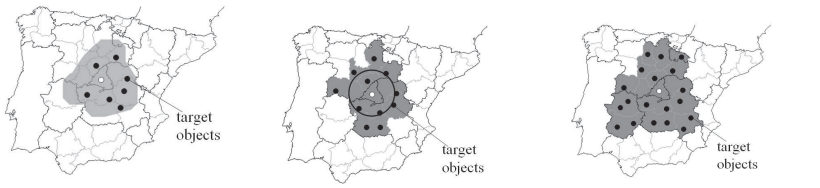
The use of location granules in location-dependent queries, as presented in [IMB07, IBM11], can have an impact on: 1) the presentation of results (location granules can be represented by using graphics, text, sounds, etc., depending on the requirements of the user), 2) the semantics of the queries (the user expresses the queries according to her/his own location terminology, and therefore the answers to those queries will depend on the interpretation of location granules), and 3) the performance of the query processing (the location tracking overload is alleviated when coarse location granules, instead of precise GPS locations, are used).

For the moment, in order to ease the explanations, we leave aside the semantic dimension of location granules. In [IBM11], we defined an inside constraint using granules and granule maps which is used to ask for objects o of a certain type (e.g., cars, ambulances) that have an attribute $o.loc$ which defines its location. There are three types of *inside* constraints as defined in [IBM11]; in the following, $r \in \mathbf{R}$, l is a location point, G is a granule, M is a granule map (a set of granules grouped under a common name), and d is a distance function defined for locations and granules:

$$\begin{aligned} inside(r, G, type) &= \{o \in type \mid d(o.loc, G) \leq r\} \\ inside(r, l, M, type) &= \{o \in type \mid o.loc \in G_i, d(l, G_i) \leq r, G_i \in M\} \\ inside(r, G, M, type) &= \{o \in type \mid o.loc \in G_i, d(G_i, G) \leq r, G_i \in M\} \end{aligned}$$

Examples of these queries are: Retrieve museums within 10 miles of a given province, museums in provinces which are within 10 miles from a given point, and museums in provinces which are within 10 miles from a given province. We illustrate the different types of inside constraints in Figure 6.7.

An SQL-like syntax is used to express the queries (the justification for this can be found in [IBM11]). The detailed syntax of the types of queries



$inside(r, G, type)$ $inside(r, g, M, type)$ $inside(r, G, M, type)$

Figure 6.7: Examples of different types of inside constraints.

considered in this work is shown in Figure 6.8, where it can be seen how location granules can be used in the queries. Nonterminals in the grammar are represented with initial uppercase and terminals are in lowercase (keywords) or surrounded by single quotes (literals). The start symbol of the grammar is *Query*. The symbol *gr* stands for *granule*: *gr(map-id, obj-id)* indicates that the location of the object named *obj-id* must be interpreted as a granule in the location granule map identified by *map-id*, and similarly *gr(map-id, class)* generalizes this to all the objects of *class*. It should be noted that there is a difference between a location appearing in the SELECT clause (identified with *Loc-Select*) and a location appearing as part of a location-dependent constraint (*Loc-Ref* or *Loc-Target*). In the first case the second argument of *gr* must be a class name, whereas in the second case it can be a class name (for *Loc-Target*) or an object identifier (for *Loc-Ref*). This difference is consistent with the way projections are performed in standard SQL. For example, *gr("city", Car)* in a SELECT clause would imply retrieving the granule (of the location granule map named "city") for each object of class *Car* retrieved by the query. The non-terminal *Object-id* represents a value of the attribute *id* of *Object*. Thus, it is not an object identifier (OID) in the object-oriented sense; the same can be said about the non-terminal *Map-id* regarding the attribute *id* of a *Location Granule Map*.

As indicated in Figure 6.8, location granule maps can be referenced in the SELECT and/or in the WHERE clause of a query, depending on whether their corresponding location granules must be used for the visualization of results and/or for the processing of constraints, respectively (of course, both usages can also appear simultaneously in the same query). Moreover, only the *Inside* operator has been included in the grammar for the sake of simplicity in the explanations of the language.

General query structure

Query	→	*select* Projections *from* Class-names (*where* Conds)?
Class-names	→	Class-name (‘,’ Class-name)*
Projections	→	Attr-Loc-Select (‘,’ Attr-Loc-Select)*
Attr-Loc-Select	→	attribute | Loc-Select
attribute	→	Qualified-attr | Unqualified-attr
Qualified-attr	→	Class-name ‘.’ Unqualified-attr
Loc-Select	→	Object-id ‘.’ ‘loc’ | *gr* ‘(’ Map-id ‘,’ Class-name ‘)’

Conditions can be standard conditions on attributes or location-dependent conditions

Conds	→	Cond ((*and* | *or*) Cond)*
Cond	→	(Bool-Cond | LDQ-Cond)
Bool-Cond	→	attribute Comp Value

Location-dependent conditions /* The focus is on inside constraints */

LDQ-Cond	→	*inside* ‘(’ Args-Inside ‘)’ | ...
Args-Inside	→	Radius ‘,’ Loc-Ref ‘,’ Loc-Target
Loc-Ref	→	Object-id | GPS-coord |
		gr ‘(’ Map-id ‘,’ Object-id ‘)’ |
		gr-map ‘(’ Map-id ’,’ Gr-id ‘)’
Loc-Target	→	Class-name | *gr* ‘(’ Map-id ‘,’ Class-name ‘)’
Radius	→	Real Units

Basic grammar productions

String	→	([a-z] | [A-Z] | [0-9])+
Real	→	([0-9]+) (‘.’ [0-9]+)?
Class-name	→	String /* Name of a class of objects */
Unqualified-attr	→	String /* Name of an attribute for a selected class */
Object-id	→	“ String ” /* Identifier of an object */
Map-id	→	“ String ” /* Identifier of a granule map */
Gr-id	→	“ String ” /* Identifier of a granule */
GPS-coord	→	‘(’ Real ‘,’ Real ‘)’ /* Two dimensions are assumed */
Units	→	*meters* | *kilometers* | *miles* | ...
Comp	→	‘=’ | ‘>’ | ‘<’ | ‘>=’ | ‘<=’ | ‘<>’
Value	→	([0-9]+) | “ String ”

Figure 6.8: Syntax of location-dependent queries with location granules.

6.3.4 LOQOMOTION Adapter

While being complementary, the Adapter for LOQOMOTION adopts the location model where the location granules are represented as instances. In this section, assuming this location model, we present how its query language is adapted to work with QueryGen.

The LOQOMOTION Adapter that we have developed is characterized

by the following tuple:

$$< \{< KeyLOQO, KeyLOQOGrammar >\}, \\ \{Snapshot, Monitoring, Continuous\}, \{CSV\} >$$

That is, it supports an adaptation of the SQL-like query language that we have presented in the previous section, it can process the three types of queries that QueryGen supports, and it returns the data in comma-separated values format. In Table 6.3, the adapted grammar is presented. According to the grammar definition presented in Section 5.2.1, the elements of this grammar are as follows:

- The initial symbol Q is *Query*.
- $\{Op_i\}$ contains *Projections, GrM, GrG, And, Or, Inside,* and *Comp* nonterminals; $\{Q_{ps}\}$ contains *Projection, Attrib, LocSelect, LocReference, LocTarget,* and *Loc*; and $\{R_{types}\}$ contains *Concept, Role,* and *Instance* nonterminals.
- T, as we have defined before, contains *C, R, I,* and ξ tokens.
- P contains each one of the productions in Table 6.3.

Note that there are additional elements: In this case, the operator that translates a location into a granule according to a map (*gr* operator, explained in Section 6.3.3) is separated into two different nonterminals depending on the arguments it takes (*GrM* and *GrG*), and, to be applicable, they need that the operators belong to specific concepts defined in the location model presented in Section 6.3.2.1, where location granules are modeled as instances.

In this language, there exist operands whose semantics are not directly checkable with a DL-reasoner. In particular, let us focus on *Inside*, *GrM*, *GrG*, and *Comp* operators, the most complex ones:

- *Inside* imposes a condition on the location attribute of the returned objects. The type of the returned objects is specified in the *LocTarget* production, and therefore, that is the concept that is propagated as global condition.
- *GrG* returns a granule object. It might take two different operands (along with the mapping to be used): A location point or the name of a granule in the mapping (in this case, an instance of granule). To

Production	**Global Conditions**	**Local Conditions**
Query → Projections Concept	And($1, $2)	
Projections → Projection Projections	And($1, $2)	
Projections → ξ	Neutral	
Projection → Attrib	$1	
Projection → LocSelect	$1	
Attrib → R	Dom($1)	
Attrib → C R	And($1, Dom($2))	Satisfiable(And ($1, Dom($2)))
LocSelect → Loc		
LocSelect → GrM	$1	
LocReference → Loc		
LocReference → GrG		
LocTarget → C	$1	Satisfiable($1)
LocTarget → GrM	$1	
Loc → I		
GrM → I C	$2	InstanceOf($1, GrMap) ∧ Satisfiable($2)
GrG → I I		InstanceOf($1, GrMap) ∧ InstanceOf($2, Granule)
GrG → I Loc		InstanceOf($1, GrMap)
Concept → C	$1	
Concept → And	$1	
Concept → Or	$1	
Concept → Inside	$1	
Concept → Comp	$1	
And → Concept Concept	And($1, $2)	
Or → Concept Concept	Or($1, $2)	
Inside → LocReference LocTarget	$2	
Comp → C R C R	Or($1,$3)	Satisfiable(And($1, Dom($2))) ∧ Satisfiable(And($3, Dom($4))) ∧ Satisfiable(And(Range($2), Range($4)))
Comp → R R	Or(Dom($1), Dom($2))	Satisfiable(And(Range($1), Range($2)))
Comp → R C C	Or($2, $3)	Satisfiable(And($2, Dom($1))) ∧ Satisfiable(And($3, Dom($1)))

Table 6.3: Annotated grammar for a subset of the query language of LOQOMOTION (KeyLOQO).

check that it is properly applied, the system has to check that the first operand is an instance of *GrMap*, and that the second operand (when

it is not derived from a *Loc* production) is an instance of *Granule*. This way, our system can perform a semantic checking on the operands. It does not impose a global condition, as it only affects to the interpretation of the reference position to define the interest area, and it does not constraint semantically the nature of the objects to be returned.

- *GrM* changes the way that the location constraint has to be processed. As we have seen in Section 6.3.3, it has different meanings depending on the position of the query. Anyway, the semantic checkings to be performed are the same ones for both situations. It needs two operands, a mapping and the concept of the target objects (the ones to be returned by the constraint). The checking is performed locally, and this time, it returns the concept of the target objects as global condition. This way, it can be propagated and its consistency checked along the rest of operands. For example, if we specified an inside constraint with *Dogs* as target objects, and the rest of the concept definition is about people (we assume that *Dog* and *Person* are stated as disjoint concepts), when the *Inside* or the *LocSelect* productions return the concept *Dog*, it will be checked against the rest of the expression.

- Finally, *Comp* is the comparison operator. It allows the system to generate JOINs, for example. We have modeled it in three flavors, depending on the number of concepts and roles that we had in the input:

  - Only two properties are specified: We establish via the local conditions that the range of both the properties has to be compatible. On the other hand, we let the result of the comparison be the union of both domains. We assume that the rest of the query constrains the resulting set.
  - Two concepts and a property: We are comparing the instances of two different concepts according to the value of a property. Thus, we establish that the property has to be *applicable*[9] to both concepts (local condition). In this case, the resulting concept is the union of both compared concepts.
  - Two concepts and two properties: The comparison is made on an arbitrary property of each of the concepts. This case is the

[9] We consider a property *applicable* to a concept when its domain its not disjoint with it. We check it by testing the satisfiability of their intersection.

combination of the two previous ones, as the ranges of the properties have to be compatible, and the properties applicable to its corresponding concept (local conditions). As with the previous one, the resulting set is the union of the participating concepts.

Once the query has been semantically checked, we let the responsibility for its correct translation into the actual language syntax to the Adapter implementation. In the following subsection, we give a complete example of QueryGen using LOQOMOTION as backend.

6.3.5 A Complete Example with LOQOMOTION as Backend System

In this section, we present a synthetic example to illustrate the keyword interpretation process with LOQOMOTION as target information system. In this case, we focus on how QueryGen is able to work with location-dependent operators such as *Inside* operator adding the semantics of the granules. Let us take a user that wants to monitor information about buses nearby Zaragoza. Thus, to do so, s/he inputs the keywords "Zaragoza bus passengers location".

The model for the location granules that we are using is the one shown in Figure 6.5. To populate it, we could easily reuse the information available in several Linked Data initiatives such as Linked GeoData [SLHA12], GeoLinkedData [VBVTS+10], or GeoNames[10]. Thus, we extend the model by adding the subconcept *City*, that is subsumed by *Granule*, and we group the cities of Spain under a mapping that is identified by *SpanishCities*.

1. In the disambiguation process, our system offers the user several interpretations for each keyword:
   - For "Zaragoza", the proposed meaning is the instance of *City*.
   - For "bus", one of the proposed meanings is the concept Bus from the OWL version of *OntoSem* ontology [BGK06].
   - For "passenger", one of the proposed meanings is the homonym property from *OntoSem*.
   - For "location", one of the proposed meanings is the integrated sense of the location property from the DBpedia ontology, *schema.org* and *OntoSem*.

[10] `http://www.geonames.org/`, last accessed October 3, 2013.

2. In the generation process, at first, our system cannot build the intended query because it needs an extra instance to assign Zaragoza the correct granularity interpretation according to the *SpanishCities* mapping. As we have seen in Section 5.4.2, despite the fact that instances were not considered from the beginning, our system can add a VT of instance type. This allows the system to find out the query intended by the user:

   $[location,\ passenger]\ Inside(GrG(Zaragoza,\ SpanishCities),\ Bus)$

   which would retrieve the location and the passengers of the buses that are inside a radius from Zaragoza considered as a *City*.

During the generation, QueryGen checks that Zaragoza is an instance of *Granule*, and that *SpanishCities* is an existing *GrMap* thanks to the local conditions established in the language definition. Moreover, *passenger* and location are projected because they are semantically applicable. However, the responsible for the modeling of the scenario has to take into account the need of a correct alignment between the properties defined in the ontologies used for the disambiguation and the properties defined in the LOQOMOTION scenario. Otherwise, the Adapter has not enough information to know how to access the data.

6.4 Summary of the Chapter

In this chapter, we have explained the solution adopted to attach different data models to QueryGen. Thanks to the use of *Adapters*, QueryGen can forward the selected query to the appropriate underlying information system, adapting both the data model and the query execution model. These *Adapters* are an evolution of the *wrappers* proposed in OBSERVER [MI01], which adapted the queries to the different answering capabilities of the data repositories. In particular, the main capabilities that our system inherits from OBSERVER are the data integration capabilities and the ability of processing incomplete queries.

Then, we have presented two successful use cases that have been implemented and registered into QueryGen: DBpedia (its SPARQL endpoint) and LOQOMOTION.

- The first case provides a complete example of a semantic data access pipeline, going from plain keywords to the semantic data available in DBpedia in the form of Linked Data.

- The second case provides an example of how a system that, in principle, did not take into account any kind of semantics can be adapted and added to QueryGen to perform a semantic search on it.

Last but not least, in this chapter, we have introduced two different semantic models for representing location granularities. They are complementary, as they are oriented for different tasks: The former is aimed at extending the capabilities of location dependent queries through the addition of semantics to location constraints [BIM10b], while the latter is aimed at providing automatic reasoning on the location model [BBMI13]. In fact, the second model can be used to feed the first one.

Chapter 7

Experimental Results

In this chapter, we present an evaluation of our approach from different points of view. We start by presenting a qualitative evaluation of the whole process, accessing data from DBpedia, and analyzing the current advantages and shortcomings that we have detected. Then, we analyze the performance of the different steps of the approach and the reduction rate achieved by the reduction techniques that QueryGen applies in the query generation process.

7.1 Evaluating the Semantic Capabilities of QueryGen

To evaluate the semantic capabilities of our system, we have performed two different evaluations. First, we have focused only on the quality of the discovery of the user's intended meaning, regardless of the source ontologies for the different meanings. Secondly, we have turned our focus on the evaluation of our current prototype accessing data from DBpedia, which has brought up several issues that conform the main lines of future work in this thesis.

In the rest of the section, we present firstly the query set that we have selected, and then we present and analyze the results of both performed evaluations.

7.1.1 Selected Query Set

To evaluate qualitatively the semantic capabilities of QueryGen, we considered at first two different query sets that are currently being used in different search contests:

- Text REtrieval Conference (TREC)[1] - Web Track

  The Web Track of this conference focused on Information Retrieval evaluation was discontinued in 2004, and it returned in 2009. In order to assess the quality of different Web Search engines, they provide a set of packages of 50 queries[2] expressed in keywords. Each of these queries has several retrieval tasks associated to it.

- Query Answering over Linked Data (QALD)[3] [LUCM13]

  This contest/track is more recent than the TREC one, and it focuses on Linked Data resources. It is used to assess natural language interfaces and different keyword interpretation techniques over Linked Data, and up to now, there have been three contests held on a year basis. It considers DBpedia along with a RDF export of MusicBrainz[4] as data sets, and provides 100 queries with the expected results for each of them. Excepting the first year (QALD-1 query set), the queries are expressed both in natural language and keywords as well. QALD-2 and QALD-3 are equivalent, as the latter is just an extension of the former to address multilingual issues. We selected the set for the 2013 contest, QALD-3, to perform the evaluation, and in the following we will refer to it as the QALD query set.

We discarded the first one as, in fact, the keyword sets that TREC provides for the Web Track are not oriented to search on structured information, thus, the semantic gap from the expressed keywords to the real meaning is huge. One could argue that this is the kind of input you have to expect from the users, but this gap seems to be the result of the users' training over the years, who have learnt that Web Search engines do not retrieve what they were looking for, and therefore, they just relax their inputs to begin with the navigation on the first results as soon as possible [MMZ09]. Thus, we focused on evaluating our approach against the second query set.

Moreover, in the QALD query set, for each query, they provide a SPARQL query that expresses the exact semantics corresponding to the input (natural language or keywords). Thus, it provides a mean to compare our results objectively. To properly evaluate the results of each of the steps of QueryGen,

[1] `http://trec.nist.gov/`, last accessed September 30, 2013.

[2] There is one package for each of the years that this track has been held, this is, 200 queries considering the 2009-2012 conferences.

[3] `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/`, last accessed October 3, 2013.

[4] `http://musicbrainz.org/`, last accessed October 3, 2013.

human assessment was required to analyze the intermediate data generated. This process is quite costly, so, to alleviate the work load of analyzing each of them step by step, we decided to select 25 queries randomly out of the 100 total ones of the QALD query set over DBpedia. The same set of queries has been used for both performed evaluations (semantic capabilities and prototype one).

7.1.2 Evaluation of Discovery of Users Intended Query

The objective of this evaluation is to assess the semantic capabilities of QueryGen to discover the meaning of the input keywords and achieve their correct interpretation comprising the user's information need.

To ease the explanations, despite the fact that QueryGen is able to use different query languages to interpret the input, we selected the simplified BACK language extended with: 1) the inverse version of *Some* and *Fill* operators, and 2) aggregation operators. We have used a controlled set of 55 ontologies to be able to trace and repeat the experiments (the test collection OWLS-TC4[5] plus the ontologies *dbpedia_3.6.owl*[6], *schema.org*[7], *People+Pets*[8], *Koala*[9], *Animals*[10], and WordNet)[11].

In this test, we consider a success if QueryGen obtains a query that expresses the same semantics as the attached SPARQL query. In Table 7.1, we show the quantitative information about the results: QueryGen achieves the correct interpretation in a 76% of the cases. The reasons for not achieving a suitable query for the input keywords were: 1) the lack of proper knowledge in the provided ontologies (5 out of 6 cases), and 2) the lack of expressivity of the language (3 out of 6 cases).

In the following, we detail 7 of the 25 cases (see Table 7.2) which cover the different issues that we have found performing the test:

- For the input keywords "female Russian Astronaut", our system maps: *astronaut* to the concept from DBpedia, among other sources (merged); *Russian* to the concept/instance from WNet, offering also Russia as an instance of country; and *female* to the instance in *Animals*, among

[5] http://projects.semwebcentral.org/projects/owls-tc/

[6] http://downloads.dbpedia.org/3.6/dbpedia_3.6.owl

[7] http://schema.org/docs/schemaorg.owl

[8] http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/people+pets.owl.rdf

[9] http://protege.stanford.edu/plugins/owl/owl-library/koala.owl

[10] http://www.cs.man.ac.uk/~rector/tutorials/Biomedical-Tutorial/Tutorial-Ontologies/Animals/Animals-tutorial-complete.owl

[11] All URIs accessed on October 3, 2013.

Intended Query	**Number of cases**	**Rate**
Generated with 0 VTs	10	76%
Generated with 1 VT	7	
Generated with 2 VTs	2	
Not Generated	6	24%

Table 7.1: Success rate of QueryGen against the QALD excerpt.

Question	**Keywords**	**Query**	**VTs**
Give me all female Russian astronauts	astronaut, female, Russian	✓ Astronaut And (Fill(birthPlace,Russia)) And (Fill(gender, female))	2 roles
Which river does the Brooklyn Bridge cross	Brooklyn Bridge, river, cross	✓ River And Inv-fill(crosses, BrooklynBridge)	0 VTs
How many monarchical countries are there in Europe?	Europe, monarchical country	✗ no query (due to lack of knowledge)	
Which countries have places with more than two caves?	cave, country, place, two	✗ no query (due to lack of knowledge and expressivity)	
Which mountain is the highest after the Annapurna?	mountain, highest, after, Annapurna	✗ no query (due to lack of expressivity)	
Which telecommunications organizations are located in Belgium?	Belgium, telecommunications, organization, located	✓ Organization And Fill(location,Belgium) And Fill(industry, Telecommunications)	1 role
Give all the movies with Tom Cruise	Tom Cruise, movie	✓ Movie And Fill(starring, Tom Cruise)	1 role

Table 7.2: Question, input keywords, generated query, and number of VTs needed for an excerpt of the QALD questions.

other sources (merged). With two extra roles, QueryGen can build *Astronaut And (Fill(birthPlace, Russia)) And (Fill(gender, female))* which is the intended query.

- For the input keywords "Brooklyn Bridge, river, cross", our system maps: *cross* to `SUMO:crosses` and `dbpedia:crosses` (due to the difference in the domain and range of the properties, our system does not assume that are equivalent), and *River* to `dbpedia:River`. With those terms, and treating *BrooklynBridge* as an instance, QueryGen can

build the intended query *River And Inv-fill(crosses, BrooklynBridge)* with no extra term.

- For the input keywords "Europe monarchical country", our system cannot find the intended query. Although *Europe* is well disambiguated, there is no way to establish in the DBpedia ontology that a country belongs to *Europe* (the continent). This fact is stated via `yago:EuropeanCountries`, which is out of the knowledge consulted for disambiguating purposes[12]. *Monarchical* is mapped to `WNet:Monarchy`, and, enriching the input with one extra role, QueryGen would be able to add `dbo:governmentType` to achieve the intended query.

- For the input keywords "cave, country, place, two", *cave*, *place* and *country* are mapped to their homonym concepts in the DBpedia ontology (among others). The keyword *two* is considered to be an integer. In this case, QueryGen is not able to find the intended query as there is no property in the consulted knowledge that expresses that a place is the location of something (we would need the inverse of `dbprop:location`). Provided that such a property would have been obtained, QueryGen could build the query if it had the *At-least* operator, which is very common in DL query languages.

- For the input keywords "mountain highest after Annapurna", our system has not been able to achieve the intended query. Although it disambiguates *mountain* mapping it to `dbpedia:Mountain` (among others), and it also detects the fact that *Annapurna* is a particular instance of `dbpedia:Mountain`; the "highest after" keywords are not correctly interpreted: They should be mapped to a particular operator that does not exist in the selected query language.

- For the input keywords "Belgium, telecommunications organization, located", our system maps: *telecommunications* and *Belgium* to instances in WNet; *organization* to its homonym in the DBpedia ontology (among others); and *located* to `protont:LocatedIn` property (equivalent to `dbprop:location`). With one extra role (in this case, *industry* from the DBpedia ontology) our system is able to build *Or-*

[12]We discarded using YAGO as a disambiguation source in this test due to their complex categories, which introduce ambiguity. For example, concepts such as `yago:PresidentsOfTheUnitedStates` or `yago:SchoolTypes` are especially difficult to handle, as explicitly stated by the QALD authors in [LUCM13]. Nevertheless, we are working on how to introduce this kind of sources in QueryGen without being affected by the noise.

ganization And Fill(location, Belgium) And Fill (industry, Telecommunications), which is the intended query.

- For the input keywords "Tom Cruise, movie", our system considers *TomCruise* altogether as a single instance and it maps *movie* to `schema.org:Movie`, among other sources (merged). Thus, QueryGen can build with one extra term (*starring* from the DBpedia ontology) the query *Movie And Fill (starring, Tom_Cruise)*, which is the intended query.

Evaluation Conclusions In spite of the lack of knowledge and expressivity of the query language selected, QueryGen has been able to generate the intended queries with exactly the semantics needed for most of the queries:

- The disambiguation process has been successful almost in all the cases, detecting correctly the meanings, and merging and enriching the senses of the keywords.
- The generation process reaches the user's intended query whenever the expressivity of the query language allows to build it (76% of the cases for the query language selected in the evaluation).
- The usage of virtual terms to discover implicit meanings is present in the 47% of the queries that we have been able to generate with our system. The rest of the successful queries have been generated with 0 virtual terms.
- Anyway, there are still times when QueryGen fails to generate the intended query (6 out of the 25 selected queries) due to two main reasons: 1) the lack of knowledge, which could be addressed by upgrading the dynamic pool of ontologies; and 2) the lack of expressivity of the query language selected for the test, which can be addressed by adding languages expressive enough to express the user's query.

Thus, the flexibility of QueryGen regarding the sources of knowledge consulted and the query models supported makes it possible to alleviate the problems detected in the evaluation. On the one hand, the more ontologies available in the pool of ontologies consulted by QueryGen, the more semantic interpretations can be considered. On the other hand, despite we have performed this test with just one selected query language, notice that QueryGen searches for the intended query taking into account all the available languages: For a given input it could express the intended query in

some query language, for the next input another query language could be used, automatically.

7.1.3 Evaluating QueryGen Accessing Data: From Keywords to Data in DBpedia

Now we turn our attention into evaluating QueryGen accessing DBpedia using the Adapter presented in Section 6.2.1. The set of consulted ontologies in the disambiguation process is the same as in the previous evaluation.

While in the previous evaluation any discovered meaning would suffice to discover the user's intended meaning (regardless the source ontology), when it comes to accessing data in a particular data repository, we need a mechanism to reconciliate the discovered meanings with the vocabulary used in such a repository. Currently, as stated in Section 6.1, we rely on the Adapter to do this task (we consider this as one of the main points of our future work).

Moreover, we have to bear in mind that the landscape of ontologies and semantic data has evolved dramatically in the last few years. The explosion of Linked Data has raised the need of considering the data behind SPARQL endpoints (such as DBpedia's one) to perform the keyword disambiguation; and, thus, taking into account the instances associated to each of the concepts in the ontologies. This is not currently addressed by our prototype, as the disambiguation is performed on the retrieved ontologies, a set of statically provided ontologies, and WordNet. The addition of SPARQL endpoints to enhance the disambiguation is part of the ongoing work of this thesis.

Thus, we have evaluated the approach by assuming that DBpedia's SPARQL endpoint has been correctly added to the disambiguation process (we could add it using several SPARQL templates to obtain the information about the involved resources, or even use directly the *lookup service*[13] that DBpedia provides us with).

In the following, we firstly detail 5 of the 25 cases (out from the same query set as in the previous section) focused on the access to DBpedia, and then we present the conclusions about our prototype, which mark different lines of future work. The queries are as follows:

- For the input keywords "female Russian Astronaut", as we have seen before, our system maps: *astronaut* to the concept from DBpedia, among other sources (merged); *Russian* to the concept/instance from WNet, offering also Russia as an instance of country; and *female* to the

[13] `http://wiki.dbpedia.org/lookup/`, last accessed October 3, 2013.

instance in *Animals*, among other sources (merged). With two extra roles, QueryGen can build *Astronaut And (Fill(birthPlace, Russia)) And (Fill(gender, female))* which is the intended query.

This query is posed to the corresponding Adapter, and DBpedia accessed. The results are not the expected, because the gender of the different people is not asserted in DBpedia, as we can see if we pose the following query:

$$[birthPlace,\ gender]\ Astronaut$$

```
PREFIX dbpedia:<http://dbpedia.org/ontology/>

SELECT DISTINCT ?pr_birthplace_0 ?pr_gender_1 ?id0
  WHERE  { { ?id0 a dbpedia:Astronaut . }
    OPTIONAL { ?id0 dbpedia:birthPlace ?pr_birthplace_0 . }
    OPTIONAL { ?id0 dbpedia:gender ?pr_gender_1 . }  }
```

A solution for this could be to include the instances of DBpedia in the disambiguation process via the lookup service. Another way to improve the disambiguation process would be to add YAGO ontology, but its special characteristics (automatically built, composed out of 400.000 concept/instances) would introduce noise in our system. However, we will study how to use both the ABox of the DBpedia and the YAGO ontology itself.

- For the input keywords "cave, country, place, two", *cave*, *place* and *country* are mapped to their homonym concepts in the DBpedia ontology (among others). The keyword *two* is considered to be an integer. In this case, QueryGen is not able to find the intended query as there is no property in the consulted knowledge that expresses that a place is the location of something (we would need the inverse of `dbprop:location`).

  That is, we would need: On the one hand, aggregation operators to express the "more than two" clause; and, on the other hand, an inverse existential operator that would give us the instances of their range that are related (being the object) to an instance of the concept (which should be a subclass of the property domain). This is needed due to the lack of an inverse property for `dbprop:location`, if there were

such as property (e.g., isLocated), we wouldn't need this operator to build this query.

- For the input keywords "Yenisei, river, flow, through, country" representing the query "Through which countries does the Yenisei river flow?", QueryGen maps correctly all the keywords to terms in several ontologies, but *flow* one. It is mapped to a concept, while the property that we were looking for querying correctly DBpedia is `dbpedia:country`. This property is hidden by the fact that we are looking for countries and the user might choose *Country* as concept instead of choosing the `dbpedia:country` property as it is not intuitive.

  As in the previous query, an inverse of one of the operators is needed (in this case, the *Fill* operator) to get the instances that are related to the Yenisei river via the inverse property (which in this case does not exist in DBpedia).

- For input keywords "school, type" meaning "Give me all school types", our system obtains `dbpedia:School` and the property `dbprop:type` from the DBpedia ontology. These meanings (among others) do not allow our system to generate the desired query because of their semantics and how they are used in the DBpedia dataset. Although is quite counterintuitive, is not possible to access from the *School* concept, to the *SchoolTypes* concept. This is due to the fact that the YAGO's concept comes from the SKOS categorization of the Wikipedia articles, which is left aside in the DBpedia ontology (as it is explained in Section 6.2, more details can be found in [BEM13]). YAGO is built extracting automatically all the information and introduces a lot of *semantic noise*. This case is a good example of it: Instead of classifying the different schools as subclasses of *School*, YAGO enumerates the types of school that there are, as directly extracted from the SKOS taxonomy of article's subjects.

  This query could be directly answered if we added the YAGO taxonomy to our system, considering the fact that both keywords were mapped just to one concept: `yago:SchoolTypes`.

- For the input keywords "Tom Cruise, movie", as we have seen before, our system disambiguates correctly *Movie*, discovering and merging the senses from different ontologies. However, it is not mapped to be equivalent to the DBpedia's *Film* concept, in spite of being able to

find the synonym *Film* for *Movie* with the extraction techniques in other ontologies. So, when our system, with an extra term, builds:

$$Movie\ And\ Dill(starring,\ Tom\_Cruise)$$

it does not retrieve the desired results. If we had input just Tom_Cruise in our system, it would have retrieved results, as the Adapter for DBpedia adds the domain of the property and builds:

```
PREFIX dbpedia:<http://dbpedia.org/ontology/>
PREFIX dbres:<http://dbpedia.org/resource/>

SELECT DISTINCT ?id0
  WHERE  { ?id0 a dbpedia:Work .
           ?id0 dbpedia:starring dbres:Tom_Cruise . }
```

which retrieves all the movies that are starred by Tom Cruise. If the *Film* concept would have been correctly evaluated as equivalent to `dbpedia:Movie` (recall that our system disambiguates and merges it correctly), the Adapter would have added an extra constraint that refines further the results (we have checked the results of adding "id0 *a* `dbpedia:Film`" to the query by hand).

Evaluation Conclusions Analyzing the results of the prototype accessing DBpedia, we detected several issues that our current prototype does not address and that affects the answering capabilities of our system, which might constitute the main lines of future work. We detail them in the following, ordered by importance (number of queries affected by the issue):

- Multiple keywords must be mapped to just one term: It is usual to have several keywords to be mapped to just one term or resource. For example, keywords *Yenisei* and *river* should be mapped together to the resource *Yenisei_river*, or *school* and *type* mapped to YAGO's concept *SchoolType*.

- Need to run the discovering and disambiguation process on DBpedia's resources: We have to take into account the new landscape of semantic data regarding the disambiguation process and include DBpedia's

resources (in fact, we have to be able to include any SPARQL endpoint). This would allow, in addition to the previous point, to detect the resource *Yenisei_river* or *Tom_Cruise* directly in DBpedia.

- Lack of expressivity of simplified BACK: The query language we used for the evaluation lacked some needed operators such as aggregation ones, operators for answering yes/no questions, etc. However, as we have seen, the addition of new operators comes at a cost, so this aspect would have to be carefully evaluated. For example, the *more than two* clause in the query "which countries have places with more than two caves?" implies being able to apply an aggregation operator (in this case, *Count*) on the intermediate results.

- Ambiguous names of properties: There are times that the keyword query itself includes keywords that are too far from their actual meaning to be mapped to it in the disambiguation. For example, *country* is used in DBpedia to denote the fact that a river flows through a country, while *Country* is used to denote the concept country. Both of the meanings are part of the considered query, it is quite difficult to infer that *flows* keyword should be mapped to *country*. We find another example when considering *married* vs *spouse*: DBpedia uses the latter one to express the relationship of being married. We have to advance further in the disambiguation to be able to stretch the semantic gaps.

- Not enough data/knowledge to answer the query: DBpedia itself might have not have enough information to be able to answer the posed query. This can happen at schema (properties that are not defined), and at data level (data that is missing). We can see these missed data in, among others, the query of the female astronauts, where their gender data is not stated in DBpedia.

- Use of a keyword/property to name its inverse one: Users might unawarely use a keyword to name the inverse property (which might not even exist in the consulted ontologies). This could be tackled by adding inverse operators as we have seen in the query examples (in fact, this issue is subsumed by the lack of expressivity of the selected language, but we think it is important enough to be pointed out). In the query of the caves, as DBpedia only offers *location* as property (and not *isLocated*), we would need to apply the inverse of the *Some* operator to obtain the desired semantics.

- Redundant keywords: The behaviour of current keyword search systems has modified how users express their keyword queries [MMZ09], and it is quite usual to have redundant keywords in the queries, such as an instance and its class (*Yenisei* and *River*), or two classes that one subsumes the another one (*Place* and *Cave*).

- Keyword must be mapped to an operator or to a datatype value rather than to an ontological term: We have to be able to detect when a keyword/set of keywords have to be mapped to a particular operator, or a datatype instance. This would affect directly to the query generation process, as we could force the usage of a particular operator, narrowing the interpretation space. In the example of the caves, our system should have disambiguated *two* as a number, and therefore only consider the operands that would take a number as operand (this currently can be restricted by giving the corresponding local conditions on the operators).

Note that these are current limitations of our prototype that we have detected while accessing DBpedia. However, assuming that the two first points were correctly addressed, we have been able to generate the proper queries with exactly the semantics needed for most of the queries. So, there exists a long road for improvements in our approach, but we think we are on the correct way.

7.2 System Performance

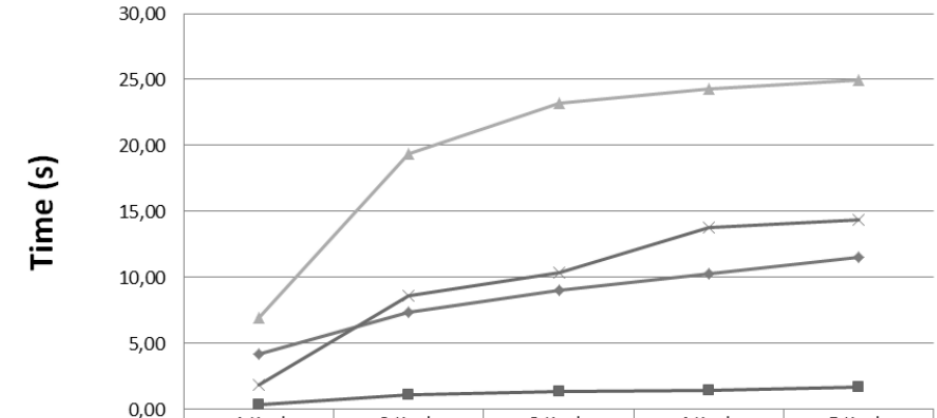
In this section, we focus on the performance of the different steps of our system. On the one hand, we measure the performance of the disambiguation process taking into account different depths when comparing ontological contexts. On the other hand, we evaluate the impact of the reduction techniques we apply during the semantic query generation; and, finally, the performance of the generation step along with the semantic filtering (it includes both the local and global checks).

7.2.1 Keyword Disambiguation Performance

To test the feasibility of our disambiguation techniques, we have performed a set of tests to evaluate its performance in a detailed and systematic way. The tests were executed on a Sunfire X2200 (2 x AMD Dual Core 2600 MHz, with 8GB RAM). We have used the same set of ontologies as in the previous

section (the test collection OWLS-TC4 plus the ontologies *dbpedia_3.6.owl*, *schema.org*, *People+Pets*, *Koala*, *Animals*, and WordNet). Thus, a total of 55 ontologies were consulted by our prototype.

The input keywords were not selected randomly but based on actual queries proposed by students of different degrees with skills in Computer Science. We considered fifty sets of input keywords to perform the tests, ten for each number of keywords. In Figure 7.1, the results for different sizes of the inputs are shown.



	1 Kwd	2 Kwds	3 Kwds	4 Kwds	5 Kwds
Depth 1	4,22	7,35	9,01	10,26	11,51
Depth 1 - cached	0,35	1,08	1,31	1,39	1,65
Depth 2	6,92	19,38	23,23	24,25	24,99
Depth 2 - cached	1,83	8,58	10,38	13,79	14,36

Figure 7.1: Keyword disambiguation performance evaluation.

As it can be seen, the disambiguation times depend on which depth is considered for matching (i.e., how many levels of parent and children terms in the ontological context). From the experiments, we have seen that using a depth greater than two lead to wrong results. This is due to the fact that the closer you get to the TOP concept in the ontologies, the more false positives appear, as too general subsumer terms are considered. So, a depth of two levels is considered to be semantically optimal. The cached results corresponds to executions on which the extraction procedures had been already performed and stored, as at first, it was the most expensive task.

7.2.2 Evaluation of Query Generation

We turn our focus now on the performance of the query generation step. The tests have been carried out using Pellet[14] 1.5 as background DL reasoner. They were performed with the same settings, that is, on a Sunfire X2200 (2 x AMD Dual Core 2600 MHz, with 8GB RAM). For the sake's of experiments repeatability, we selected two well-known ontologies: *People+Pets* and *Koala*. They are two popular ontologies of similar size to those used in well-known benchmarks such as the OAEI[15]. We only show the experimental results obtained with simplified BACK as output query language because most search approaches are based only on conjunctive queries. Nevertheless, we have also performed the experiments with another non-DL languages, and we obtained similar execution times and conclusions.

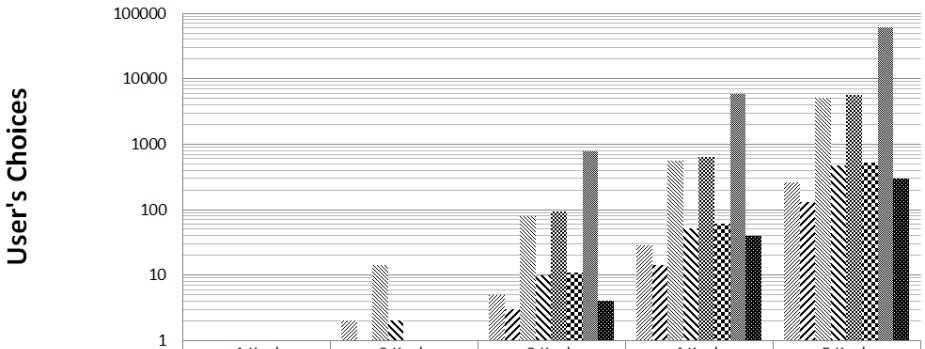
For the experiments, we considered different sample sets of input keywords (selected from the terms of the above ontologies) and measured average values grouped by the number of keywords in the set. As in the evaluation of the performance of the disambiguation process, these inputs were based on actual queries proposed by students of different degrees with skills in Computer Science. The sets were chosen according to the following distribution: 10 sets with a single keyword (5 selecting a role and 5 selecting a concept), 15 sets with two keywords (5 sets where both keywords are roles, 5 sets where both keywords are concepts, and 5 sets where one keyword is a role and the other one is a concept), 20 sets with three keywords (5 with 2 concepts and 1 role, 5 with 1 concept and 2 roles, 5 with 3 concepts, and 5 with 3 roles) and, following the same idea, 25 sets with four keywords and 30 sets with five keywords. Notice that, even though our approach can effectively deal with instances as well, we do not consider sets with instances because the selected ontologies do not have instances (as it happens frequently [WPH06]). We set the maximum number of keywords to 5, as the average number of keywords used in keyword-based search engines "is somewhere between 2 and 3" [MRS08], and thus we can see how our system performs with inputs below and above this average number of keywords.

We conducted four experiments: 1) no VTs added, the system works only with the user keywords; 2) one VT added, to try to find a possible missing keyword; 3) two VTs (1+1), is the same situation as 2) with an extra refinement step once the user has selected a candidate for the first VT to be rendered; and 4) two VTs added, to find two possible missing

[14] `http://clarkparsia.com/pellet`, last accessed October 3, 2013.
[15] `http://oaei.ontologymatching.org/`, last accessed October 3, 2013.

keywords at the same time[16]. We have also considered that the user inputs at least one keyword. The X-axis in Figures 7.2 and 7.3 represents the total keywords considered, i.e., the input and the VTs added by the system. Thus, considering 3 keywords, the results are for 3 user keywords (no VTs), 2 user keywords and 1 VT (one VT), and 1 user keyword and 2 VTs.

Figure 7.2 shows the average number of generated queries and the average number of patterns that are presented to the user (notice that the Y-axis is in log scale).



	1 Kwd	2 Kwds	3 Kwds	4 Kwds	5 Kwds
No VTs (#queries)	1	2	5	28	258
No VTs (#patterns)	1	1	3	14	129
One VT (#queries)		14	79	552	5073
One VT (#patterns)		2	10	51	471
Two VTs (1+1) (#queries)			92	630	5624
Two VTs (1+1) (#patterns)			11	60	521
Two VTs (#queries)			787	5754	61163
Ttwo VTs (#patterns)			4	40	299

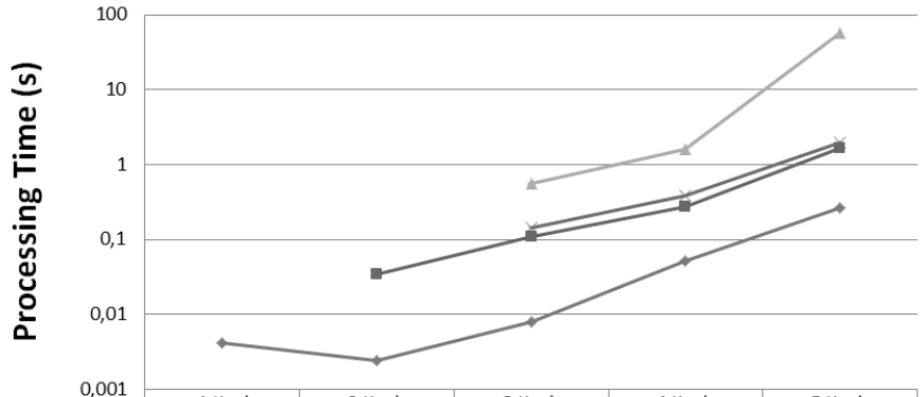
Figure 7.2: Performance evaluation: average number of queries and shown patterns.

As expected, the number of queries generated rapidly increases with the number of input terms, as the more operands there are, the more queries can be built. Moreover, performing the semantic enrichment leads also to a significant increase in the number of queries because many new interpretations appear. However, the use of query patterns reduces up to an average 92% the options that the user is presented with. Figure 7.2 also shows that, despite generating a higher number of queries, the system compresses the queries more when it has two VTs at once than in the other situations. This may be beneficial to the user, but it might require her/him more time navi-

[16]We do not consider adding more than 2 VTs because we do not aim at discovering the user's intended query when too many keywords were missed in the input.

gating through the candidate keywords for the VTs. The number of queries is lower for two VTs (1+1) as, in the refinement step, the user has fixed a VT and there are less options. Last but not least, no possible interpretation (according the query language) is discarded.

Finally, the average times that the generation process takes are shown in Figure 7.3. They include the generation and the semantic filtering time. Being the low they are makes the system suitable to be a responsive front-end (note that we would have to add the times of the sense discovery module, but this module can be use in a standalone mode as well). As it can be seen in Figure 7.3 (notice that the Y-axis is in log scale), the average times for 3 and 4 keywords are similar and really low (recall that the average number of input keywords was between 2 and 3).



	1 Kwd	2 Kwds	3 Kwds	4 Kwds	5 Kwds
No VTs	0,004	0,002	0,008	0,052	0,262
One VT		0,035	0,108	0,274	1,688
Two VTs (1+1)			0,143	0,382	1,962
Two VTs			0,565	1,604	56,307

Figure 7.3: Performance evaluation: processing time of the query generation step.

7.3 Summary of the Chapter

In this chapter, we have analyzed QueryGen in a qualitative and a quantitative way. Firstly, adopting a standard that is being used to query DBpedia from keywords, we have evaluated the semantic capabilities of our approach discovering the user's intended meaning. Secondly, we have evaluated our system working with the DBpedia Adapter, which uses our simplified ver-

sion of BACK as query language. The results of both evaluations show the potential of our techniques. Moreover, the analysis of the queries that presented problems has raised several issues that, far from being a dead end for the approach, will guide our future work regarding keyword interpretation. In particular, exploiting the knowledge in Linked Data repositories and detecting the operators to be used are specially well positioned to be good lines of improvement.

Then, we have moved into performance related aspects of the system. At first, the disambiguation procedure might be seen as quite expensive so as to be used in a system with user interaction. Thanks to the inner structure and the reimplementation of several parts of the original code, we have managed to lower the times while not compromising the quality of the keyword disambiguation. Regarding the query generation, our system presents a good performance. In particular, the inconsistent query filtering is fast enough thanks to the fact that, once we have the original ontology classified, the reasoners can assess the satisfiability of the expressions without reclassifying this ontology.

Finally, the impact of the reduction techniques has also been evaluated. The results show that the reduction rates that are managed reduce the possibilities presented to the user dramatically, which is an remarkable achievement as it is done without losing any possible interpretation.

Chapter 8

Conclusions

In this chapter, we present different conclusions about our work. Due to the broad aspects dealt with during the thesis' period, first, we present the main contributions of QueryGen as a whole system, and then, we present the secondary contributions to different fields such as location-dependent queries. After this, we present the publications related to the work presented in this thesis, analyzing their quality according to different quality index rankings. Finally, we are aware that there is plenty of work ahead and, so, we present some future work.

8.1 QueryGen: Main Contributions

In this thesis, we have presented QueryGen, a system that enables *semantic keyword-based search* over heterogeneous information systems. We have explored the problem of keyword interpretation in depth, providing a solution that, exploiting the semantics of all the elements that participate all along the search process, is flexible enough to deal with different data schemas (ontologies), different query languages, and different execution semantics.

We have got rid of the ties that other keyword interpretation approaches have to the specific data model that they are accessing. However, flexibility comes at the cost of having to explore a bigger search space to find the intended interpretation; but, thanks to several semantic techniques that we apply, we also reduce dramatically this search space without losing any possible query. Moreover, the visualization technique we use further reduces the efforts that the user should make to select their intended query.

We have done an initial step towards the modeling of query languages in DL, making it possible to establish the consistency of a query accord-

ing to the knowledge stored in an ontology even with non-DL languages, such as our extended SQL-like language for location-dependent queries. Using QueryGen, users can turn their information needs into formal queries without having to master the formal languages which they are written in. Having formal queries instead of information needs removes the ambiguity, and enables the systems to focus on answering the specific query that users would have posed if they knew how to write it.

QueryGen automatically accesses the information systems which are capable of answering the selected query, transparently to the user, and giving a single entry point to a heterogeneous set of systems. In this access, not only the data is integrated, but the execution semantics, integrating also continuous queries.

Main Features of QueryGen

Summing up the whole system, the main features of QueryGen are:

- The discovery of the meaning of the input keywords is done by consulting a dynamic pool of ontologies. This allows QueryGen to handle very different domains, and not being constrained to a fixed source of semantic information. During this process, our system merges the meanings that are considered similar enough and proposes the most probable semantics for each of the input keywords. To do so, it performs a disambiguation process that takes into account the semantics of all them as a whole.

- Regarding the disambiguation process and the posterior ontology integration, we have introduced an intelligent sense library whose goal is two-fold: 1) to speed-up this process, achieving execution times that makes it possible to use the system in real-time; and 2) to manage the evolution of the source ontologies and adapt QueryGen to the possible changes in the ontologies, and therefore, in the schemas describing the underlying data sources.

- Our approach performs the keyword interpretation via a query generation process independent of the available query languages. In this interpretation process, our system can handle any associated query language whenever it is specified via a semantically annotated grammar. Moreover, it takes into account the semantic properties of the query languages to avoid generating semantically equivalent queries. Finally, the user is not aware of the different underlying query models,

as the system performs the generation in all the available ones (they correspond to different data repositories).

- With the help of a DL reasoner, it is able to filter out inconsistent queries according to the knowledge retrieved. This is not only applicable to DL queries, but also to non-DL languages, which makes our approach very flexible. This filtering is greatly boosted by the fact that the set of generated queries are a conservative extension of the source ontologies, and therefore, their satisfiability can be assessed without having to reclassify the knowledge.

- It performs internally a semantic enrichment of the input to fill the possible gap between the user keywords and the user's intended query. This is done by using *virtual terms*, which allows the system to explore further meanings when the user's input is incomplete. To render them, the system considers the semantic information dynamically obtained and integrated during the disambiguation process.

- The process is independent of the underlying data models and makes the access to them transparent to the user, providing a unique point of entry to heterogeneous systems. In our prototype, we have successfully integrated the system to query, using plain keywords, two very different information systems such as LOQOMOTION and DBpedia (its SPARQL Endpoint).

However, the contributions of this thesis are not limited to the keyword search field. The different lines of research of our group and the know-how acquired developing the different modules of QueryGen, have allowed to make contributions to other different fields as we present in the following section.

8.2 Other Contributions

The work done during this thesis has also made contributions to other aspects apart from *semantic keyword-based search*. They are mainly related to the semantics of locations, and their influence on location-dependent queries and their processing. In particular:

- We have introduced the notion of location granule in the context of location-dependent queries, studying how their use affects to the semantics of different well-known constraints such as *inside* and *nearest*

neighbour queries. The use of location granules greatly increases the expressiveness of the location-dependent queries and the range of applications that can benefit from the query processing.

- We have extended LOQOMOTION with support for these location granules, achieving higher query expressivity while benefiting of the distributed query processing that LOQOMOTION provides us with. Due to the use of location granules it is now possible to define queries that otherwise would not be possible. Moreover, the advantages of their use do not come at the expense of performance (we performed an extensive experimental evaluation that can be found in [IBM11]).

- We have extended the notion of location granules to *semantic location granules*, where they become more than simple areas. For this purpose, we have proposed two complementary semantic models that fit different needs:

  - Modelling them as instances to exploit ABox reasoning: This model allows us to extend dynamically expressivity of the query language with the help of a DL reasoner and support for Horn rules.

  - Modelling them as concepts to exploit TBox reasoning: This model makes it possible to infer new knowledge with the help of a DL reasoner, while avoiding wrong conclusions due to the geographic inclusion of concepts. Moreover, it allows us to express non-geographical information about the different locations and reason about it along with the spatial information.

Far from being a handicap, the heterogeneity of the work developed has given me a broad view on information systems. In particular, working in the intersection of Mobile Computing and the Semantic Web (its associated technologies) has opened a new line of research close to semantic datastreams (to reconciliate the use of volatile knowledge with static/intensional one), which is one of our current works [BBIM13]. Moreover, we also have started a line of research on how to exploit semantics in mobile environments, which first step has been to study how to use DL reasoners in Android-based devices [YBE+13].

8.3 Evaluation of Results

In this section, we present the contributions of each of the publications related to this thesis, grouped by issue and providing several quality measures of each of them.

- Publications related strictly to QueryGen and semantic keyword-based search:
  - In [BTMB08], we presented our approach to keyword interpretation for the first time. Based on syntax-free grammars, we could build the desired query from selected ontological terms into any target query language. This conference was ranked as B in 2008 in CORE Conference Ranking, and 0.82 at CSCR (top 23% in Artificial Intelligence category). Currently, is ranked as CORE C.
  - In [BM10], we introduced the Multi-Ontology Sense Library in the context of the Semantic Web Services. Via the integration of senses, we reconciliated the different vocabularies used in the descriptions of the services provided by the service providers and the service requesters. The conference is ranked as CORE B, and has a CSCR score of 0.81 (top 43% in Architecture / Hardware category).
  - In [BTMI10], we extended the work in [BTMB08], redefining the approach to achieve an scalable solution, defining and taking into account the properties of the operators to avoid generating duplicated queries, and reducing the search space dramatically. Moreover, we introduced the pattern-based presentation. The conference is ranked as CORE A.
  - In [BMT12], we revisited the idea of integrating an ontology out from a definition of the domain expressed by keywords. This approach could be used as a starting point in an ontology-engineering process as it acts also as an ontology-search tool (it searches for meanings instead of plain keywords). The conference is ranked as CORE B.
  - Finally, in [BEM12] and [BEM13], we presented our approach to provide an ECA with semantic keyword search on Linked Data. The work in [BEM13] is an extended version of [BEM12] as we were invited to submit it to an special issue comprising the best papers of the conference. The conference is ranked as CORE B, and the journal is also ranked as B in the CORE journal ranking.

Moreover, we have submitted the work presented in this thesis to a special issue on Semantic Search of the Journal of Web Semantics [BM13], and it is currently under review.

- Publications related to Mobile Computing and location-dependent query processing:

  - In [IMB07] and [IBM11], we introduced the notion of location granules, as well as the implications and benefits of their use in location-dependent queries. While the conference is not ranked, the extended journal version is published in JSS journal, which had a JCR impact factor of 1.135 (top 33% in Computer Science, Theory and Methods category) in 2012. On the other hand, it is ranked as CORE A in CORE journal ranking, and it has a SJR impact factor of 1.164 in 2011 (top 15% in Computer Science Applications category).
  - In [ICBM09], we extended the semantics of location granule-based queries by adding new constraints (nearest neighbour) and new types of granules (probabilistic ones). The conference is ranked as CORE B and has a CSCR score of 0.79.
  - In [BIM10a], we explained how the use of mobile agents in our systems had helped us in the development of robust and adaptable distributed information systems. This conference is ranked as CORE B, and has a CSCR score of 0.81 (top 43% in Architecture / Hardware category).
  - In [BIM10b], we presented the first of the semantic models of location granules (modeling location granules as instances) and an extension of location-based constraints which had into account the semantics of the model to calculate the relevant objects. This conference is ranked as CORE B and has a CSCR score of 0.79.
  - Finally, in [BBMI13], we presented the other semantic model of location granules (modeling location granules as concepts), complementary to the one presented in [BIM10b]. This model allows the DL reasoner to make intensive use of the TBox to infer the containment relationships. We are currently working on both models. This journal has a JCR impact factor of 1.613 in 2012 (top 21% in Computer Science, Information Systems category) and is ranked as CORE A in CORE journal ranking. On the other hand, it has a SJR score of 0.933 (top 8% in Geography,

Planning and Development category, and top 12% in Library and Information Sciences category) in 2012.

There has been a quite extensive work in both Mobile Computing and Semantic Web fields. As we have said before, the result of this has been a new line of research close to semantic datastreams, to handle volatile knowledge with the help of DL reasoners. We have currently submitted our seminal work to the IJSWIS journal, and it is under review [BBIM13]. Moreover, we have also published a preliminary study on how to use DL reasoners on Android-based devices in the OWL Reasoner Evaluation Workshop (ORE) [YBE+13].

8.4 Future Work

Being the keyword-interpretation problem an ill-posed problem as it is, there are many lines of improvement that could be explored in the context of QueryGen. Out from the analysis of the results presented in Sections 7.1.2 and 7.1.3, we can point out the following ones as the most relevant:

Regarding the Discovery and Disambiguation Process

- We should study the possibility of mapping multiple keywords to just one term: It is usual to have several keywords to be mapped to just one term or resource, and in its current state, this issue is not addressed, which introduces noise in the keyword interpretation process.

- We cannot turn our back on the huge amount of information provided by DBpedia's resources, and the Linked Data repositories in general. We should include them as information sources for the discovery and disambiguation processes; however, this must be done carefully as we will not be longer dealing with intensional data, but extensional one, and therefore the volumes of information dealt with will grow heavily.

Regarding the Integration Process

- We should study the impact of the quality of the source ontologies used in the quality of the integrated ontology. To do so, we are planning to apply different quality measures that there exist for ontologies (there is no a single value that express how useful an ontology is) and study their influence depending on the tasks that the integrated ontologies are oriented to. We could start by analyzing the influence of different

ontology metrics provided by several works, such as [ABS06, TA07], to assess the robustness of our method regarding the maintenance or improvement of quality metrics in the resulting ontologies.

- We also want to further research on more different techniques to fill the possible semantic gaps generated by the extraction techniques used during the discovery and disambiguation. Currently, we use Scarlet and ProSÉ, but we want to evaluate other possibilities.

Regarding the Query Generation Process

- As we said before, the way that current keyword search systems behave have changed the way users express their keyword queries, and in the queries is quite usual to have redundant keywords. Although QueryGen is able to handle it automatically when the language has the appropriate operators (e.g., the *And* operator, being restrictive as it is, can help to get rid of redundant concepts), we could detect it before and use less words for the query generation, resulting in an initial smaller set of possible interpretations.

- Another possible research line would be how to map keywords to operators in the language, and include this information in the query generation process. We could force the usage of a particular operator, narrowing the interpretation space.

- We want to further research in the user interaction issue. In particular, we are planning to apply visual techniques to help the user to select their intended query, and perform massive tests with different kinds of final users to measure the semantic accuracy of our prototype. Moreover, we have to still analyse the possibility of providing a ranking even on the patterns presented to the user. To this end, we could use the n-grams provided by Google, but it is not clear how to address this issue without attaching our solution to a particular language.

- We also want to advance in the analysis of the modeling of languages with DLs, analyzing the expressiveness needed, and enabling semantic filters for different systems.

Relevant Publications Related to the Thesis

[BBIM13] Carlos Bobed, Fernando Bobillo, Sergio Ilarri, and Eduardo Mena. Answering Continuous Description Logic Queries: Managing Static and Volatile Knowledge in Ontologies. *International Journal on Semantic Web and Information Systems*, Under Review, 2013.

[BBMI13] Jorge Bernad, Carlos Bobed, Eduardo Mena, and Sergio Ilarri. A Formalization for Semantic Location Granules. *International Journal of Geographical Information Science*, 27(6):1090–1108, 2013.

[BEM12] Carlos Bobed, Guillermo Esteban, and Eduardo Mena. Ontology-Driven Keyword-Based Search on Linked Data. In *Proc. of the 16th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES'12), San Sebastián (Spain)*, pages 1899–1908. IOS Press, September 2012.

[BEM13] Carlos Bobed, Guillermo Esteban, and Eduardo Mena. Enabling Keyword Search on Linked Data Repositories: An Ontology-Based Approach. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 17(1):67–77, 2013.

[BIM10a] Carlos Bobed, Sergio Ilarri, and Eduardo Mena. Distributed Mobile Computing: Development of Distributed Applications Using Mobile Agents. In *Proc. of the 16th International Conference on Parallel and Distributed Computing (PDPTA'10), Las Vegas (Nevada, USA)*, pages 562–568. CSREA Press, July 2010.

[BIM10b] Carlos Bobed, Sergio Ilarri, and Eduardo Mena. Exploiting the Semantics of Location Granules in Location-Dependent Queries. In *Proc. of the 14th East-European Conference on Advances in Databases and Information Systems (ADBIS'10), Novi Sad (Serbia)*, pages 79–93. Springer LNCS, September 2010.

[BM10] Carlos Bobed and Eduardo Mena. Enhancing the Discovery of Web Services: A Keyword-Oriented Multiontology Reconciliation. In *Proc. of the 16th International Conference on Parallel and Distributed Computing (PDPTA'10), Las Vegas (Nevada, USA)*, pages 724–730. CSREA Press, July 2010.

[BM13] Carlos Bobed and Eduardo Mena. QueryGen: Semantic Keyword-Based Search on Heterogeneous Information Systems. *Web Semantics: Science, Services and Agents on the World Wide Web (Special Issue on Semantic Search)*, Under Review, 2013.

[BMT12] Carlos Bobed, Eduardo Mena, and Raquel Trillo. FirstOnt: Automatic Construction of Ontologies out of Multiple Ontological Resources. In *Proc. of the 16th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES'12), San Sebastián (Spain)*, pages 1909–1919. IOS Press, September 2012.

[BTMB08] Carlos Bobed, Raquel Trillo, Eduardo Mena, and Jorge Bernad. Semantic Discovery of the User Intended Query in a Selectable Target Query Language. In *Proc. of 7th International Conference on Web Intelligence (WI'08), Sydney (Australia)*, pages 579–582. IEEE Computer Society, December 2008.

[BTMI10] Carlos Bobed, Raquel Trillo, Eduardo Mena, and Sergio Ilarri. From Keywords to Queries: Discovering the User's Intended Meaning. In *Proc. of 11th International Conference on Web Information System Engineering (WISE'10), Hong Kong (China)*, pages 190–203. Springer LNCS, December 2010.

[IBM11] Sergio Ilarri, Carlos Bobed, and Eduardo Mena. An Approach to Process Continuous Location-Dependent Queries on Moving Objects with Support for Location Granules. *Journal of Systems and Software*, 84(8):1327–1350, 2011.

[ICBM09] Sergio Ilarri, Antonio Corral, Carlos Bobed, and Eduardo Mena. Probabilistic Granule-Based Inside and Nearest Neighbor Queries. In *Proc. of the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS'09), Riga (Latvia)*, pages 103–117. Springer LNCS, September 2009.

[IMB07] Sergio Ilarri, Eduardo Mena, and Carlos Bobed. Processing Location-Dependent Queries with Location Granules. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops (PerSys'07), Vilamoura, Algarve (Portugal)*, pages 856–866. Springer LNCS, November 2007.

[YBE+13] Roberto Yus, Carlos Bobed, Guillermo Esteban, Fernando Bobillo, and Eduardo Mena. Android Goes Semantic: DL Reasoners on Smartphones. In *Proc. of 2nd International Workshop on OWL Reasoner Evaluation (ORE'13), Ulm (Germany)*, pages 46–52. CEUR-WS, July 2013.

Bibliography

[ABC+02] B. Aditya, Gaurav Bhalotia, Soumen Chakrabarti, Arvind Hulgeri, Charuta Nakhe, Parag, and S. Sudarshan. BANKS: Browsing and Keyword Searching in Relational Databases. In *Proc. of 28th International Conference on Very Large Data Bases (VLDB'02), Hong Kong (China)*, pages 1083–1086. Morgan Kaufman, August 2002.

[ABS06] Harith Alani, Christopher Brewster, and Nigel Shadbolt. Ranking Ontologies with AKTiveRank. In *Proc. of 5th International Semantic Web Conference, (ISWC'06), Athens (Georgia, USA)*, pages 1–15. Springer LNCS, November 2006.

[ACD02] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proc. of 18th International Conference on Data Engineering (ICDE'02), San Jose (California, USA)*, pages 5–15. IEEE Computer Society, March 2002.

[Ala06] Harith Alani. Ontology Construction from Online Ontologies. In *Proc. of the 15th International World Wide Web (WWW'06), Edinburgh (Scotland, UK)*, pages 491–495. ACM, May 2006.

[ALSU07] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2007.

[ART95] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural Language Interfaces to Databases – An Introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

[BBIM13] Carlos Bobed, Fernando Bobillo, Sergio Ilarri, and Eduardo Mena. Answering Continuous Description Logic Queries: Managing Static and Volatile Knowledge in Ontologies. *International Journal on Semantic Web and Information Systems*, Under Review, 2013.

[BBMI13] Jorge Bernad, Carlos Bobed, Eduardo Mena, and Sergio Ilarri. A Formalization for Semantic Location Granules. *International Journal of Geographical Information Science*, 27(6):1090–1108, 2013.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Scheneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[BCP09] Alberto Belussi, Carlo Combi, and Gabriele Pozzani. Formal and Conceptual Modeling of Spatio-Temporal Granularities. In *Proc. of the 13th International Database Engineering & Applications Symposium (IDEAS'09), Cetraro, Calabria (Italy)*, pages 275–283. ACM, September 2009.

[BDG+11] Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Raquel Trillo-Lado, and Yannis Velegrakis. Keyword Search over Relational Databases: A Metadata Approach. In *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11), Athens (Greece)*, pages 565–576. ACM, June 2011.

[BEM12] Carlos Bobed, Guillermo Esteban, and Eduardo Mena. Ontology-Driven Keyword-Based Search on Linked Data. In *Proc. of the 16th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES'12), San Sebastián (Spain)*, pages 1899–1908. IOS Press, September 2012.

[BEM13] Carlos Bobed, Guillermo Esteban, and Eduardo Mena. Enabling Keyword Search on Linked Data Repositories: An Ontology-Based Approach. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 17(1):67–77, 2013.

[BG04] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, 2004. `http://www.w3.org/TR/rdf-schema/`, last accessed October 3, 2013.

[BG08] Eric Belden and Janis Greenberg. Oracle Database Object-Relational Developer's Guide 11g Release 1 (11.1), 2008. `http://docs.oracle.com/cd/B28359_01/appdev.111/b28371/toc.htm`, last accessed October 3, 2013.

[BGK06] Guntis Barzdins, Normunds Gruzitis, and Renars Kudins. Re-Engineering OntoSem Ontology Towards OWL DL Compliance. In *Proc. of the 7th Joint Conference on Knowledge-Based Software Engineering (JCKBSE'06), Tallinn (Estonia)*, pages 157–166. IOS Press, August 2006.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *Proc. of 18th International Conference on Data Engineering (ICDE'02), San Jose (California, USA)*, pages 431–440. IEEE Computer Society, March 2002.

[BIM10a] Carlos Bobed, Sergio Ilarri, and Eduardo Mena. Distributed Mobile Computing: Development of Distributed Applications Using Mobile Agents. In *Proc. of the 16th International Conference on Parallel and Distributed Computing (PDPTA'10), Las Vegas (Nevada, USA)*, pages 562–568. CSREA Press, July 2010.

[BIM10b] Carlos Bobed, Sergio Ilarri, and Eduardo Mena. Exploiting the Semantics of Location Granules in Location-Dependent Queries. In *Proc. of the 14th East-European Conference on Advances in Databases and Information Systems (ADBIS'10), Novi Sad (Serbia)*, pages 79–93. Springer LNCS, September 2010.

[BJ07] George Boolos and Richard Jeffrey. *Computability and Logic*. Cambridge University Press, 1974, 2007.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[BLK+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 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.

[BM10] Carlos Bobed and Eduardo Mena. Enhancing the Discovery of Web Services: A Keyword-Oriented Multiontology Reconciliation. In *Proc. of the 16th International Conference on Parallel and Distributed Computing (PDPTA'10), Las Vegas (Nevada, USA)*, pages 724–730. CSREA Press, July 2010.

[BM13] Carlos Bobed and Eduardo Mena. QueryGen: Semantic Keyword-Based Search on Heterogeneous Information Systems. *Web Semantics: Science, Services and Agents on the World Wide Web (Special Issue on Semantic Search)*, Under Review, 2013.

[BMT12] Carlos Bobed, Eduardo Mena, and Raquel Trillo. FirstOnt: Automatic Construction of Ontologies out of Multiple Ontological Resources. In *Proc. of the 16th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES'12), San Sebastián (Spain)*, pages 1909–1919. IOS Press, September 2012.

[Bor97] Willem Nico Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. CTIT, 1997.

[BP03] Satanjeev Banerjee and Ted Pedersen. Extended Gloss Overlaps as a Measure of Semantic Relatedness. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco (Mexico)*, pages 805–810. Morgan Kaufmann, August 2003.

[BTMB08] Carlos Bobed, Raquel Trillo, Eduardo Mena, and Jorge Bernad. Semantic Discovery of the User Intended Query in a Selectable Target Query Language. In *Proc. of 7th International Conference on Web Intelligence (WI'08), Sydney (Australia)*, pages 579–582. IEEE Computer Society, December 2008.

[BTMI10] Carlos Bobed, Raquel Trillo, Eduardo Mena, and Sergio Ilarri. From Keywords to Queries: Discovering the User's Intended Meaning. In *Proc. of 11th International Conference on Web Information System Engineering (WISE'10), Hong Kong (China)*, pages 190–203. Springer LNCS, December 2010.

[CHCX06] Ying Cai, Kien A. Hua, Guohong Cao, and Toby Xu. Real-Time Processing of Range-Monitoring Queries in Heterogeneous Mobile Databases. *IEEE Transactions on Mobile Computing*, 5(7):931–942, 2006.

[CHKS08] Bernardo Cuenca, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research*, 31(1):273–318, 2008.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proc. of the 9th Annual ACM Symposium on Theory of Computing (STOC'77), Boulder (Colorado, USA)*, pages 77–90. ACM, May 1977.

[Cod70] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cod71] Edgar F. Codd. A Data Base Sublanguage Founded on the Relational Calculus. In *Proc. of the 1971 ACM SIGFIDET Workshop on Data Description, Access and Control (SIGFIDET'71), San Diego (California, USA)*, pages 35–68. ACM, November 1971.

[CR12] Claudio Carpineto and Giovanni Romano. A Survey of Automatic Query Expansion in Information Retrieval. *ACM Computing Surveys*, 44(1):1–50, 2012.

[CSPC00] Justine Cassell, Joseph Sullivan, Scott Prevost, and Elizabeth F. Churchill. *Embodied Conversational Agents*. MIT Press, 2000.

[dBG+07] Mathieu d'Aquin, Claudio Baldassarre, Laurian Gridinoc, Sofia Angeletou, Marta Sabou, and Enrico Motta. Characterizing Knowledge on the Semantic Web with Watson. In

Proc. of 5th International Workshop on Evaluation of Ontologies and Ontology-based Tools (EON'07), Busan (South Korea), pages 1–10. CEUR-WS, November 2007.

[dSM06] Mathieu d'Aquin, Marta Sabou, and Enrico Motta. Modularization: A Key for the Dynamic Selection of Relevant Knowledge Components. In *Proc. of the 1st International Workshop on Modular Ontologies (WoMO'06), at ISWC'06, Athens (Georgia, USA)*. CEUR-WS, November 2006.

[DTS08] Hui Ding, Goce Trajcevski, and Peter Scheuermann. Efficient Maintenance of Continuous Queries for Trajectories. *Geoinformatica*, 12(3):255–288, 2008.

[EN11] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2011.

[FA11] Haizhou Fu and Kemafor Anyanwu. Effectively Interpreting Keyword Queries on RDF Databases with a Rear View. In *Proc. of the 10th International Semantic Web Conference (ISWC'11), Koblenz (Germany)*, pages 193–208. Springer LNCS, October 2011.

[FCOO12] André Freitas, Edward Curry, Joao Gabriel Oliveira, and Sean O'Riain. Querying Heterogeneous Datasets on the Linked Data Web: Challenges, Approaches, and Trends. *IEEE Internet Computing*, 16(1):24–33, 2012.

[FGA11] Haizhou Fu, Sidan Gao, and Kemafor Anyanwu. CoSi: Context-Sensitive Keyword Query Interpretation on RDF Databases. In *Proc. of 20th International World Wide Conference (WWW'11), Hydebarad (India) (Companion Volume)*, pages 209–212. ACM, March 2011.

[GdM09] Jorge Gracia, Mathieu d'Aquin, and Eduardo Mena. Large Scale Integration of Senses for the Semantic Web. In *Proc. of 18th International World Wide Web Conference (WWW'09), Madrid (Spain)*, pages 611–620. ACM, April 2009.

[GL06] Bugra Gedik and Ling Liu. MobiEyes: A Distributed Location Monitoring Service Using Moving Location Queries. *IEEE Transactions on Mobile Computing*, 5(10):1384–1402, 2006.

[GM03] Ramanathan Guha and Rob McCool. TAP: A Semantic Web Platform. *Computer Networks*, 42(5):557–577, 2003.

[GM08] Jorge Gracia and Eduardo Mena. Ontology Matching with CIDER: Evaluation Report for the OAEI 2008. In *Proc. of 3rd Ontology Matching Workshop (OM'08), Karlsruhe (Germany)*, pages 140–146. CEUR-WS, October 2008.

[GM09] Jorge Gracia and Eduardo Mena. Multiontology Semantic Disambiguation in Unstructured Web Contexts. In *Proc. of Workshop on Collective Knowledge Capturing and Representation (CKCaR'09), Redondo Beach (California, USA)*, September 2009.

[GMM03] Ramanathan Guha, Rob McCool, and Eric Miller. Semantic Search. In *Proc. of the 12th International World Wide Web Conference (WWW'03), Budapest (Hungary)*, pages 700–709. ACM, May 2003.

[GPFLC04] Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering*. Springer, 2004.

[Gru93] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gru95] Thomas R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1995.

[GSBM08] Rolf Grütter, Thomas Scharrenbach, and Bettina Bauer-Messmer. Improving an RCC-Derived Geospatial Approximation by OWL Axioms. In *Proc. of the 7th International Semantic Web Conference (ISWC'08), Karlsruhe (Germany)*, pages 293–306. Springer LNCS, October 2008.

[Gua98] Nicola Guarino. Formal Ontology and Information Systems. In *Proc. of the 1st International Conference on Formal Ontology in Information Systems (FOIS'98), Trento (Italy)*, pages 3–15. IOS Press, June 1998.

[HG01] Lynette Hirschman and Rob Gaizauskas. Natural Language Question Answering: The View from Here. *Natural Language Engineering*, 7(4):275–300, 2001.

[Hig03] Jeffrey Hightower. From Position to Place. In *2003 Workshop on Location-Aware Computing, Seattle (Washington, USA)*, pages 10–12. Springer, October 2003.

[HKP+12] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer (Second Edition), 2012. `http://www.w3.org/TR/owl-primer/`, last accessed October 3, 2013.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th International Conference of Knowledge Representation and Reasoning (KR'06), Lake District (UK)*, pages 452–457. AAAI Press, June 2006.

[HM85] Dennis Heimbigner and Dennis McLeod. A Federated Architecture for Information Management. *ACM Transactions on Information Systems*, 3(3):253–278, 1985.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proc. of 28th International Conference on Very Large Data Bases (VLDB'02), Hong Kong (China)*, pages 670–681. Morgan Kaufman, August 2002.

[HP12] Matthew Horridge and Peter F. Patel-Schneider. OWL 2: Web Ontology Language Manchester Syntax (Second Edition), 2012. `http://www.w3.org/TR/owl2-manchester-syntax`, last accessed October 3, 2013.

[HS07] Christian Hoareau and Ichiro Satoh. A Model Checking-Based Approach for Location Query Processing in Pervasive Computing Environments. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops (PerSys'07), Vilamoura, Algarve (Portugal)*, pages 866–875. Springer LNCS, November 2007.

[HS09] Christian Hoareau and Ichiro Satoh. From Model Checking to Data Management in Pervasive Computing: A Location-Based Query-Processing Framework. In *ACM International Conference on Pervasive Services (ICPS'09), London (UK)*, pages 41–48. ACM, July 2009.

[HSP13] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 Query Language, 2013. `http://www.w3.org/TR/sparql11-query/`, last accessed October 3, 2013.

[IBM11] Sergio Ilarri, Carlos Bobed, and Eduardo Mena. An Approach to Process Continuous Location-Dependent Queries on Moving Objects with Support for Location Granules. *Journal of Systems and Software*, 84(8):1327–1350, 2011.

[ICBM09] Sergio Ilarri, Antonio Corral, Carlos Bobed, and Eduardo Mena. Probabilistic Granule-Based Inside and Nearest Neighbor Queries. In *Proc. of the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS'09), Riga (Latvia)*, pages 103–117. Springer LNCS, September 2009.

[IMB07] Sergio Ilarri, Eduardo Mena, and Carlos Bobed. Processing Location-Dependent Queries with Location Granules. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops (PerSys'07), Vilamoura, Algarve (Portugal)*, pages 856–866. Springer LNCS, November 2007.

[IMI06] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. Location-Dependent Queries in Mobile Contexts: Distributed Processing Using Mobile Agents. *IEEE Transactions on Mobile Computing*, 5(8):1029–1043, 2006.

[IMI10] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. Location-Dependent Query Processing: Where We Are and Where We Are Heading. *ACM Computing Surveys*, 42(3):1–73, 2010.

[ISO11a] ISO/IEC. ISO/IEC 13249-3:2011 Standard, "Information Technology - Database Languages - SQL Multimedia and Application Packages – Part 3: Spatial", 2011.

[ISO11b] ISO/IEC. ISO/IEC 9075:2011 Standard, "Information Technology - Database Languages - SQL", 2011.

[JCS+08] Ernesto Jimenez, Bernardo Cuenca, Ulrike Sattler, Thomas Schneider, and Rafael Berlanga. Safe and Economic Re-Use of Ontologies: A Logic-Based Methodology and Tool

Support. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Tenerife (Spain)*, pages 185–199. Springer LNCS, June 2008.

[KB10] Esther Kaufmann and Abraham Bernstein. Evaluating the Usability of Natural Language Query Languages and Interfaces to Semantic Web Knowledge Bases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):377–393, 2010.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.

[LM01] Ora Lassila and Deborah McGuinness. The Role of Frame-Based Representation on the Semantic Web. *Linköping Electronic Articles in Computer and Information Science*, 6(5), 2001.

[LUCM13] Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. Evaluating Question Answering Over Linked Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, page To appear, 2013.

[LUM06] Yuangui Lei, Victoria S. Uren, and Enrico Motta. SemSearch: A Search Engine for the Semantic Web. In *Proc. of 15th International Conference on Knowledge Engineering and Knowledge Management (EKAW'06), Podebrady (Czech Republic)*, pages 238–245. Springer LNCS, October 2006.

[LUSM11] Vanessa Lopez, Victoria S. Uren, Marta Sabou, and Enrico Motta. Is Question Answering Fit for the Semantic Web?: A Survey. *Semantic Web - Interoperability, Usability, Applicability*, 2(2):125–155, 2011.

[MI01] Eduardo Mena and Arantza Illarramendi. *Ontology-Based Query Processing for Global Information Systems*. Kluwer Academic Publishers, 2001.

[Mil95] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[MM04] Frank Manola and Eric Miller. RDF Primer, 2004. `http://www.w3.org/TR/rdf-primer/`, last accessed October 3, 2013.

[MMZ09] Peter Mika, Edgar Meij, and Hugo Zaragoza. Investigating the Semantic Gap through Query Log Analysis. In *Proc. of the 8th International Semantic Web Conference (ISWC'09), Chantilly (Virginia, USA)*, pages 441–455, October 2009.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In *Proc. of the 21st Conference on Automated Deduction (CADE-21), Bremen (Germany)*, pages 67–83. Springer, July 2007.

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36(1):165–228, 2009.

[MVI95] Riichiro Mizoguchi, Johan Vanwelkenhuysen, and Mitsuru Ikeda. *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*, chapter Task Ontology for Reuse of Problem Solving Knowledge, pages 46–59. IOS Press Amsterdam, 1995.

[MXHA05] Mohamed F. Mokbel, Xiaopeng Xiong, Moustafa A. Hammad, and Walid G. Aref. Continuous Query Processing of Spatio-Temporal Data Streams in PLACE. *Geoinformatica*, 9(4):343–365, 2005.

[Pel91] Christof Peltason. The BACK System – An Overview. *ACM SIGART Bulletin*, 2(3):114–119, 1991.

[PG88] Raymond C. Perrault and Barbara J. Grosz. *Exploring Artificial Intelligence*, chapter Natural-Language Interfaces, pages 133–172. Morgan Kaufmann, 1988.

[PXK+02] Sunil Prabhakar, Yuni Xia, Dmitri V. Kalashnikov, Walid G. Aref, and Susanne E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Transactions on Computers*, 51(10):1124–1140, 2002.

[RCC92] David A. Randell, Zhan Cui, and Anthony G. Cohn. A Spatial Logic Based on Regions and Connection. In *Proc. of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Cambridge (Massachusetts, USA)*, pages 165–176. Morgan Kaufmann, October 1992.

[RS06] Wararat Rungworawut and Twittie Senivongse. Using Ontology Search in the Design of Class Diagram from Business Process Model. In *Proc. of International Conference on Computer Science (ICCS'06), Vienna (Austria)*, pages 165–170, March 2006.

[SBF98] Rudi Studer, Richard Benjamins, and Dieter Fensel. Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering*, 25(1-2):161–197, 1998.

[SDK01] Ayse Y. Seydim, Margaret H. Dunham, and Vijay Kumar. Location Dependent Query Processing. In *Proc. of 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDe'01), Santa Barbara (California, USA)*, pages 47–53. ACM, May 2001.

[SdM08] Marta Sabou, Mathieu d'Aquin, and Enrico Motta. SCARLET: SemantiC relAtion discoveRy by harvesting onLinE onTologies. In *Proc. of the 5th European Semantic Web Conference (ESWC'08), Tenerife (Spain)*, pages 854–858. Springer LNCS, June 2008.

[SHBL06] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[SL90] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[SLHA12] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. LinkedGeoData: A Core for a Web of Spatial Open Data. *Semantic Web - Interoperability, Usability, Applicability*, 3(4):333–354, 2012.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A Practical OWL-DL Reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.

[Sto03] Knut Stolze. SQL/MM Spatial – The Standard to Manage Spatial Data in a Relational Database System. In *Proc. of BTW-Konferenz (BTW'03), Leipzig (Germany)*, pages 247–264. GI, February 2003.

[SWCD97] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and Querying Moving Objects. In *Proc. of the 13th International Conference on Data Engineering (ICDE'97), Birmingham (UK)*, pages 422–432. IEEE Computer Society, April 1997.

[TA07] Samir Tartir and I. Budak Arpinar. Ontology Evaluation and Ranking Using OntoQA. In *Proc. of the 1st International Conference on Semantic Computing (ICSC'07), Irvine (California, USA)*, pages 185–192. IEEE Computer Society, September 2007.

[TB83] Marjorie Templeton and John D. Burger. Problems in Natural-Language Interface to DBMS with Examples from EUFID. In *Proc. of the 1st Conference on Applied Natural Language Processing (ANLP'83), Santa Monica (California, USA)*, pages 3–16, February 1983.

[TGEM07] Raquel Trillo, Jorge Gracia, Mauricio Espinoza, and Eduardo Mena. Discovering the Semantics of User Keywords. *Journal on Universal Computer Science*, 13(12):1908–1935, 2007.

[The13] The PostgreSQL Global Development Group. PostgreSQL Documentation, 2013. `http://www.postgresql.org/docs/manuals/`, last accessed October 3, 2013.

[THL11] Thanh Tran, Daniel M. Herzig, and Günter Ladwig. SemSearchPro: Using Semantics Throughout the Search Process. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4):349–364, 2011.

[TIM07] Raquel Trillo, Sergio Ilarri, and Eduardo Mena. Comparison and Performance Evaluation of Mobile Agent Platforms. In

Proc. of 3rd International Conference on Autonomic and Autonomous Systems (ICAS'07), Athens (Greece), pages 41–46. IEEE Computer Society, June 2007.

[TWRC09] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In *Proc. of the 25th International Conference on Data Engineering (ICDE'09), Shangai (China)*, pages 405–416. IEEE Computer Society, March 2009.

[VBVTS+10] Luis M. Vilches-Blázquez, Boris Villazón-Terrazas, Victor Saquicela, Alexander de León, Oscar Corcho, and Asunción Gómez-Pérez. GeoLinked Data and INSPIRE through an Application Case. In *Proc. of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'10), San Jose (California, USA)*, pages 446–449. ACM, November 2010.

[vHSW97] Gertjan van Heijst, Guus Schreiber, and Bob J. Wielinga. Using Explicit Ontologies in KBS development. *International Journal of Human-Computer Studies*, 46(2):183–292, 1997.

[VP97] Vasilis Vassalos and Yannis Papakonstantinou. Describing and Using Query Capabilities of Heterogeneous Sources. In *Proc. of 23rd International Conference on Very Large Data Bases (VLDB'97), Athens (Greece)*, pages 256–265. Morgan Kaufmann, August 1997.

[WPH06] Taowei David Wang, Bijan Parsia, and James A. Hendler. A Survey of the Web Ontology Landscape. In *Proc. of 5th International Semantic Web Conference (ISWC'06), Athens (Georgia, USA)*, pages 682–694. Springer LNCS, November 2006.

[WZL+08] Haofen Wang, Kang Zhang, Qiaoling Liu, Duc Thanh Tran, and Yong Yu. Q2Semantic: A Lightweight Keyword Interface to Semantic Search. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Tenerife (Spain)*, pages 584–598. Springer LNCS, June 2008.

[YBE+13] Roberto Yus, Carlos Bobed, Guillermo Esteban, Fernando Bobillo, and Eduardo Mena. Android Goes Semantic: DL Reasoners on Smartphones. In *Proc. of 2nd International Workshop on OWL Reasoner Evaluation (ORE'13), Ulm (Germany)*, pages 46–52. CEUR-WS, July 2013.

[ZWX+07] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. SPARK: Adapting Keyword Query to Semantic Search. In *Proc. of 6th International Semantic Web Conference (ISWC'07), Busan (South Korea)*, pages 687–700. Springer LNCS, November 2007.

[ZZM+09] Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. From Keywords to Semantic Queries – Incremental Query Construction on the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):166–176, 2009.